Multi-Currency Accounting System

The project is a work in progress (as is this informal document which describes it), but most of its functions are now in use. The development time spent on it could never be justified in “business” terms, but it does provide an opportunity to use programming “best practice”, and also as a code repository. It's been essential to my professional development. I not be publishing the source code, but the more useful snippets often find their way into software I write for clients.

Functions

The project is a book-keeping and budgeting system handling multiple accounting entities and multiple currencies. It handles invoicing, VAT calculations and book-keeping to trial balance with personal and asset accounts and it allows journalled adjustments to be made. It stores data from bank statements, receipts cheques and payment slips, plus historical data on foreign exchange rates and inflation. Some document management and other administration functions are also provided.

Technology

- C# (Framework 3.5, WPF, Linq) using Visual Studio 2008
- VSTO using Excel 2007
- SQL Server Express 2008

Size of application

Where there will be direct access to the database by maintenance programmers, there is a high risk that changes made will put the data into an invalid state unless business rules are enforced in the database. Since that will definitely be the case with this application, most of the accounting rules are enforced in the database. The database structure is therefore relatively complex as compared with databases where all business rules reside in the object layer. The statistics are as follows:

Object Name	Count
USER_TABLE	289
VIEW	68
SQL_STORED_PROCEDURE	662
CLR_STORED_PROCEDURE	7
SQL_INLINE_TABLE_VALUED_FUNCTION	59
SQL_TABLE_VALUED_FUNCTION	25
SQL_SCALAR_FUNCTION	115
CLR_SCALAR_FUNCTION	1
CLR_TABLE_VALUED_FUNCTION	1
SQL_TRIGGER	7

The large number of stored procedures is because database permissions don't allow CRUD operations to be performed directly on tables by the client, they are done through stored procedures. The development time to create these procedures is greatly reduced by developer utilities, as discussed below.

Multiple books

Small businesses can benefit from one system handling their personal book-keeping and business

books. Many payments relating to the business will be made from personal bank accounts. The system can then ensure that these payments do not “fall between the cracks”. They will have to be allocated to one book or the other, and any which are not allocated can be found easily.

Multiple Currencies

Where there are overseas obligations it may be necessary to record them in the foreign currency, since the rate applying at the future date of payment is not known. Personal accounts can be set up in a foreign currency and an ad hoc estimated forex rate applied to reports, until the actual payment is made. A full history of foreign exchange transactions is available.

Client Application

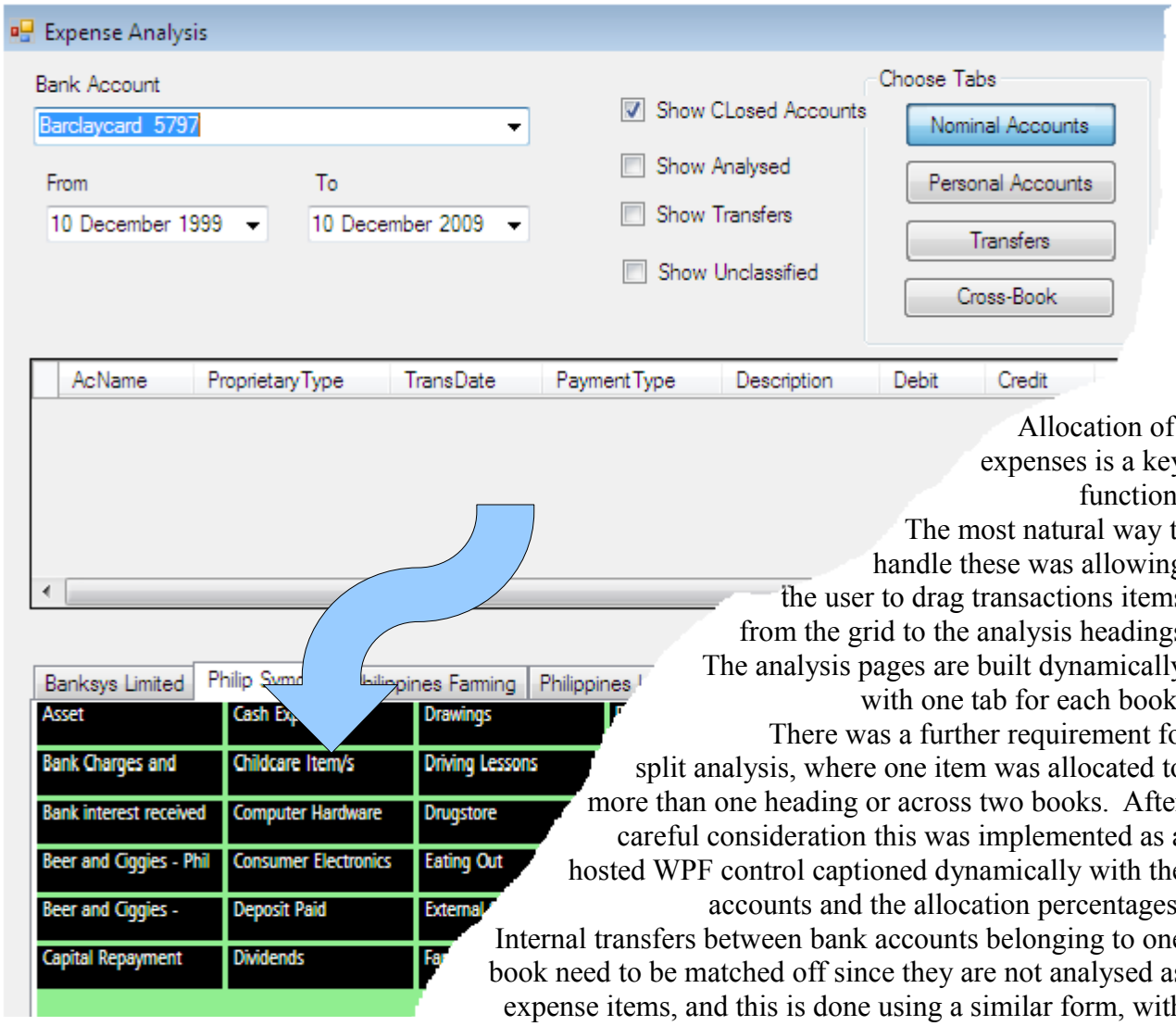
Database connections are opened and closed as required. Connections are made through a Singleton login form which exposes a connection function. The application detects which kind of connection will be required, and the form is only shown to the user if Sql Server security must be used (as in a remote call over a peer-to-peer network). Once the user id and password have been entered , they are retained as instance variables, so users only have to enter these details once. The application also connects to an Oracle back end so this form supplies Oracle connections as well.

Reference data is also implemented in the Singleton pattern, so that there don't have to be repeated trips to the server to look up static information. Using this approach in a big corporate environment would need careful consideration but it should be possible to keep some of the benefits.

Income and Expense Analysis

The starting point of the accounts is the cash book. Transactions in the cash book are matched to bank statements, and can be posted to nominal or personal accounts. These accounts belong to financial periods. Accounts have start and end date properties which are contained by the parent financial period and cannot span periods. When a new period starts a new set of accounts must be created. This makes it possible to close a financial period and “lock” its accounts so that no further items can be posted. When a new financial period starts a new set of child accounts is created.

An AccountSet object contains the collection of accounts for a given nominal or personal account head. This makes it possible to use them without getting involved in dates or financial periods. This object exposes the common properties such as AccountName and BookCurrency. A GetAccount() method looks up the relevant account for a given date and currency. Each of the specific account types implements an ILedgerAccount interface containing the common properties used to post items to those accounts.



Allocation of expenses is a key function. The most natural way to handle these was allowing the user to drag transactions items from the grid to the analysis headings. The analysis pages are built dynamically with one tab for each book. There was a further requirement for split analysis, where one item was allocated to more than one heading or across two books. After careful consideration this was implemented as a hosted WPF control captioned dynamically with the accounts and the allocation percentages. Internal transfers between bank accounts belonging to one book need to be matched off since they are not analysed as expense items, and this is done using a similar form, with transactions appearing in the upper grid and possible matches in the lower grid. A transaction is matched by dragging and dropping it on the chosen match.

Database

DataLoad

Bank statements are loaded into the database and like any other type of raw data, can cause havoc to reporting if they are not checked carefully before being allowed in. This is the “barricade” approach described in the latest edition of “Code Complete”.

Relationships

Composite keys have been used rather than artificial keys in all cases where the objects modelled allow this. The reason for this is because during development it's more difficult to understand the business rules behind relationships defined with artificial keys, and also more difficult to enforce those rules, which is why this is often left out. In the event that the application grew to the extent that this was a performance or storage issue, the composite keys could be replaced with artificial keys. Since this would only be done where there was a performance gain, the advantage of composite keys would be retained in most places.

All lists are centralised and allocated to categories, with an inheritance tree defined in the database. This makes the lists easier to manager since virtually all classification used within the system can be

located with the following query: “SELECT * FROM tblCategory”. Sample category types are shown below.

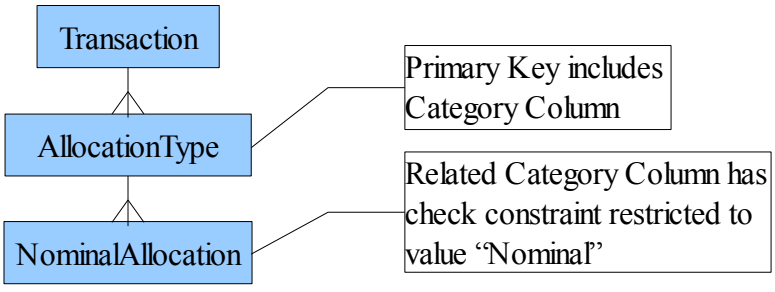
ACCOUNT
BANK ACCOUNT TERM/CONDITION
BANK PAYMENT METHOD
BANK SORT CODE PURPOSE
BANK STATEMENT FREQUENCY
BANK TRANSACTION TYPE
DOCUMENT
DOCUMENT STORAGE LOCATION
GRAPH X AXIS DATA TYPE
GRAPH Y AXIS DATA TYPE
LAND ACQUISITION COST
LAND ACQUISITION DATA INPUT
MARKET DATA INFORMATION
PARTY CATEGORY
PARTY DETAIL
PARTY LOCATION
PARTY LOCATION ELEMENT
PARTY RELATIONSHIP
URL

Analysis

Many of the documents such as bank statements, cheque and payment records are stored in top-level tables (with no foreign-key dependencies on other tables). This means that records can be added as soon as convenient and matched later to transactions or to other documents.

A crucial feature of database accounting systems is that an entry must not be allocated more than once, or allocated to an incompatible accounting period or to a pair of incompatible accounts. This has been made impossible in the database by use of “type” tables which are in a parent relationship to the allocation table. Used with check constraints in the child tables where the allocation is actually done, this prevents invalid allocations.

The structure is as follows:



Maintenance/Development Support

There are a number of schemas defined in the application. These are important as they make it easier to logically segregate the objects. Several schemas which may be of interest to developers are

Schema	Use
Coord	“Controller” procedures that chain other stored procedures together to do a high-level job
Data	Mainly functions and objects intended to test and manipulate data
DataLoad	Objects related to loading of raw data into the database and validating it
DBA	Database administration
DEV	Objects not approved for PROD use
Meta	Information about the database
Tactical	Contains objects built to do a job but not intended to form a permanent part of the system
Test	Scratch objects for experimentation
Util	Utilities for string processing and manipulating dates, etc..
XXX	Location where objects can be moved prior to being dropped.permanently