Enterprise modeling advantages of San Francisco for general ledger systems

by E. E. Inman

With the advent of San Francisco™, object technology can seriously be considered for commercial enterprise applications. Much more work needs to be done in explaining why object technology will be important to business users. In accounting, for example, objects—and San Francisco frameworks in particular—provide elegant solutions to some of the problems encountered in conventional accounting information systems, particularly in the general ledger area. They also support an approach for generalizing accounting systems, allowing them to become models of the business enterprise rather than merely systems of accounts, ledgers, and journals. Such systems will support a much wider spectrum of management and analysis needs than conventional systems.

As object technology begins to meet the demands of commercial enterprise application environments, there needs to be a clearer picture of what it can do for users in the business world. This picture is currently obscured by the abstract nature of the technology and by the fact that its impact is felt more directly by software developers than by end users. Some of the general benefits are often touted, such as the broad reusability of business objects in diverse contexts, the reconfigurability of object-oriented systems into new solutions, and the lower maintenance costs. For the user these benefits translate to earlier product availability and higher quality for less cost. These general benefits, however, need to be supported by specific examples of what users will gain. This paper will illustrate the use of object technology in an enterprise application, specifically in the general ledger portion of an accounting information system (AIS). These observations will be based on what has been developed so far in IBM's San Francisco* frameworks.

IBM's San Francisco project

The purpose of the San Francisco project is to make it possible for application developers to take advantage of the benefits of distributed objects without having first to develop all of the underlying infrastructure necessary to support object-oriented applications.[1] The San Francisco frameworks supply not only the base set of distributed object infrastructure, but also much of the common application logic that can be shared among applications and among different providers of applications. This allows the resources of application developers to be reserved for the high-level features of the applications, where competitive discrimination can take place.

The frameworks have three layers. The lowest layer, or base, provides a distributed object environment that runs on multiplatform networks. Above the base is the common business objects layer. These business objects are commonly used in a variety of applications, and they also facilitate interoperability be-

0018-8670/98/$5.00 © 1998 IBM

tween applications. The top layer of the frameworks is the core business processes layer, which provides the basic and default business logic for various "vertical" domains, such as general ledger, accounts payable, accounts receivable, warehouse management, order management, etc. Applications can be built on top of the core business processes, fleshing them out into complete applications, or they can be built directly on top of the common business objects layer or the base.

General ledger

An accounting system, whether manual or automated, records the flow of value through an enterprise. Values are measured in terms of cash and are classified as to whether they represent assets of the enterprise or claims against those assets in the form of liabilities or owners' equity, where the total value of the assets matches the sum of the liabilities and owners' equity. Further classification identifies various categories and subcategories of assets, liabilities, and equity, such as cash and accounts receivable, short- and long-term debt, capital and retained earnings, etc. Additional classification might divide values according to different divisions within the enterprise, different parties with whom the enterprise does business, different products produced by the enterprise, etc. Each type of value is represented by an *account*. Changes in values over various time periods are also classified into a system of revenue and expense accounts. Two of the primary outputs of an accounting system are the *balance sheet* and the *income statement*, which show the main accounts of the system and the values they represent.

Financial *transactions* are initially recorded chronologically in a *journal*, listing each of the accounts involved and the change in value that occurred for each of those accounts, plus additional information describing the transaction. All transactions satisfy the constraint that any change in the total value of assets is matched by a corresponding change in the total value of liabilities and owners' equity, thus keeping the system balanced and giving rise to the convention of double-entry bookkeeping. Various types of commonly recurring transactions are typically handled by specialized journals that streamline the record keeping for the given type of transaction. Examples include special journals for cash, accounts receivable, accounts payable, fixed assets, payroll, etc.

A *ledger* shows a list of accounts and the value changes that occurred in each account. All transactions appearing in a ledger were first recorded in a journal. The ledger containing the accounts appearing in the balance sheet and income statement is called the *general ledger.*

A *general ledger system* is the portion of an AIS that handles the chart of accounts, the general ledger, and a basic set of journals. Specialized journals and subsidiary ledgers are typically handled by other applications within the AIS, such as those for accounts payable, accounts receivable, payroll, etc., and these applications periodically transmit to the general ledger system summaries of the financial data that they maintain. The general ledger system, then, is the one place where all of the financial data of the enterprise are represented, at least in summary form.

In complex organizations, the general ledger system is relied on not only as a provider of a standard set of financial reports, but also as a tool that allows the financial data to be used for various analyses of the enterprise and its operations. Questions such as those regarding operating efficiency, product costing, etc., often require various segments of the historical financial data for their answer, giving the general ledger system a very important role in supporting the management of the enterprise.

Improving the model

If one were to examine many of the existing commercial AISs and from these derive the purpose of accounting, one might conclude that it is to maintain a chart of accounts and a set of journals and ledgers in order to produce a balance sheet, income statement, and other financial reports. More accurately, the purpose of accounting is to record information about the operations of an enterprise and its environment, and then to make relevant views of that information available to decision makers at the appropriate time. Accounts, journals, ledgers, etc., are tools that evolved when accounting was done manually.

Starting as early as the 1960s, accounting researchers have been seeking ways to use computers to extend "the conventional accounting model to accommodate a broader spectrum of management information needs" and "to rethink some of the basic constructs of traditional double-entry bookkeeping."[2] Researchers recognized the need to have data that supported a wider variety of decision models

and were free of the layer of interpretation imposed by conventional accounting.[3]

Research on extended accounting models has focused heavily on *events accounting*, introduced by Ijiri[4] and Sorter.[3] Amer et al.[5] state that events accounting research resulting from Sorter is the largest single body of research in the AIS discipline. Research in object-oriented AIS (OOAIS) is also focused on events accounting. Many of the highlights of this research are traced by Adamson and Dilts.[6] The gist has been to find a system that models an enterprise and its operations rather than merely modeling the artifacts of accounting. What kind of sculptor would create a three-dimensional likeness of a two-dimensional portrait, rather than of the real person? Accounting researchers are hoping AISs will eventually model the enterprise directly instead of merely modeling it through the conventional accounting model.

The San Francisco frameworks supply the basic business objects and the distributed environment to support them.

Integration of events accounting with traditional AISs has progressed through a number of modeling approaches. First was data modeling from hierarchical[7] and relational[8] standpoints. Then came entity-relationship modeling[9] and finally object modeling.[10–14] Along the way, McCarthy[15] proposed a generalized accounting framework called the REA (resources, events, and agents) accounting model, where activities are modeled in terms of economic resources, economic events, and "inside" and "outside" economic agents.[16] The collection of event/resource/agent tuples, such as implemented in the prototype by Kandelin and Lin,[17] directly models all varieties of the tangible components of the enterprise and its operations without stripping away information that does not fit in conventional accounting models.

An events accounting system must still provide many of the outputs of a conventional accounting system, including reports showing figures that comply with generally accepted accounting principles, but this information must be derived from the events-oriented data while at the same time preserving those data for many other uses.

In examining a hierarchy of enterprise information systems, McCarthy et al.[18] point out that economic tracking systems need to be supplemented by an organizing rationale. For traditional accounting systems that rationale is the equation, "assets = liabilities + owners' equity." What is more desirable, however, is a model centered around the "chain" of value creation within the enterprise. Separated accounting, manufacturing, distribution, and marketing applications, with minimal integration between them, impede this view.

Activity-based costing (ABC) and activity-based management (ABM) conventions[19] mitigate some of the distorting effects that traditional transaction tracking and costing schemes have on enterprise models, but they lack full value-chain organization and analysis. Some enterprise resource planning (ERP) systems contain enterprise value-chain models but are monolithic and inflexible.[18]

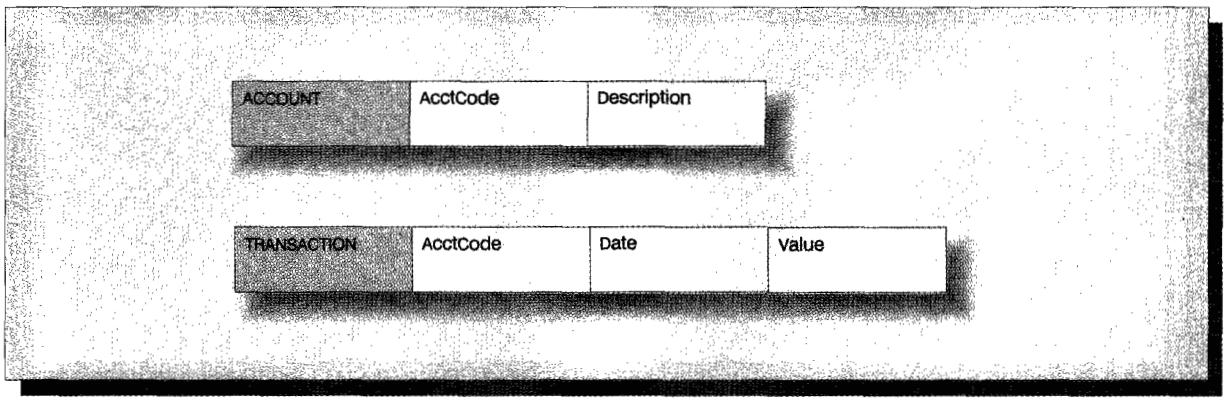
Commercially available AISs are, for the most part, based on relational database technology. In some cases, a move to OOAISs is underway. Business objects are needed in order to implement an enterprise value-chain perspective that can be easily adapted to individual enterprises and their various ways of doing business.

The San Francisco frameworks supply the basic business objects and the distributed object environment required to support them. The San Francisco *general ledger* core business process provides the basis for a general ledger system that can be much more flexible and adaptable in modeling an enterprise from a value-chain perspective. These features will be illustrated by examining the new levels of flexibility found in the San Francisco chart of accounts, and also by examining how financial transaction data are modeled when they are associated with some of the nonfinancial data that are part of the enterprise model. The San Francisco chart of accounts is especially interesting because of its ability to be defined in reference to other business objects within the model.

The chart of accounts

The chart of accounts and the system of account codes have been highly problematic for developers

Figure 1 Simple account code



of financial applications, because users need to have the diverse structures found in complex enterprises reflected in the structure of the chart of accounts. The capabilities of object technology have added new levels of flexibility to that typically available through relational database (RDB) technology.

The basic idea of a chart of accounts is very simple. Table 1 shows an example of a simple chart of accounts. It is simply a list of accounts, each identified by an account code, which is sometimes referred to as the account number. For much of the processing in an accounting system, no further structure is required. Using RDB technology, this structure is easily implemented with an account table using the account code as the primary key. The account code is represented as a character string of a limited length. Some accounting systems for small operations allow as few as four digits or characters in the account code. Figure 1 shows a relational schema for account and transaction tables under this approach.

Even for small operations, the structure of the chart of accounts is more than a flat list. The accounts are grouped into the categories of assets, liabilities, equity, revenues, and expenses. They are often further subdivided into smaller categories at different levels of detail. The structure is really a hierarchy, even though the system, for the most part, sees it as a flat list. The hierarchy may be reflected merely in the values of the account codes, with account codes having the same prefix being in the same group. Summary or heading accounts are also included. The reporting system then uses these cues to reflect the account hierarchy in printed output.

Larger operations require charts of accounts with more accounts, more categories, and more levels in

Table 1 Example of a chart of accounts

1000	Assets
1100	Current Assets
1110	Cash in Bank
1120	Petty Cash
1130	Inventory
1140	Accounts Receivable
1200	Fixed Assets
2000	Liabilities
2100	Current Liabilities
2110	Accounts Payable
2120	Tax Payable
2130	Short-Term Loans Payable
2200	Long-Term Liabilities
2210	Fixed Assets Payable
2220	Long-Term Debt
3000	Owners' Equity
3100	Capital
3200	Drawings
3300	Contributions
4000	Revenue
4100	Sales
4200	Interest
5000	Expenses
5100	Direct Expenses
5110	Materials
5120	Labor
5200	Overhead
5210	Salaries
5220	Utilities
5230	Rent

Figure 2 Segmented account code

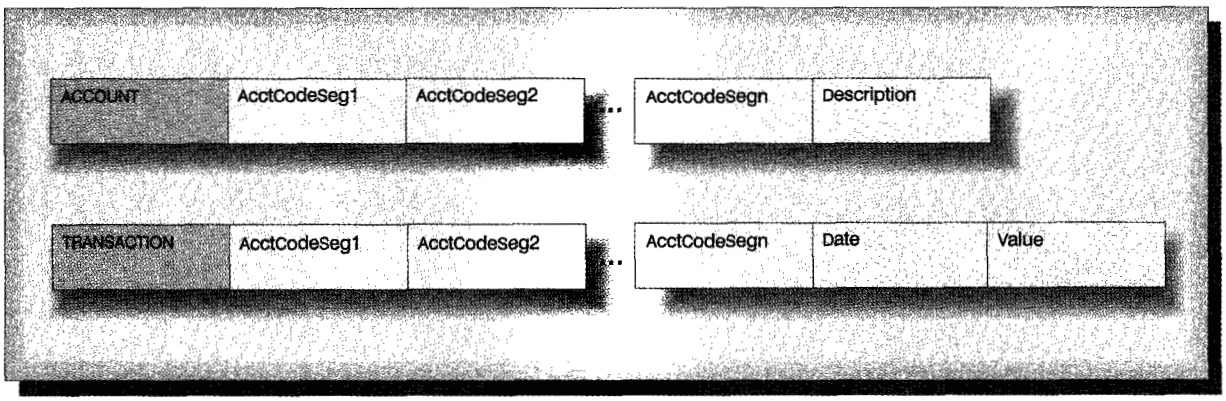
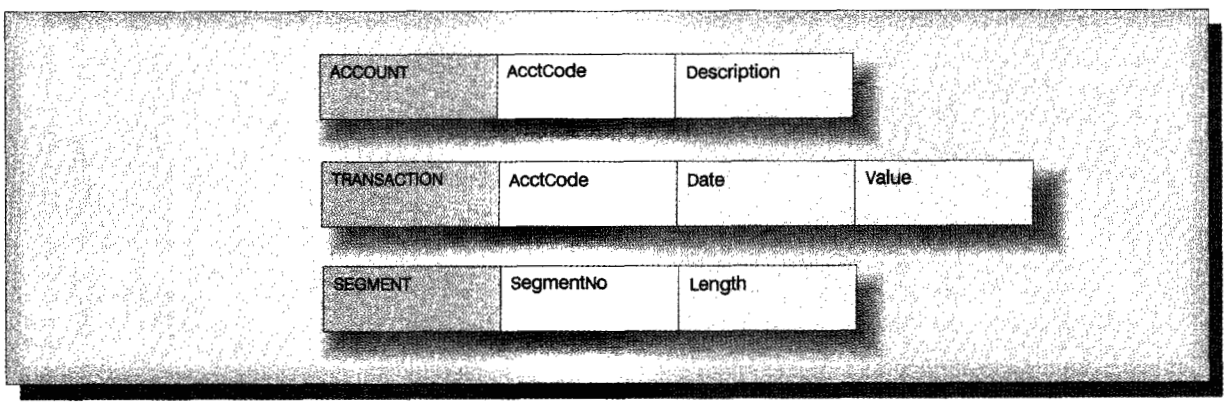


Figure 3 Segmented account code, number of segments, and segment lengths variable between charts of accounts, but not between accounts within the same chart

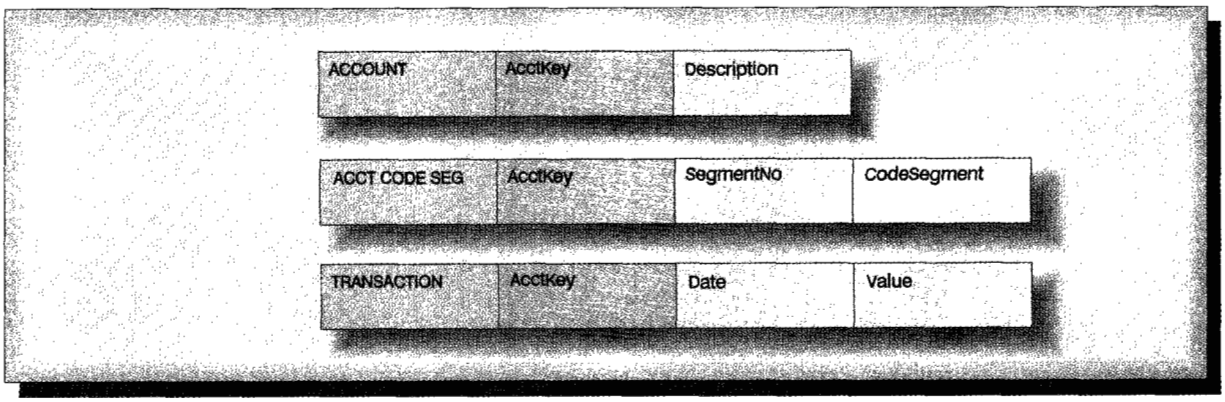


the hierarchy, resulting in the need for longer account codes. Eventually account codes become so long that segmentation is desired. An account code might be broken up into three segments, one for a company code, one for a cost center code, and the last for an account number. Each of these segments might allow its own hierarchy of values. Figure 2 shows a relational schema for this approach. Segmentation not only makes long account codes easier to handle for the user, but also allows certain segments—those for company code, for example—to be validated against other data in the system, such as the list of companies.

Two problems arise at this point with respect to RDB technology: code lengths and differing segmentation requirements. Field lengths in general are a problem with standard RDB technology. All fields, such as account codes or account code segments, must have a uniform length and that length must be large enough for the longest value to be stored. In cases where most values will be relatively short but some will be very long, a great deal of space will be wasted. When designing a packaged business application, the field length must satisfy not only a single enterprise, but all the enterprises in the targeted market.

With object technology, however, field lengths are not an issue. Different objects can have different lengths for a given value without producing a lot of empty space. Although with RDB technology there are ways to ameliorate the field length issue, this is

Figure 4 Segmented account code, number of segments, and segment lengths variable between accounts, surrogate key required



an example of something that is easy with object technology but very difficult with RDB technology.

Even with an issue as trivial as field lengths, object technology provides a benefit that affects end users. When implementing an object-oriented system, it is no longer necessary to agonize over the absolute maximum length that will be required for a given field, nor is there any fear of eventually encountering data that must be entered into the system but will not fit inside the maximum field length.

The second problem that arises with RDB technology is the problem of differing segmentation requirements. One company might want account codes divided into three segments. Another company might want four segments, or three segments but with lengths different from those of the first company. Some accounting applications have predefined segmentation: a fixed number of segments with predetermined lengths for each segment. An enterprise already using a different segmentation scheme would have to either change its system account codes or do extensive customization of the accounting application.

A more advanced implementation of account code segmentation allows the segmentation to be user-defined. The user can determine the number of segments and the length of each. Figure 3 shows a relational schema of the account, transaction, and segment tables using this approach. The last table simply contains a list of segment lengths that define how the account codes stored in the account and transaction tables are segmented. All account codes are segmented in the same way.

Even with user-defined segmentation, existing implementations require that the same segmentation scheme be used for all accounts. It may be desirable, however, for different accounts to have different segmentation. The sales division of a company, for example, may want to use account segments to track sales by product, region, and vertical market segment, while the manufacturing division may want to track certain expenses by plant and production unit. Rather than try to define a fixed segmentation scheme that satisfies such varied requirements, it is better to vary the segmentation by account.

Figure 4 shows a diagram for an account table, a transaction table, and an account code segment table in which each account can have its own segmentation scheme. One of the penalties for this is that the account code no longer serves as a key in the tables. Instead, a surrogate key is required, referred to as the account key in this case. The surrogate key, unknown to the user, is created "behind the scenes" by the system as a way to tie an account to its variable number of account code segments. The surrogate key is used in the account and transaction tables in place of the account code, and in the account code segment table it is used along with segment numbers to index the various segments of the account code. At this point, however, the account information is no longer accessible by the account code using conventional querying methods, since the account code now occupies several rows in the table that links

Figure 5 The primary San Francisco classes for the chart of accounts

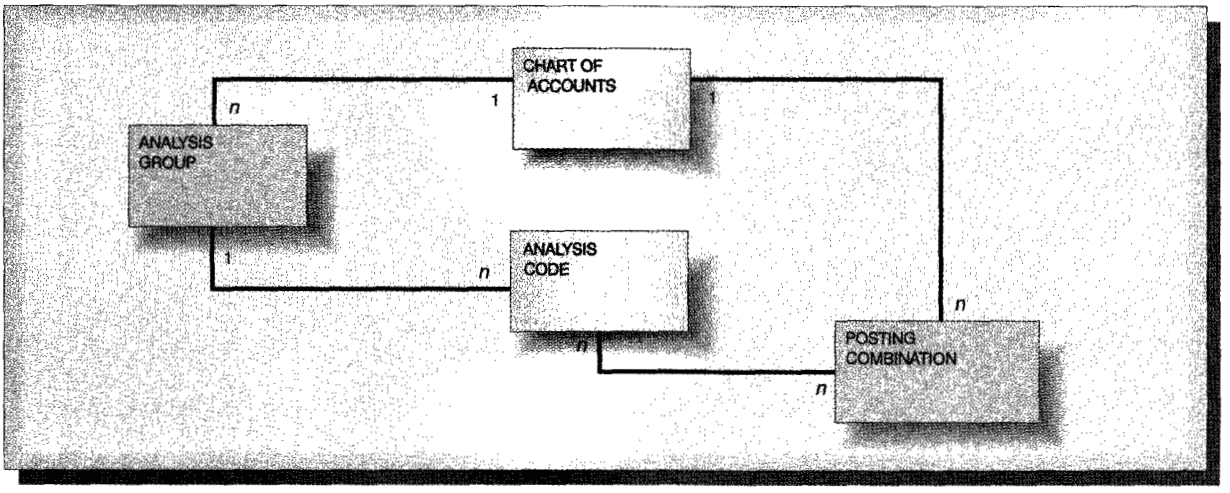
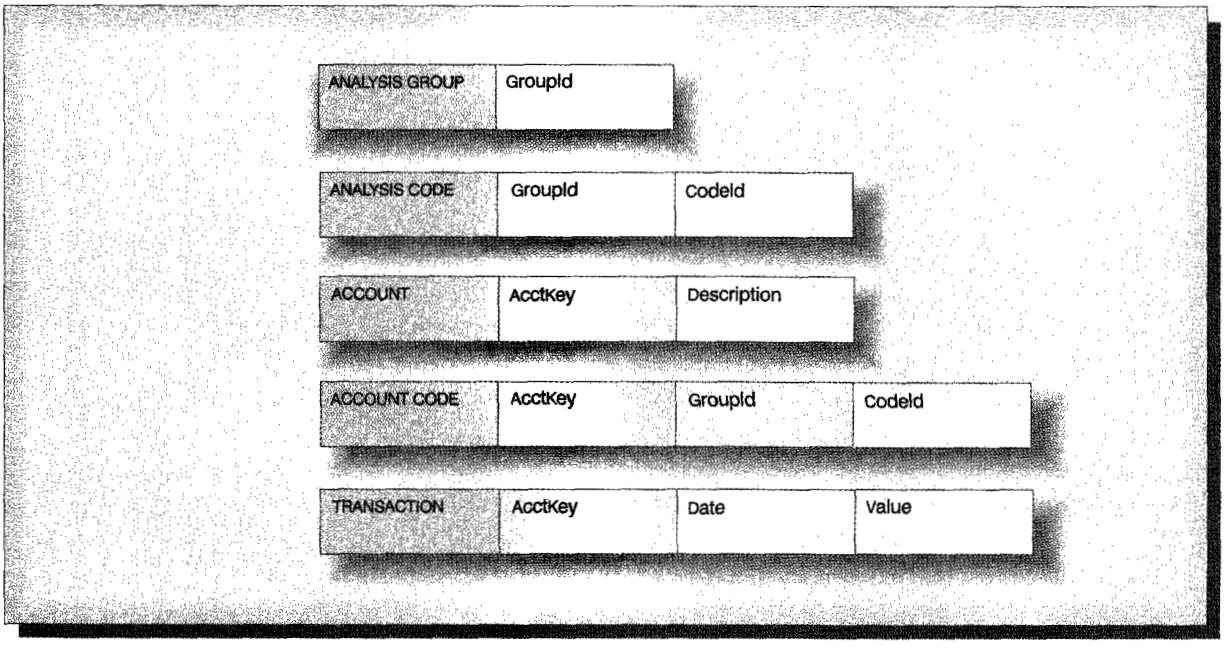


Figure 6 Segmented account code using analysis groups and codes as in San Francisco, number of segments and segment lengths variable between accounts, surrogate key required



it to the surrogate key. Thus RDB technology cannot support a fully flexible account code structure without sacrificing the account code as a convenient means of accessing account data.

The San Francisco implementation. These requirements for flexible account segmentation can be satisfied easily using objects, and this is the case under the San Francisco general ledger core business pro-

cess. Using San Francisco terminology, accounts are identified by *posting combinations*, representing the account codes discussed previously. A posting combination consists of a set of *analysis codes*, where each analysis code represents a segment of the account code.

Analysis codes are drawn from various *analysis groups*. One analysis group might contain an analysis code for each company, another group for departments, another for products, another for regions, etc. There is no limitation on the character length of posting combinations or analysis codes, nor on the number of analysis codes in a posting combination. Figure 5 shows a portion of the class diagram for this approach. A chart of accounts holds a number of analysis codes and a number of posting combinations. Each analysis code belongs to only one analysis group, but may be part of more than one posting combination. Figure 6 shows a relational schema supporting this approach, but again, surrogate keys are required. While this degree of flexibility is very difficult to achieve using RDB technology, it is merely a straightforward use of object technology.

To satisfy differing needs between sales and manufacturing, under a San Francisco implementation analysis groups could be set up for products, regions, vertical market segments, plants, and production units. The analysis group for products would contain an analysis code for each product, and the other analysis groups would follow the same pattern. A sales transaction would post to an account with a posting combination that included the analysis code for the product sold, the analysis code for the region where it was sold, and the analysis code for the vertical market segment that was represented by the customer. Certain manufacturing expense transactions would post to an account with a posting combination that included an analysis code for the plant and one for the production unit. Sample values for analysis groups and analysis codes are shown in Table 2. These can be combined to make up the posting codes shown in Table 3. This degree of flexibility actually pushes the idea of account coding to the background and replaces it with the idea of freely tagging each transaction with whatever items of information are relevant for its classification.

The use of objects for account code segmentation provides a great deal more than flexible formatting. In this case, each segment stands for more than just a series of characters. It "knows" about the underlying object that is represented by the code. Consider, for example, the account code "010-23-400-3800," made up of several segments encoded as numbers. The first segment might be the company number, the second the cost center number, the third

Table 2 Sample analysis groups and analysis codes

Analysis Group	Analysis Code
Main Account	1000 Assets 1100 Current Assets 1110 Cash in Bank ... 4100 Sales ... 5110 Material Expense ...
Product	P100 Hot Rolled Coil P200 Cold Rolled Coil P300 Cold Rolled Sheet P400 Pipe P500 Galvanized Sheet P600 Long Products
Region	R1 Eastern R2 Central R3 Western R4 International
Vertical Market Segment	M100 Automotive M200 Oil and Gas M300 Appliance M400 Building
Plant	F1 Northeast Works F2 Great Lakes Works F3 Gulf Works
Production Unit	U20 Tandem Cold Mill U50 Continuous Annealing Line U60 Temper Mill U70 Finishing Line

Table 3 Sample accounts (posting combinations)

Account Code	Description
4100-P300-R3-M100	Sales, Cold Rolled Sheet, West, Automotive
4100-P400-R2-M200	Sales, Pipe, Central, Oil & Gas
4100-P600-R1-M400	Sales, Long Products, East, Building
5110-F1-U20	Material Expense, Northeast, Tandem Cold Mill
5110-F2-U60	Material Expense, Great Lakes, Temper Mill
5110-F3-U70	Material Expense, Gulf, Finishing Line

the analysis code for "sales," and the fourth the product number. On seeing the account code displayed, a user might from time to time need help in remembering what some of the numbers refer to. The user might recall that "23" refers to a cost center, but cannot remember which one. As for "3800," the user might not be able to recall whether it refers to a product number or a sales region. Ideally the user could point to the segment in question and get a full description of what the code represents and what group it is a part of.

With San Francisco this feature can easily be implemented using *properties* of the analysis group and analysis code objects. Analysis codes can maintain a link to the business objects that they represent, and the business objects can be defined to support a common interface for supplying descriptive information. Many of the San Francisco business object classes, such as Company, BusinessPartner, and Area, are extensions of the base class Describable, which means that they provide a string description of themselves in response to the getDescription( ) method call. In the example just described, the account may need to show additional company information, cost center information, or product information, yet it is not necessary for the account or the general ledger to have facilities for obtaining and displaying these different types of information. The account merely requests the information from the desired account code segment, and the logic for making this request is the same regardless of which of the code segments is involved.

Thus the general ledger can provide access to the information and functions of business objects that are represented by the account code segments without containing its own logic to handle these different classes of business objects. In fact, it will support new classes of business objects that are defined after the general ledger application is completed. The close linkage of account code segments to the business objects that they represent has the effect of identifying an account less in terms of an encoded account number and more in terms of the set of business objects with which it is associated.

The chart of accounts also benefits from some of the design patterns used by the frameworks. Design patterns are used extensively throughout the frameworks. Some of them, such as Abstract Factory, Proxy, Chain of Responsibility, and Command, are described by Gamma et al.[20] Other authors, such as Fowler[21] and Hay,[22] have defined business patterns especially useful for accounting, and the San Francisco project has done this as well.[23] An interesting example is the framework's implementation of summary accounts through the use of cached balances and the Keyables design pattern.

A *cached balance* is an account total that is always kept up to date (as opposed to requiring a query over the journals each time its value is desired). Groups of cached balances are stored in cached balance sets, and the set of all cached balance sets is the cached balance set collection. Cached balances are selected, either singly or in sets, through the use of keys. Keys can also be used in other contexts.

There are two kinds of keys: access keys and specification keys. An access key selects a single item; a specification key selects a group of items. Keys are defined to examine a specified list of attributes, referred to as "keyables," when making the selection. For cached balances, keys can be used to examine the analysis codes of the accounts that are associated with the balances. Access keys specify a specific value for each keyable, for example, warehouse "5" and product code "1234." Specification keys specify various types of ranges or sets of values for each keyable, for example, all warehouses and product codes 1000 to 1999. Thus summary totals can be obtained for virtually any grouping of accounts, and an overlapping set of hierarchies can be supported as well.

Thus in a San Francisco implementation, the chart of accounts, defined by its collections of analysis groups and analysis codes, has much greater flexibility for reflecting diverse structures and relationships within the enterprise. More importantly, it arises naturally as a set of groupings of the business objects within the information system.

Combined financial activities

In a typical enterprise system with several integrated applications, different applications record different types of financial activities. The purchasing application records purchases, the payroll application records payroll disbursements, and so on. Each of these applications records the financial aspect of these activities as well as other information, depending on the type of activity. For example, for each purchase there is information regarding what was purchased and who the vendor was. For each payroll disbursement there is information identifying the employee and time period involved, and so on. At some point

these different applications feed all of the financial information (some of it summarized) to the general ledger. Once this is done, the general ledger can provide a view at a certain level of all financial activities within the enterprise, regardless of which applications originally recorded them. While this approach has made it possible to use RDB technology to handle several types of transactions in a single, integrated system, it has some shortcomings as well.

First of all, while the general ledger application provides access to all of the financial data, those data are not available until the originating applications feed the data to the general ledger. Once the data are in the general ledger, they are redundant, since the data now reside both with the originating application and in the general ledger. Users wanting access to cross sections of financial data that cross the boundaries of these different applications must be cognizant of the timing of the flow of the data into the general ledger. Search criteria that rely on more information than is transferred to the general ledger cannot be used. Furthermore, the complexity of the system is increased because of the requirement to maintain consistency between redundant sets of data. It would be better to have all financial activities stored in a single location and accessible for use in the general ledger as well as in the applications focusing on certain types of activities. This would be difficult with RDB technology but straightforward with object technology.

While RDB technology is strongly oriented toward uniformity, object technology supports diversity.

RDB technology is oriented toward uniform data. All the items in a table must be described by the same set of columns. If two types of activities are represented by two different tables, each including columns for financial data, processing the financial data from each table would require two different procedures, or at least two different versions of the same procedure. With objects, however, the financial activities represented in a collection can be different, with each activity containing financial data plus any additional information required by its activity type. Processing the financial data from each activity can be done in the same way regardless of the activity type. Thus while RDB technology is strongly oriented toward uniformity, object technology supports diversity. Elements can be grouped that have similarities while at the same time having significant differences.

In San Francisco, the commonality of transactions is provided by the Journal and Dissection classes, where the term "dissection" refers to the association of a value with an account. One or more transactions are represented by a journal, and the journal contains a dissection object for each posting to an account. The Dissection class can be as specialized as desired, so that a dissection object can contain additional information about the particular transaction that is involved. Because its class is an extension of the Dissection class, it will still be usable as a simple dissection by the general ledger implementation.

As the trend continues from managing vertically to managing horizontally,[19] financial information needs to be freed from its vertical orientation. It needs to be grouped according to business activities and processes without first being stripped of its auxiliary information by a conventional general ledger system. Addressing this problem eventually leads not only to refinements of conventional accounting, but also to a new underlying accounting model.

Conclusion

San Francisco frameworks and object technology in general have a lot to offer, not only to developers who are seeking more powerful ways of expressing their business solutions, but also to end users, who will see a new generation of applications that are more compliant to their view of their enterprises. One area where the frameworks provide these benefits is the chart of accounts, where the frameworks allow full flexibility in allowing account structures to exactly mirror the structure of a given enterprise and its information requirements, regardless of the number and variations in levels of complexity. The San Francisco general ledger frameworks also support tighter integration of financial data that originate from other applications within the system. The San Francisco frameworks will be a valuable tool for crafting information systems that not only track financial transactions but also provide a clear view of the value chain of an enterprise.

Acknowledgments

I would like to thank Joel Jorgenson, Marketing Financial Products Specialist, and Brad Preston, Tool Development Manager, of Lawson Software for their assistance in identifying some of the more challenging general ledger issues being encountered in today's market.

*Trademark or registered trademark of International Business Machines Corporation.

Cited references

1. V. D. Arnold, R. J. Bosch, E. F. Dumstorff, P. J. Helfrich, T. C. Hung, V. M. Johnson, R. F. Persik, and P. D. Whidden, "IBM Business Frameworks: San Francisco Project Technical Overview," *IBM Systems Journal* **36**, No. 3, 437–445 (1997).
2. W. E. McCarthy, "The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment," *The Accounting Review* **57**, No. 3, 554–578 (1982).
3. G. H. Sorter, "An 'Events' Approach to Basic Accounting Theory," *The Accounting Review* **44**, No. 1, 12–19 (1969).
4. Y. Ijiri, *The Foundations of Accounting Measurement*, Prentice-Hall, Inc., New York (1967).
5. T. Amer, A. D. Bailey, Jr., and P. De, "A Review of the Computer Information Systems Research Relating to Accounting and Auditing," *Journal of Information Systems* **1**, No. 2, 3–28 (1987).
6. I. L. Adamson and D. M. Dilts, "Development of an Accounting Object Model from Accounting Transactions," *Journal of Information Systems* **9**, No. 1, 43–64 (1995).
7. C. S. Colantoni, R. P. Manes, and A. Whinston, "A Unified Approach to the Theory of Accounting and Information Systems," *The Accounting Review* **46**, No. 1, 90–102 (1971); and A. Z. Lieberman and A. B. Whinston, "A Structuring of an Events-Accounting Information System," *The Accounting Review* **50**, No. 4, 246–258 (1975).
8. G. C. Everest and R. Weber, "A Relational Approach to Accounting Models," *The Accounting Review* **52**, No. 2 (1977).
9. W. E. McCarthy, "An Entity-Relationship View of Accounting Models," *The Accounting Review* **54**, No. 4 (1979); and W. E. McCarthy, "Construction and Use of Integrated Accounting Systems with Entity-Relationship Modeling," *Entity-Relationship Approach to Systems Analysis and Design*, P. P. Chen, Editor, New Holland, New York (1980).
10. C. Mui and W. E. McCarthy, "FSA: Applying AI Techniques to the Familiarization Phase of Financial Decision Making," *IEEE Expert* **2**, No. 3, 33–41 (1987).
11. N. A. Kandelin and T. W. Lin, "A Computational Model of an Events-Based Object-Oriented Accounting Information System for Inventory Management," *Journal of Information Systems* **6**, No. 1 (1992).
12. P. C. Chu, "An Object-Oriented Approach to Modeling Financial Accounting Systems," *Accounting, Management, and Information Technology* **2**, 39–56 (1992).
13. U. S. Murthy and C. E. Wiggins, Jr., "Object-Oriented Modeling Approaches for Designing Accounting Information Systems," *Journal of Information Systems* **7**, No. 2, 97–111 (1993).
14. I. L. Adamson and D. M. Dilts, "Development of an Accounting Object Model from Accounting Transactions," *Journal of Information Systems* **9**, No. 1, 43–64 (1995).
15. W. E. McCarthy, "The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment," *The Accounting Review* **57**, No. 3, 554–578 (1982).
16. G. Geerts and W. E. McCarthy, "Modeling Business Enterprises as Value-Added Process Hierarchies with Resource-Event-Agent Object Templates," in *Business Object Design and Implementation*, J. Sutherland and D. Patel, Editors, Springer-Verlag, Inc., New York (1996).
17. N. A. Kandelin and T. W. Lin, "A Computational Model of an Events-Based Object-Oriented Accounting Information System for Inventory Management," *Journal of Information Systems* **6**, No. 1 (1992).
18. W. E. McCarthy, J. S. David, and B. S. Sommer, "The Evolution of Enterprise Information Systems—From Sticks and Jars Past Journals and Ledgers Toward Interorganizational Webs of Business Objects and Beyond," Michigan State University, September 27, 1996, presented at OOPSLA'96 Workshop on Business Object Design and Implementation II, http://www.tiac.net/users/jsuth/oopsla96/mccarthy.html.
19. J. A. Miller, *Implementing Activity-Based Management in Daily Operations*, John Wiley & Sons, New York (1996).
20. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Co., Reading, MA (1995).
21. M. Fowler, *Analysis Patterns*, Addison-Wesley Publishing Co., Reading, MA (1997).
22. D. Hay, *Data Model Patterns: Conventions of Thought*, Dorset House, New York (1996).
23. K. Bohrer, "Architecture of the San Francisco Frameworks," *IBM Systems Journal* **37**, No. 2, 156–169 (1998, this issue).

Accepted for publication January 6, 1998.

Eric E. Inman *World Enterprise Software Corporation, 5871 Oxford Street, Shoreview, Minnesota 55126 (electronic mail: inman002@gold.tc.umn.edu).* Eric Inman, a software architect and development manager, received his B.A. degree in computer science from the University of Minnesota in 1980. He led the development of functional specifications for the orbital mechanics and radar imaging portions of a space object identification system for the North American Aerospace Defense Command (NORAD). He later operated a small business for developing business applications, primarily for cost accounting in the telecommunications industry. He then led the development of requirements analysis and human factors engineering tools used in FAA (Federal Aviation Administration) air traffic control system and NASA (National Aeronautics and Space Administration) space station projects. He then spent five years in Indonesia on the island of Java, where he led the development of a production management system for an integrated steel mill and also provided consulting and management assistance for a construction accounting system. At Lawson Software, a provider of business enterprise application software, he was first in charge of the CASE (computer-assisted software engineering) and repository tool development. He later accepted the position of manager of technology planning in order to examine new technologies to be used in Lawson Software products. Beginning in 1995 he became a member of the advisory group and later the reference group for the IBM San Francisco project, providing input to IBM on the requirements for a commercially viable distributed object environment and development platform for enterprise applications. In 1998 he founded World Enterprise Software Corporation to build enterprise applications using San Francisco frameworks.

Reprint Order No. G321-5670.