© 1998 Braided Matrix, Inc. All Rights Reserved

Accounting the OO & RDB Way, Part III: Application

Lincoln Stoller, Ph.D.

I. Introduction

This is the last of a three-part series about designing accounting applications using object oriented and relational database methods. My objective is to present an "architecture-driven" design for accounting software that is scalable, extensible, and reusable. The successful result described here is achieved by combining relational and object oriented design principles.

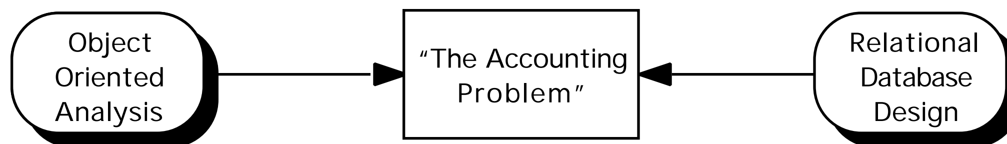


Figure 1. A successful architecture-driven application design requires both OO and RDB concepts.

In the previous articles I developed a diagrammatic method using boxes, brackets, and arrows whereby:

boxes	:	represent how the data is modularized,
brackets	:	represent how the code is modularized,
arrows	:	indicate how the code (brackets) affects the data (boxes)

Architecture-driven design results from the following three-step method for combining RDB and OO concepts:

1 — apply RDB concepts to identify the data stores (draw the boxes)
2 — apply OO concepts to identify the functional code modules (draw the brackets)
3 — connect the code modules to the data stores using arrows, whose heads indicate the direction in which the data moves (code that reads from and/or writes to data stores)

The current article applies this method to the problem of accounting.

© 1998 Braided Matrix, Inc. All Rights Reserved

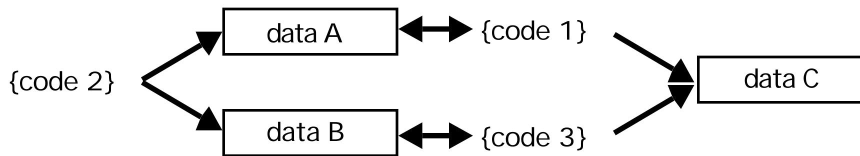


Figure 2. The general form of a design expressed using boxes, brackets, and arrows.

II. Handling the basics

Before applying the 3-step method, let us review the basic conditions of double-entry accounting, since these conditions must be satisfied by any accounting system. My objective is to satisfy them generally so that the software design can be used as a basis for any accounting-based business system. First some terminology:

• Data stores are categories used to identify data. Each data store is composed of one or more data files or tables. A data store is a logical entity used to visualize the larger relationships between data. A data file, on the other hand, is a relational concept used to design the physical structure of the database.

•Assets are things that have a value. The term "asset" refers to money, inventory, debts, and other financial obligations. In accounting everything is given a monetary value whether or not money actually changes hands.

•Transactions record the movement of assets between accounts. We can speak of assets moving from one account to another, but in truth the notions of "to" and "from" are ambiguous. Transactions actually debit and credit accounts. The actions of debiting and crediting are well defined.

• Accounts are categories that correspond to operations within or outside of the business. Accounts have a balance that represents the net amount of assets provided by, or supplied to this operation.

• Accounts also have a Type that identifies the kind of effect the operation has on the business. Accounting standards provide for a fixed set of account types. We'll include account types in our description even though we won't list the types or describe their effects. The actions of accounts of different types is completely determined by the rules of accounting.

© 1998 Braided Matrix, Inc. All Rights Reserved

•A balanced transaction is one that records an equal flow of assets out of some accounts and into others. More accurately, a balanced transaction is one whose total debits to accounts is equal to its total credits.

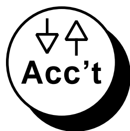
•Allocation is the process of matching debts with payments. In general, a payment might pay off various debts, or a given debt may be retired through multiple payments. The allocation process can take place as transactions are entered, or afterwards. Because transactions and allocations are different, their actions can be considered independently.

In section II-1 I'll describe an accounting system without mentioning allocations. In section II-2 I'll add allocations in a manner that builds on the existing design.

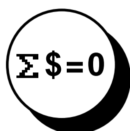
II-1. Double-entry accounting rules

The following three rules are the basis of double-entry accounting:

Transactions record where all assets come from and where they go.



Accounts are the source and destination of all assets.



The total assets coming into accounts in any transaction must equal the total assets going out. This is the balance condition.

The most general structure satisfying these rules consists of an account and a transaction data store. These two data stores are sufficient to meet the most complex accounting requirements.

Data stores

Accounts

- keep a record of a particular business function: sales, debt, payable, etc.

© 1998 Braided Matrix, Inc. All Rights Reserved

- have a name, code (usually a number), current balance, and an account type

Transactions

- record the movement of assets to and from one or more accounts
- have a heading portion that includes a date and description applying to all components
- are composed of 1 or more components
- each component refers to an account, has a due date (which may be left blank) and an amount (actually a debit amount that represents a credit when negative)

The following figure shows the relational file structure that underlies these data stores. The transaction data store is composed of two files to which a third will be added in the next section.

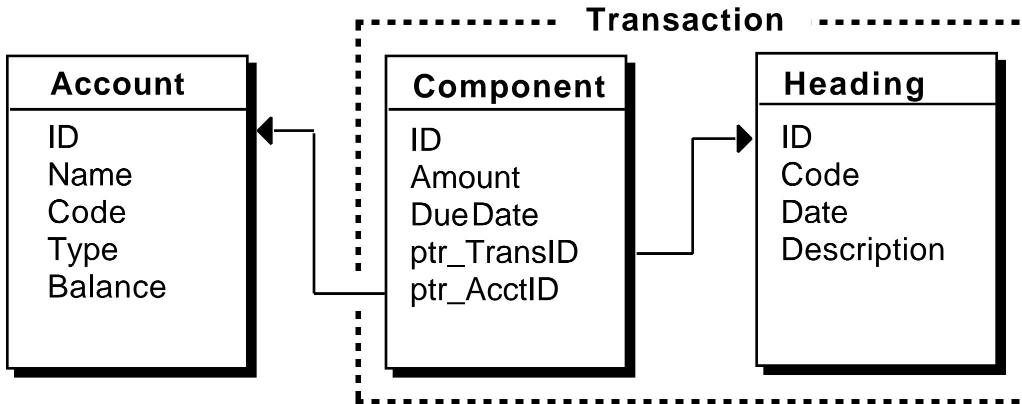


Figure 3. The general accounting file structure.

Procedures

The following actions need to be supported in order to use these data stores. The following items that appear in brackets represent actions that are segregated into groups. The effect of the actions within each group is largely constrained to affect only data in the corresponding data store.

{Accounts}
- create : prevent duplicates
- update : changes to code, name, balance
- delete : preserve history
- report : statement, trial balance, history

{Transactions}
- create : ensure that transactions balance, call account update

© 1998 Braided Matrix, Inc. All Rights Reserved

- update : check references, ensure balance, call account update
- delete : call account update
- report : information extracted according to value, date, account or account type.

The relation of data stores (boxes) to actions (brackets) is shown below.

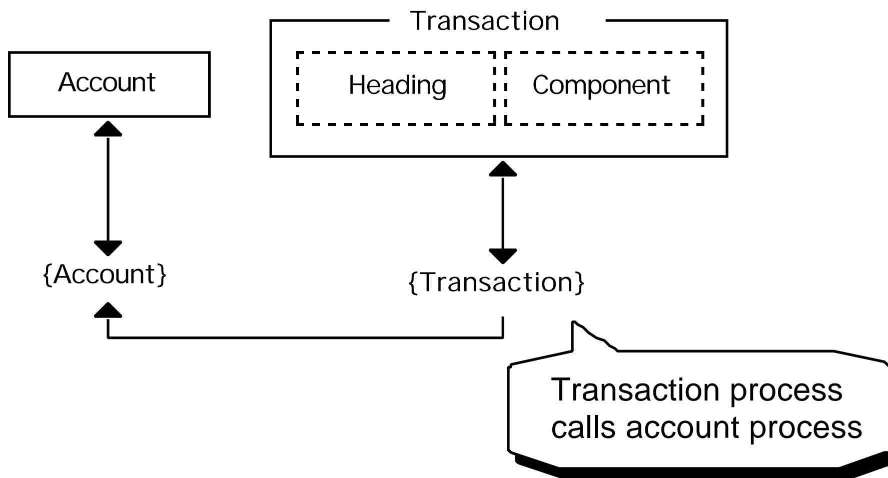


Figure 4. The code and data store elements that support the general accounting system.

II-2. Debts and payments

The accounting system presented thus far is adequate for giving a financial snapshot, but lacks an element essential for managing the ongoing affairs of any business. The account and transaction picture can record debts and payments, but has no means to match one with the other — a means for tracking financial obligations such as payables and receivables, and for matching such obligations with offsetting payments. That is, it lacks a mechanism for allocating funds.

A minor modification provides what is needed. This change is accomplished by adding a file I will refer to as "allocation." This file is added to the heading and component files, and becomes part of what we are calling the transaction data store.

The allocation file serves two functions. First, it acts as an intermediary file that records which payment components are used to pay off which debt components. Second, each allocation record stores the amount of the payment that is allocated to the debt. This is necessary because the

© 1998 Braided Matrix, Inc. All Rights Reserved

payment may only partially cover a debt. It isn't enough to know which components are related, we must also know the allocated amount .

- record the connection between a debt component and any contributing payment component
- record the amount to be applied from a given payment to the debt it pays.

These two functions are supported using three fields:
ID of debt component
ID of payment component
allocation amount

The component ID's are foreign key values. They refer to the primary key values of records in the components file.

The allocation amount is a positive value. We don't need to distinguish payment from debt allocations because the allocation record always links a payment with a debt.

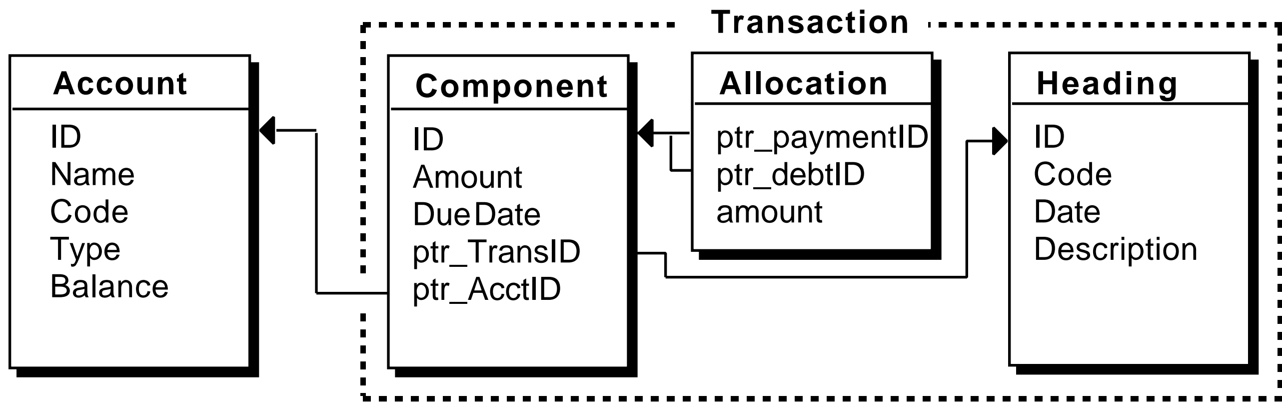


Figure 5. The file structure after the addition of the allocation file.

The addition of the allocation file requires additional transaction procedures in addition to those listed above.

{Transactions}
- allocate by account
  : match all available payments with current debts for a given account
- allocate by transaction
  : specify the portion of a payment that is to be allocated to a particular debt

© 1998 Braided Matrix, Inc. All Rights Reserved

- report : debts by account and age (aging report),
  debts and payments by account (statement)

Adding the allocation file and functions to the system previously described results in the system diagrammed below. While this addition changes the design only a little, it greatly increases the system's functionality. The addition is sufficient to support a complete accounts payable and receivable system.

The user interface developed for this system can either be simple or sophisticated. The important point is that even if a simple interface is developed, it can later be expanded without any changes in the file structure or changes to the existing data. The design supports a smooth path to more complex systems.

It has been my experience in designing complex systems for businesses involved in manufacturing, brokerage, banking, and marketing that this structure provides all the core accounting functionality needed in any situation. Accordingly, I will refer to this as the "core system."

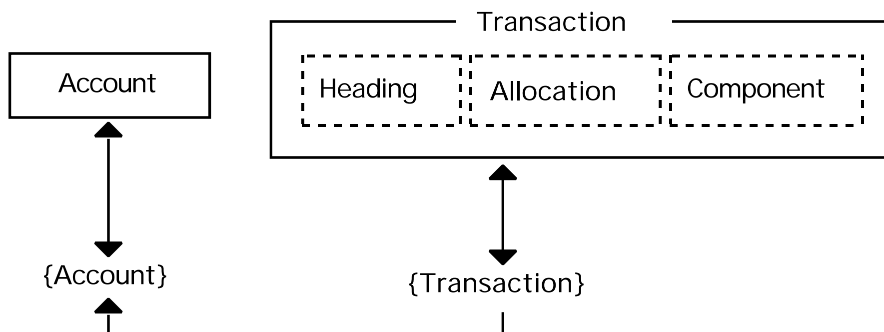


Figure 6. The data store and code function diagram for a complete accounting system.

As we'll see in the next two sections, this core structure can be expanded to support additional functions by adding data stores and processes. The important point is that this core design does not need to be changed.

III-1. Adding noninventory sales data and function

Noninventory sales systems are those used by service-oriented businesses. This can be realized through a simple extension to the previous system, one that only requires the addition of client and sales information. I will simplify the example by eliminating line item details that would normally be part of the description of the sale. The changes required for adding line items are simple.

© 1998 Braided Matrix, Inc. All Rights Reserved

The existing system already supports the handling of accounts receivable (AR) through its ability to allocate payments to debts. If special AR functions are needed, they can be handled in the context of the previous design without adding any additional structures.

Begin by considering a set of rules for handing client and sales information:

Client Rules
- when a client is added, create a client account
- prevent the deletion of accounts linked to clients
- prevent the deletion of clients with past sales
- if clients are deleted (when allowed) also delete their accounts

Sales Rules
- specify a client
- the specified client must have an account
- when a sale is entered, create a sales transaction

Data stores
A client data store could be complex, requiring various client-related files. Its impact on accounting, however, will be through one or more accounts whose data, rules, and functions are exactly those already built into the system. Therefore, instead of creating a new data store for client account information, we can create an account in the existing account data store and link it to the client record.

The point is that no matter how complex the business, the client's account is still just a record in the accounts file. And this account is controlled by the core accounting system that we've already written. For simplicity I'll assume that there is only one account for each client. This would usually be the client's receivable account. I'll also assume that the client data store comprises a single client file, although others could be added.

There is a one-to-one relation between the receivable account created for a client and the client record itself. The simplest way to link the client to the account is to add a foreign key field to the client that refers to the ID of the client's account.

Notice that it would be a bad idea to add a field to the account file in order to link the account to the client. Doing this would introduce a specific data field into an otherwise general data store and violate the modularity of the core system.

© 1998 Braided Matrix, Inc. All Rights Reserved

The data store for recording sales information, like the client data store, can also be arbitrarily complex. Unlike an accounting transaction whose actions are circumscribed, sales information can, and usually does, exhibit as much variety as the circumstance of the sale. This is the reason we add a specific sales data store.

The complexity of the sales information usually concerns the commodities or services provided, the manner in which they're provided, and the agreement between the provider and the client for payment and delivery. This is information contained in the sales data store.

In contrast, the accounting functions of the sale will follow standard patterns. While it is possible that a business might apply special accounting rules to the handling of its sales transactions, this is unlikely. This is because one of the primary objectives of accounting is regularity and predictability. Accounting systems, generally accepted accounting principles, government and regulatory agencies, stockholders, bookkeepers and accountants all require that a business practice standard accounting. This means that all accounting information will share a common structure and be processed similarly. This is good news for the design of reusable software.

The existence of accounting standards implies that the portion of the sales information relating to accounting will be subject to the same rules that we have already designed into our core application. Rather than incorporate the existing rules in the new sales data store, a better option is to store sales specific information in the sales data store and create complementary transaction information in the preexisting transaction data store.

Just as the client record stood in a one-to-one relation with the client's receivable account, so too the sales specific information stands in one-to-one relation to the sales transaction information. In particular, the sales data store records information that relates to the client, while the sales transaction records information that relates to the client's account. The client and sales files are shown below.

Client

- record client name, address, etc.
- create client receivable account

© 1998 Braided Matrix, Inc. All Rights Reserved

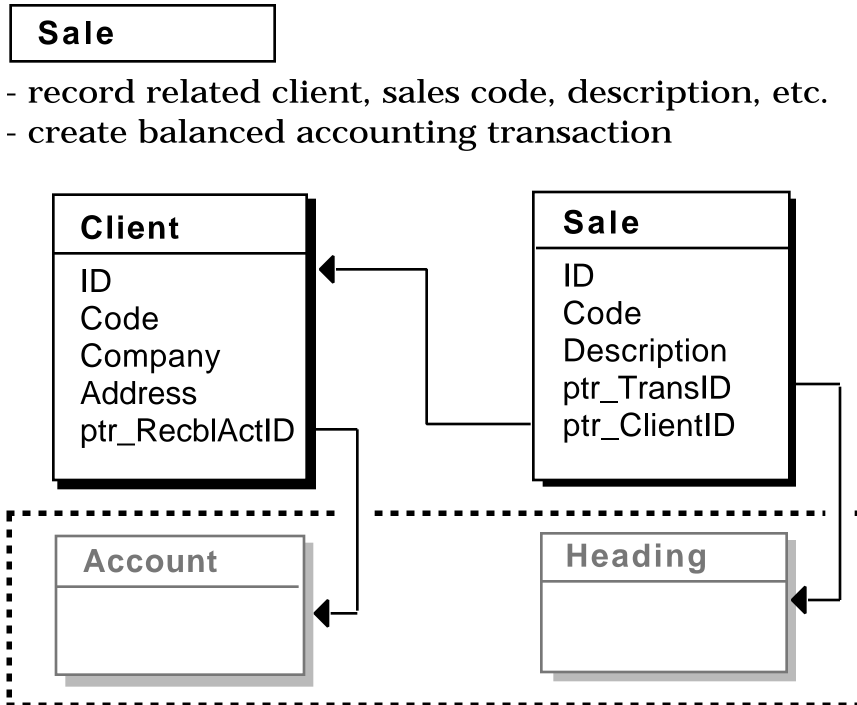


Figure 7. The additional client and sales files added to the existing system to support noninventory sales.

Procedures

The procedures necessary to support this extension include those for adding, modifying, and deleting clients and sales. Added to these basic operations are operations to create client accounts and sales transactions. These operations affect the core account and transaction data stores.

These account and transaction procedures do not affect the client and sales data stores; rather they modify data in the core accounting area. To preserve modularity and encapsulation, as discussed in the previous articles in this series, these operations are best accomplished by calling methods that are part of the code library designed to handle accounts and transactions.

In particular, to create a receivable account, to call an account creation procedure, and to pass it identifying account information. In addition, monitor the result of the procedure to determine whether it succeeds in creating the requested account. After the account is created, assign its account ID to the "ptr_RecblActID" field in the client file.

A similar set of procedures is needed to support the creation of sales transactions. The sales entry procedure calls a transaction creation

© 1998 Braided Matrix, Inc. All Rights Reserved

procedure to which it passes transaction information. This information includes the details of the accounts and the amounts involved. After the transaction is created, the transaction ID is assigned to the "ptr_TransID" field in the sales file.

{Client}
- add
- modify
- delete
- create receivable account → {Account} procedure
- delete receivable account → {Account} procedure

{Sale}
- add
- modify
- delete
- link to client
- create transaction → {Transaction} procedure
- modify transaction → {Transaction} procedure
- delete transaction → {Transaction} procedure

The client and sales procedures do not directly modify data in other data stores. They call procedures in the suites of procedures designed to handle data in the other data stores.

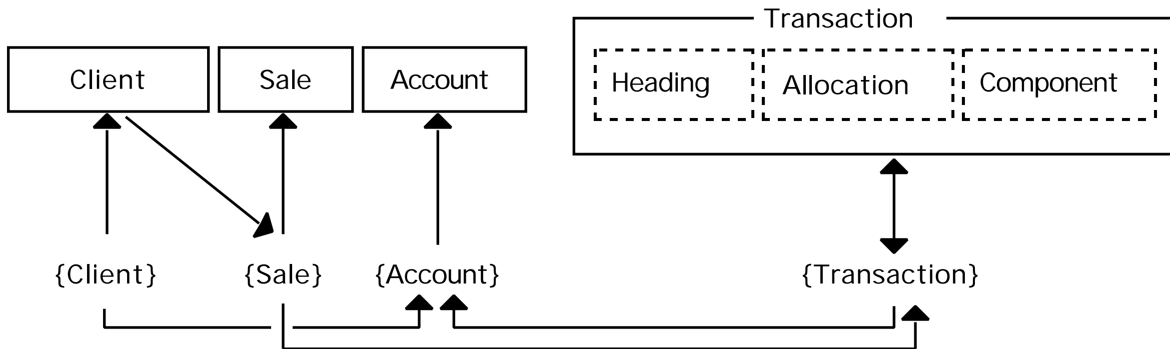


Figure 8. Noninventory sales extensions added to the core accounting system.

III-2. Adding vendors, inventory, contracts and payments

This last example is provided to illustrate the great flexibility of this core accounting structure. I omit the details here and simply list the data stores and procedures.

Data Stores

© 1998 Braided Matrix, Inc. All Rights Reserved

Vendor

- vendor specific information
- vendor payable account

Inventory

- inventory specific information
- inventory asset account
- inventory cost of goods sold account

Contract

- purchase agreement established between you and the vendor
- specifies items purchased and terms of sale

Receipt

- records goods received as per contract
- relates to a payment obligations

Payment

- due on delivery as per contract

Procedures

{Vendor}
- create → {Account} create procedure
- modify
- delete → {Account} delete procedure

{Inventory}
- create → {Account} create procedure
- modify
- delete → {Account} delete procedure

{Contract}
- create
- modify
- delete
- receive delivery → {Inventory} & {Transaction} procedures
- payment → {Transaction} procedure

© 1998 Braided Matrix, Inc. All Rights Reserved

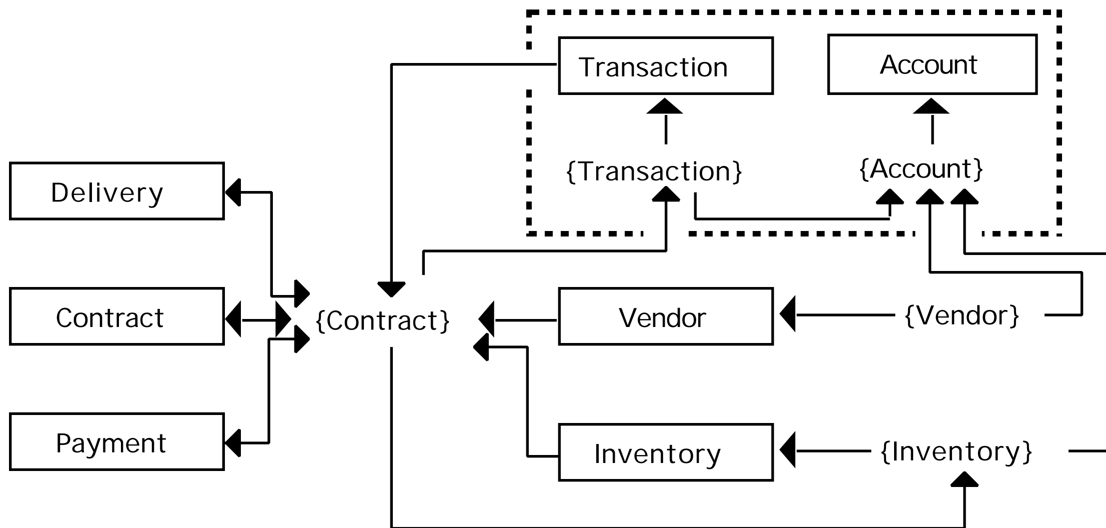


Figure 9. The core accounting system extended to handle a complex business model.

V- Summary: Modularity of code and data

This series has focused on the benefits of modularizing both code and data with the goal of developing scalable, extensible, and reusable software. I've emphasized this by using a diagrammatic technique in which one identifies data stores, code modules, and the interaction between them all.

These techniques are general. I have applied the techniques to the development of an accounting system, but the same techniques can and should be applied to all software design projects. While accounting is especially well-suited to modular design techniques because of the traditionally segregated role it plays in business, a successful modular solution can also be found in other situations.

This core accounting system is both modular and reusable. Any information processing system can be built from this system by adding code and data structures as I've done in these examples. Since accounting is the core of any enterprise-wide business system, you will find most custom systems naturally evolve to include accounting.

© 1998 Braided Matrix, Inc. All Rights Reserved

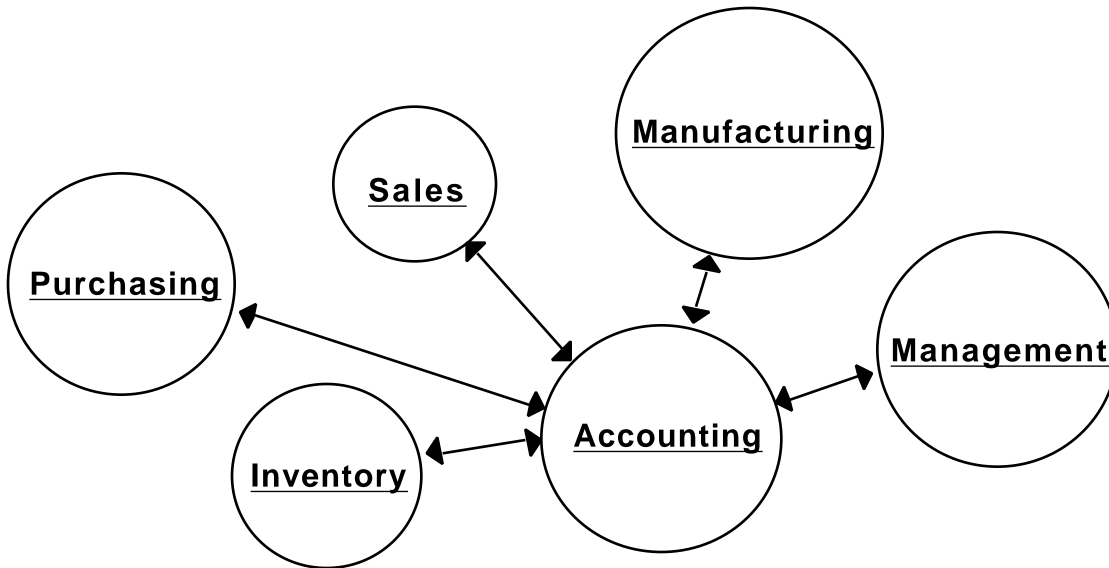


Figure 10. A modular accounting system can be extended without having to be modified.

Developing a modular and reusable accounting system can be of great benefit to both the users and the developers of such systems. Developing skills in customizing a core accounting framework can provide high-end opportunities to developers looking to develop high paying and mission critical skills.

I have developed a complete accounting framework along these lines called 4th Quarter® Accounting Solution. 4Q has proved itself in the design of a dozen complex business systems. Source code, source code training, customization, and a full suite of development tools are available. Please contact me at Braided Matrix, Inc. for information about the product.