Embracing Change:
Financial Informatics and Risk Analytics

Mark D. Flood
Senior Financial Economist
Federal Housing Finance Board
floodm@fhfb.gov

April 17, 2006

EXTENDED ABSTRACT

Embracing Change: Financial Informatics and Risk Analytics

Overview

We present an enterprise design pattern for managing metadata in support of financial analytics packages. The complexity of financial modeling typically requires deployment of multiple financial analytics packages, drawing data from multiple source systems. Business domain experts are typically needed to understand the data requirements of these packages. Financial product innovation and research advances imply that data requirements are chronically unstable. These forces of complexity and instability motivate a software architecture that exposes financial metadata declaratively for editing by financial analysts.

The key contributions of the paper are twofold. First, we present a simple model that captures the most important software development costs involved in maintaining risk-management databases. This model allows us to identify and measure the scalability of alternative data-integration architectures. Second, we present a high-level data-integration architecture that is flexible, scalable, practical, and portable. This solution uses an ontology editor (e.g., Protégé) to expose the financial metadata for user editing in a controlled context. As a result, domain experts can make on-the-fly metadata modifications without provoking a costly design-develop-test-deploy iteration. The solution would not be practical without some facility for ontology editing.

The problem

Three basic forces – financial innovation, model risk, and strategic policy evolution – conspire to create a very unstable data integration environment for risk-managment analytics. The nature of the data coming into the models may be changing, due to financial innovation. The set of models in use may be changing, due to modeling innovations or shifting conditions and priorities. The nature of outputs requested from the models may be changing, due to changes in strategic goals. Most risk-management applications– including trading decisions, capital allocations, and hedging limits – require highly accurate data, implying a premium on the quality of the metadata. Wall Street spends billions of dollars every year addressing the operational (data integration) issues implied by these complications.

Specification and mapping costs

There is a potentially large and unstable list of source systems from which data will ultimately be drawn, and target analytical packages which the data will feed. There are two costly operations involved with managing the metadata for such a system: (a) *specification* of the data schemas for each system; and (b) *mapping* between schemas in the source systems and input schemas in the target analytics packages. To minimize the costs of mapping and create a data-integeration architecture that scales linearly in the number of software packages, we recommend the introduction of a central normalizing schema, which we call the "numeraire schema". The scalability properties of alternative architectures are established in a very simple model.

A design pattern

We specify a data-integration architecture that meets four high-level requirements:

R1. The solution must be ***flexible*** – capable of adapting to financial and research innovation in a short time frame. To achieve this, the solution exposes the metadata for editing by business analysts, as the needed responsiveness (for re-specification and re-mapping) is too quick to allow imposition of full-blown design-develop-test-deploy lifecycles.

R2. The solution must be ***scalable***. Modeling and mapping costs must not explode as the number of analytics packages, source data systems, and financial contract types grows.

R3. The solution must be ***practical***. That is, it must be implementable with production-quality tools and technologies available today.

R4. The solution should be ***portable***, to facilitate adoption. In other words, the design must be self-contained, and therefore deployable in a range of enterprise architectures (e.g., XML messaging, shared database, etc.), and software platforms (J2EE, .Net, etc.).

The design divides into four separate subsystems: data integration, position data, market data, and analytics. This division enhances portability, since the subsystems are very loosely coupled. The data integration subsystem sits at the center of the design. It includes the database maintaining the full official state of the system. Importantly, it also includes an ontology editor (e.g., Protégé). This addresses the flexibility requirement by allowing financial analysts to modify (perhaps frequently) the specifications of the various data sources and targets. The system has a number of disparate pieces – user documentation, data schemas, mappings, etc. – that must remain consistent with one another, even in the face of such changes. The design employs a formal ontology to record these facts as declarative "meta-metadata," since an ontology can include the rules to enforce consistency among the disparate pieces at all times. The ontology editor must enable users to modify both the general *structure* of a data-specification ontology as well as the individual specification *instances* corresponding to each data source and target.

The resulting four-layer architecture (consiting of: data, data schema, ontology instance, and ontological structure) is closely analogous to the Object Management Group's four-layer meta-object facility, with the innovation that an ontology editor (Protégé) is used to manage the top two layers of the metadata abstraction hierarchy. In a four-layer design, traditional metadata, typically consisting of SQL or XML schemas, are governed by a higher-level "meta-metadata" that describes the traditional metadata. In turn, these are governed by the ontological structure (the "meta-meta-metadata"), an abstract language for defining metadata. Each layer is an abstraction of the layers below it, and each layer must obey the constraints defined by the layer above it. The constraint that ontology instances must share a common ontological structure greatly facilitates programmatic access to the ontology for editing and generating documentation, data schemas, test data, etc.