Data: The Forgotten Business Application

If you asked financial services business users to name the most important business application, no one would name Data. Data is viewed as a foundation for every other business application, not an application itself.

While it is certainly true that business applications use data, data is also important in its own right and has certain properties that make it unique. When properly managed, data supports business relationships that would otherwise be problematic for applications.

Historically, we have looked at data solely as a requirement of business processes. If the data exists, the applications can run. If the data is invalid or missing, errors will occur, and we can use routines to execute defaults or process changes.

Each process uses data in its own unique way and for its own proprietary business reason. So, any one field of data may be used in hundreds of ways, which complicates data organization and process modernization.

The Nine Ways Data Can Be Managed

- Create
- Delete
- Modify
- Copy
- Use as a 'key' to link with other data
- Display or print
- Format
- Transmit
- Associate to other data elements (relational integrity)

To gain a better understanding of the role of data, let's look at the brokerage business. For brokers, the trades transaction stream is the heart of the business. To support a variety of processes, data is arranged within databases in ways that best satisfy trades transaction needs. But what about the needs of other departments like Corporate Actions which uses and manipulates data in ways that have little or no relevance to trades transactions? Are these needs any less relevant to brokerage?

The same arguments can be made for stock loan processing, mutual funds processing, compliance and regulatory processing, and Automated Customer Account Transfer (ACAT)/ delivery processes. By focusing on one application area when designing the data structure, all other areas suffer to some degree from design inefficiency.

Database software has attempted to address these inefficiencies by maintaining multiple entry paths to data, creating redundant data tables, and offloading cycles from the server or mainframe into its own engine(s). Yet these solutions are still focused on the ways an application uses data, rather than addressing the data itself.

Data can only be managed in nine ways (see sidebar) so creating services that handle data is relatively simple. The keys to creating these services are the same as in any application:

- there must be an owner for every data element
- there must be a department or person accountable for the service, and
- there must be strict documentation that describes the exact meaning of each data element (data taxonomy—not a data dictionary).

Transaction vs. Static Data

There should also be distinctions made about the type of data that should be looked at as a business application. In general, transaction data should not be included. This data is usually a grouping of retrieved and derived data elements created to satisfy a single process need. It is transient in nature, usually becoming a source for other transactional or static data. For example, trade stream spins off into settlement transactions and eventually into portfolio accounting (tax-lot) data.

This leaves static data. Static data can be roughly defined as data that has a unique identity and value regardless of how other applications may use it. It usually remains with a constant value for long periods of time, and it always has a distinct taxonomy meaning. Examples of static data for brokerage firms are ticker symbols, instrument types, exchanges, client codes, client name and address information, and broker number. The value within these data elements has a meaning separate and distinct from how any application uses the information.

Static data has other qualities that make the creation of data application services useful. Since static data elements are rarely derived from transactions (tax-lots are an exception), the relationships between static data elements remain constant. This built-in relational integrity factor allows business rules associated with relational integrity to be integrated into the service, rather than managed in each business application. This feature alone is worth its weight in gold because it helps prevent 'dirty data' from either entering a primary database or being inadvertently created by a business process.

Structuring Static Data

One example of how structuring static data as a business application differs from more conventional approaches can be seen in Name & Address files. Every business needs to have this static data subset, and nearly every business process uses this data in some way. In fact, due to factors like technology changes over time, the need for crisis reparation of broken applications, and simple expediency, most companies have multiple Name & Address databases and a complicated and often convoluted process to synchronize them. The average brokerage firm has over seven Name & Address databases in use.

Since each database incurs ongoing costs to update, manage and maintain it, imagine the savings if a firm could combine them into a single source without any loss of functionality.

The key to a universal design for Name & Address data is to create two related tables. The first contains the data elements themselves, organized into tables of associated data elements, as is traditionally done. Variations would be appropriate based on ownership of specific elements, since ownership groupings would simplify update logic within the services.

The second set of tables would contain a list of valid relationships between the ways in which a business application uses that data, and the data itself. This table may contain entries such as client, attorney, mailing address, responsible party, CRM activity or owner. The business applications own the relationship tables; the static data groups own the Name & Address data elements. The called service retrieves or updates data based on the business logic associated to the relationship definition.

Static Data at Work: An Example

Corporate Actions needs to send the owner of a mutual fund a semi-annual report. However, for this fund, owners must be sent registered mail letters if they are corporate owners. Additionally, all corporate officers must also be sent the report. By creating a special owner relationship, the data service can store this logic away from the traditional processes for sending reports and funnel it to appropriate processing. The actual mailing processes never need to have exception logic coded or changed if or when the rules change.

If we used the above example to create Name & Address data tables, only one instance of every data element would ever be needed. Applications would call the data service, providing the relationship to be used to acquire the data. If changes are ever needed, only the service needs to be modified, and usually that can be a real-time change after the completion of proper testing.

How do we describe data?

Data is described in many ways—the elements that make it up, the source it comes from, the way it is used. For our purposes, we look at data in two ways:

Data type: transaction and static data describe the way the data is created and updated.

Data communities: source data, enterprise data and private data describe the ways data are grouped together based on how they are used. Source data comes from external sources or feeds; enterprise data is used across multiple business groups or business silos; and private data only has meaning in a single application.

Where this technique shows immediate value is in modernization efforts, especially when the legacy applications are COBOL / CICS focused. Generally, modernization is problematic at best because the business applications are all intertwined. This makes any approach other than the "big bang" nearly impossible. The applications themselves cannot easily be tested individually.

The Value of Data Applications

In the example above, by creating data applications and their associated data services, many of the integrations of business logic can be made to disappear. Replacing the data retrieval and validation logic in subsets of programs will reduce their size and complexity, leaving true process logic, which by definition stands alone and apart from unrelated process streams. Testing will primarily involve maintaining parallel data structures, replacing the data logic in existing programs with the new services, and seeing if the results from each version of the programs are the same. If they are, the new service works and the modernized program can be returned to production.

Clearly there is much more to modernization efforts than data structures. This approach merely illustrates how data can be used efficiently in a modernization effort. All other factors such as creation of a solid infrastructure, establishment of governance, creation and use of testing and staging environments, and implementation of appropriate security are all equally important.

The other critical factor is finding a program manager who understands these concepts and is skilled at managing this kind of data effort. The main concern will be to continually explain to business why this approach will be effective and not just a costly experiment.

About the Author

David Hirsh is part of the Enterprise Information Management Consulting Practice within Capgemini Financial Services.

For more information on our enterprise information management offerings, please contact us at **financialservices@capgemini.com.**

Learn more about how to separate data into a business application that can be used across your enterprise during legacy modernization efforts through Controlled Migration.

About Capgemini and the Collaborative Business Experience

Capgemini, one of the world's foremost providers of Consulting, Technology and Outsourcing services, has a unique way of working with its clients, called the Collaborative Business Experience.

Backed by over three decades of industry and service experience, the Collaborative Business Experience™ is designed to help our clients achieve better, faster, more sustainable results through seamless access to our network of world-leading technology partners and collaboration-focused methods and tools. Capgemini utilizes a global delivery model called Rightshore® which aims to offer the right resources in the right location at competitive cost, helping businesses thrive through the power of collaboration.

Capgemini reported 2009 global revenues of EUR 8.4 billion and employs over 90,000 people worldwide.

More information about our services, offices and research is available at **www.capgemini.com.**

FS20101102100TCS