

Context Mediation on Wall Street

Allen Moulton
amoulton @ mit.edu

Stuart Madnick
smadnick @ mit.edu

Michael Siegel
msiegel @ mit.edu

MIT Sloan School of Management

Abstract

This paper reports on efforts to construct a practical implementation of a context mediator for the fixed income securities industry. We describe industry circumstances and the DCS mediator product that was built in the early 1990s. The mediator was designed as an interpretive engine controlled by a static declarative knowledge structure and client preference data. In addition to heterogeneous, autonomous data sources, the mediator integrated autonomously developed local and remote procedural components. Client access to both data and computational resources were provided through an active conceptual model. Structural and semantic context conversions were used to integrate disparate components and to support varying client needs. Lessons learned from the implementation and usage of this mediator provide insight into the requirements for a successful context mediator.

1. Introduction.

The fixed income securities industry is an environment with potential for major benefits from rapid and effective interchange of semantically heterogeneous information among autonomous entities. Many firms in the industry develop specialized, and often proprietary, knowledge. That knowledge is more valuable when used in combination with knowledge from other specialists. Although standardization can facilitate knowledge interchange, the rapid evolution of financial products, transactions, and markets characteristic of the fixed income securities industry suggests that a dynamic context mediator approach may have significant value.

The challenge of the Data and Calculation Services (DCS) mediator product described here was to bring together six highly competitive firms to cooperate in providing decision support information to their customers while retaining autonomous control of proprietary data and analytic resources. Wiederhold [1][2] described mediators as an extra software layer to provide client applications with access to the integrated knowledge

available from heterogeneous, autonomous data servers. The DCS mediator extended the definition to include computational resources as well as data sources.

Context refers to the implicit assumptions that add meaning to symbols. Autonomous organizations often develop different contexts, resulting in semantically heterogeneous representations of the same knowledge[3]. In making investment decisions, it is vital the information conveyed among parties be understood by both. Context mediation can fill the gap where industry association and regulatory standards leave off. In COIN [4][5][6], sources and receivers independently describe their context to the mediator, which uses metadata to detect conflicts in data representation and interject conversion operations in the query plan. The DCS mediator supported context definition and automatic conversions for parameters and values of computational functions as well as source and receiver data. In addition, the DCS mediator experimented with context conversions among different representations of security valuation related through abstract models, often implemented in proprietary analytic software.

The DCS mediator employed a domain model of information applicable to fixed income decision-making. SIMS [7], Information Manifold [8], Infomaster [9], and Garlic [10] also employ a domain-specific ontology. In DCS and these other projects, data sources are integrated through a shared model of knowledge specific to the problem domain. Users access information through the mediator by working directly with the domain model. DCS differs from the approach of TSIMMIS [11], which uses an ontology-free approach where data is integrated and delivered in its own terms, from Carnot [12][13] which draws on an ontology of general common-sense knowledge, and from Benaroch's [14] use of macro-level ontological knowledge to integrate disparate knowledge-based systems for financial risk management.

2. The fixed income securities industry.

Information is the critical resource in the fixed income securities industry – information about securities and their

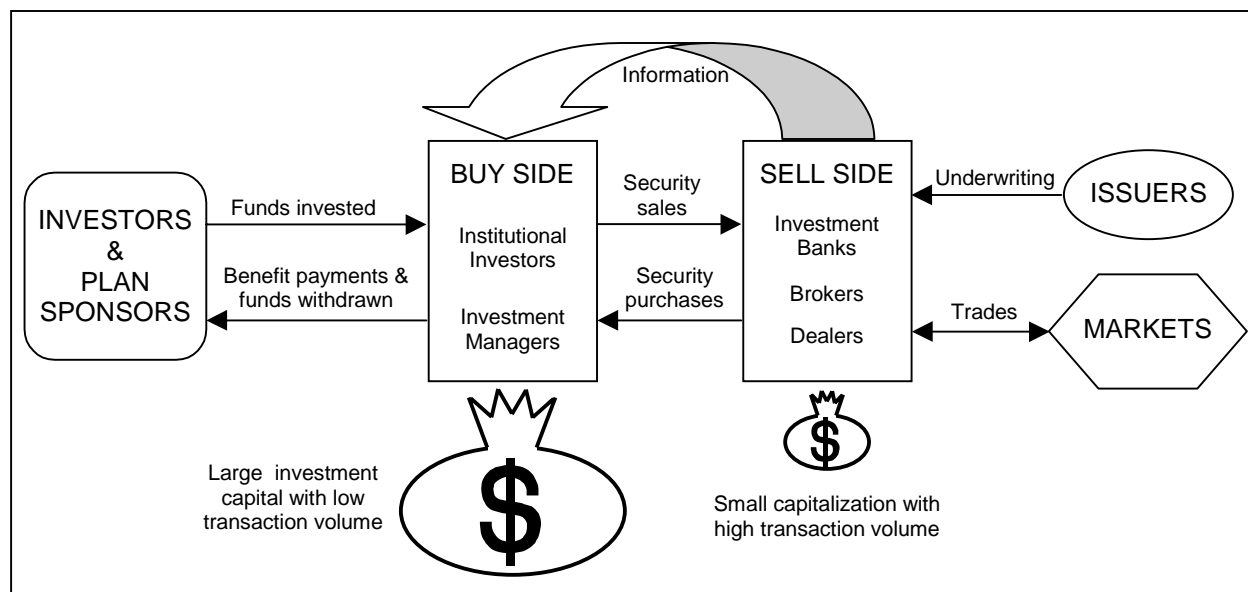


Figure 1. The fixed income securities industry

issuers, information about markets, information about economic conditions and events, and information about methodologies and models (see Figure 1). Billions of dollars of debt instruments trade every week. Firms on the “buy side” (institutional investors and investment managers) manage capital on behalf of investors and benefit plan sponsors. Firms on the “sell side” (investment banks, brokers, and dealers, often known as “Wall Street”) bring new securities to market and interact to create capital markets. While the popular connotation of sell-side firms is of great wealth, the capitalization of these firms is actually relatively low. The big money is on the buy-side. The sell-side makes money from a small commission or price spread on large volumes of transactions. The sell-side offers buy-side managers access to its skill, expertise, and knowledge in anticipation of purchase and sale orders.

Bonds and other fixed income securities are obligations to pay sums of money at points in time over the life of the security. Unlike equity securities, an investor has no stake in the financial entity that issued the security. To an investor, a fixed income security represents a stream of future cash flows. A purchase decision trades present money for future payments. There may be optional events that change the cash flow stream and there may be risk of default. In essence, however, all fixed income securities are interchangeable commodities from the point of view of an investor. The cash flows from one or more obligations may even be repackaged by selling off rights to payments or by combining rights to payments into new composite securities. This repackaging, or “financial engineering” can produce securities known as “derivatives.” Faced with a vast array

of combinations of cash flows, risks, and optional events, every industry participant needs timely information and effective methods for determining investment value from raw data.

In 1990, six major Wall Street investment banking firms formed a partnership establishing an “Electronic Joint Venture” aimed at using advanced technology to improve communication with their buy-side clients. During the 1970s and 1980s, fixed income investment management had become substantially more complex. Interest rates had fluctuated over a wide range. Huge quantities of new debt instruments had been issued. New securities products had been regularly introduced. Analytic methodologies had advanced to cope with the new complexities. Markets had become more globally integrated and moved at a faster pace.

To cope with the new complexity, sell-side firms hired bright young “rocket scientists” with mathematical and engineering backgrounds into “fixed income research” departments. They developed databases with detailed descriptive information about securities as well as historical prices, interest rates, and economic statistics. Proprietary real-time broadcast networks brought current market information and news to trader and investment analyst desktops. Application software was built to display securities data, perform securities and portfolio analytics, and apply the models developed by financial engineers. Information technology had also been advancing rapidly. Business mainframes stored databases, supercomputers performed the most complex calculations on an overnight basis, while a combination of dumb terminals, PCs, and workstations were piled high on trading and sales desks.

In the midst of this jumble of technology, buy-side firms began to demand direct access to data, analytics, and models. Previously, the sell-side would offer ideas and supporting analysis on the phone and by FAX. The buy-side wanted more control over the analysis used to make investment decisions. Investors recognized that the sell-side had the knowledge that was needed. Investors wanted to be able to integrate proprietary sell-side data, analytics, and models with their own internal data, analytics, and models so that they could make better decisions. The sell-side firms were willing to provide information resources to the buy-side, but wanted to protect proprietary information from being passed on to competitors. The sell-side also hoped to receive information from the buy-side about positions held in their portfolios in order to be able to make better trading recommendations.

Investment management involves buying, selling, and swapping securities in order to improve the performance of an investment portfolio while meeting the sponsor's objectives. In fixed income, quantities of securities are generally measured in face value. Prices, interest, yields and many other values are usually expressed as a percentage of face value. The price (or valuation) of a security depends on the general interest rate market and the security's predicted cash flow compared to other investments on the market. A variety of methodologies, such as internal rate of return (IRR), horizon rate of return (ROR), and option-adjusted spread (OAS), are regularly used to assist in investment decision-making. An

investment manager needs to be able to apply the same methodology to all securities held in a portfolio and all potential investments. A requirement for an effective decision support tool is the ability to apply a common methodology across diverse securities. An effective tool must also be extensible to new kinds of securities invented by financial engineers and give investment decision-makers access to and control over assumptions and models used in valuation methodologies.

The mission of the Venture was to provide the infrastructure to facilitate the interchange of information between buy-side and sell-side firms. The technology plan involved six major areas: physical connectivity via a proprietary network, a desktop workstation platform, a suite of standard applications, standard databases, standard analytic calculations, and the data and calculation services (DCS) mediator. By offering "generic" or industry standard applications, databases, and analytic calculation software the venture would be able to replace duplicative efforts on the part of clients and its partner firms. All Venture-supplied applications, databases, and calculations were required to be designed on an open architecture that would allow partner firms and clients to develop their own proprietary extensions and substitute parts. For example, a partner firm might provide a proprietary application that would use and extend the Venture's standard databases and calculation software. A partner firm might alternatively supply a proprietary algorithm for use within a standard application.

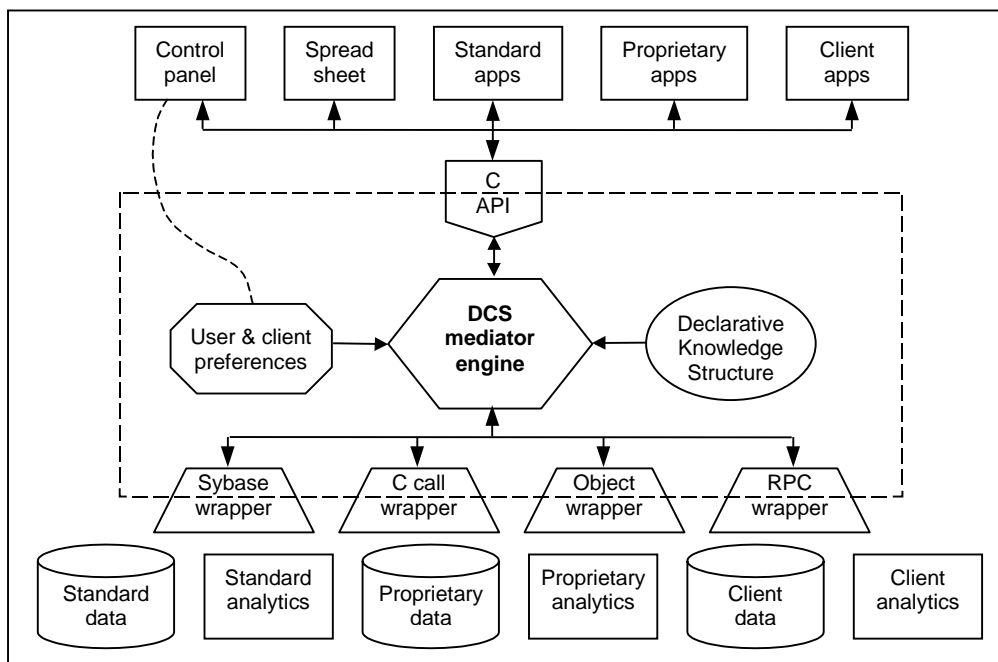


Figure 2. DCS mediator architecture

3. Architecture of the DCS mediator.

The DCS mediator (see Figure 2) was designed as a demand-driven general interpretive engine controlled by a static declarative knowledge. The declarative knowledge structure included a domain model for fixed income securities and analytics, database retrieval and calculation function wrapper specifications, and a dependency graph. The declarative knowledge was prepared in a definition language script compiled into a binary structure for runtime interpretation. Application software accessed the mediator through a C-language application-programming interface (API). Ad-hoc user access was provided through a spreadsheet interface. Generic wrapper interfaces were developed for the Sybase relational DBMS, C function libraries, an object technology contributed by one of the partner firms, and remote procedure calls (RPC).

Client preferences and assumptions were contained in global data objects that could be viewed and manipulated through the Control Panel application. By changing values in these global objects, the user could set the financial assumptions, data sources, and analytic models used by the mediator in responding to other application queries. In essence, the Control Panel allowed the user (and the client firm's management) to define the context, the implicit assumptions, of information to be received.

3.1. A fixed income securities domain model

The core of the domain model was a DCS conceptual model, which defined an abstract view of the entities, attributes and relationships available through the mediator and served as a shared ontology for integrating heterogeneous, autonomous data and calculation resources. Publication of the conceptual model served as the principal documentation of the data and analytic functionality available from the mediator. Each entity class defined a collection of attributes accessible through "get" and "put" operations provided in the API. No object

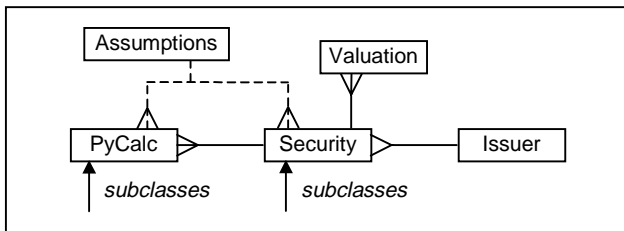


Figure 3. Abstract classes and relationships

methods or parameterized procedures were allowed. Application and spreadsheet developers worked directly with conceptual model objects to access DCS resources. The user asked for information from the DCS mediator.

The mediator decided whether that information would be calculated or obtained from a data source.

Figure 3 shows some of the abstract classes for in the conceptual model for fixed income securities and analytic methodologies. In the center is the *Security* class, representing the generic fixed income security with a many-to-one relationship to the *Issuer* class with information about the issuing financial entity. There is also a one-to-many relationship to *Valuation*, which contains information about end of day prices from various sources. The *PyCalc* (price-yield calculator) class represents information about an actual or hypothetical trade in a security and analytic measures describing the trade. The *Assumptions* class holds user and client financial assumptions, and control information about data sources, and analytic models for use by the mediator in responding to application queries.

The DCS domain model supported single inheritance and dynamic polymorphic subclassing. Figure 4 shows a sample of the polymorphic subclasses of the *Security* and *PyCalc* abstract classes. In each case, the subclass is determined by the value of a meta-attribute in the parent class. *Security* subclasses correspond to different kinds of descriptive data relevant to each type of security: *SecBond* for bonds, *SecPool* for mortgage pools, *SecMGen* for a generic mortgage model, and *SecTranche* for a tranche of

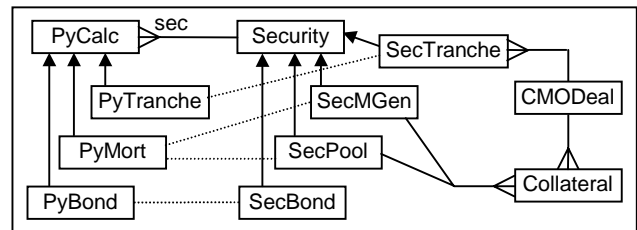


Figure 4. Polymorphic class relationships

a collateralized mortgage obligation (CMO). For the *SecTranche* subclass, the security is related to a *CMODDeal* entity and in turn to a set of *Collateral*, each of which may be a position in a pool or a generic mortgage.

Subclasses of *PyCalc* correspond to alternative procedures for performing the functionality of the abstract class depending on the type of security involved. In some cases, a subclass might also provide additional functionality specific to that type of security. Figure 4 shows subclasses *PyBond* for bonds, *PyMort* for mortgage generics and pools, and *PyTranche* for CMO tranches. The subclass of *PyCalc* is derived from the type of *Security* entity referenced by the "sec" attribute in the calculator.

Enumerated metavalues provided a standard terminology for mediating among different representations of security and calculation features. For

example, the description of coupon interest payment frequency and interest accrual convention for a security used metavalues. Similarly, the yield calculation conventions in the *Security* and *PyCalc* classes used metavalues. The context definition for a data source or computational function library included relations to map context-specific representations to the DCS domain model standard. The mediator used these mapping relations to convert representation across contexts.

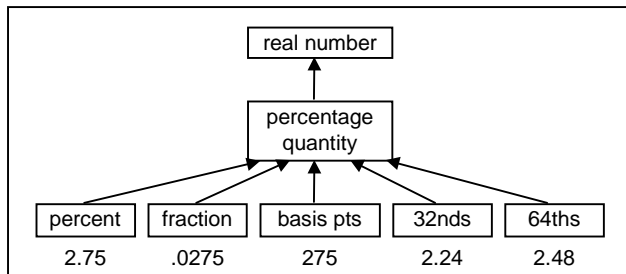


Figure 5. Representations of percentages

DCS context conversion also helped alleviate one of the pesky problems of fixed income software development by providing automatic conversion among different representations of percentage quantities. Securities are priced at a percent of face value, interest rates and yields are percentages, as are spreads between two prices or yields, the interest accrued on a unit investment as of some point in time, and many other quantities. Figure 5 shows five alternative representations of the same 2.75%. The "fraction" form is favored by spreadsheets and many programmers. Basis points (hundredths of a percent) are handy for talking about small spreads. The "32nds" and "64ths" forms are used for bond markets that trade in those increments. The DCS domain model provided types for alternative representations of percentages and automatic conversion among them.

Figure 6 shows a sample of the "placemat" depiction of the generic *Security* class, showing some attributes and

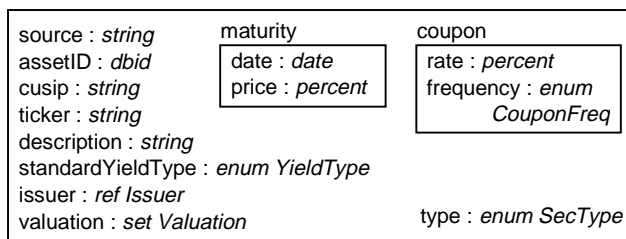


Figure 6. Generic Security class placemat

data types. The *type* attribute in the lower right provides an explicit representation of the type of the security. Type may be implied (because a source is limited to a single type) or derived from stored information. The *maturity*

and *coupon* boxes inside the diagram are internal structures describing the maturity and the coupon interest payments of the security. Attributes declared as *percent* could be filled with any form of percentage quantity. Many-to-one relationships are shown as role attributes, such as *issuer*, referring to a single object. One-to-many relationships appear as role attributes, such as *valuation*, referring to a set of objects.

An object of the *Security* class could be used as a hypothetical security, with attribute values filled in by the application program. More often, a *Security* object would be associated with stored data using key attributes such as *cusip*, *ticker*, or *assetID*. CUSIP is an industry standard identifier used in the United States and Canada [15]. An *assetID* was an artificially constructed globally unique key for all security data objects accessible through DCS. Venture-supplied databases were designed with *assetID*. Autonomous databases are made accessible by extracting the primary keys of each security record into a lookup table and assigning a unique *assetID* to each. The *assetID* also contained codes indicating the source for the data and the security type.

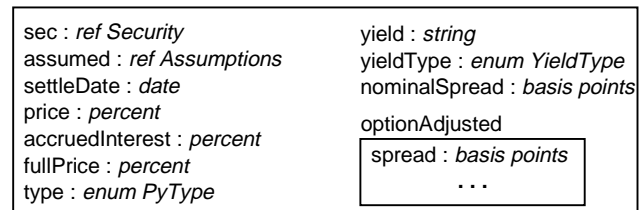


Figure 7. Generic PyCalc class placemat

The *PyCalc* class (Figure 7) provided both IRR and OAS analytics for a real or hypothetical trade in a security. The *sec* attribute provides a reference to a security object. The *assumed* attribute connects to an *Assumptions* object with data about user preferences and financial assumptions. The *type* attribute is derived from the security's type and controls subclassing. Subclasses of *PyCalc* primarily represent different methods of computing standard analytic results based on the class of security involved.

One of the requirements of the DCS mediator was delivery of a common analytical framework across all kinds of securities. Many of these calculations are specified in detail by industry conventions or regulatory agencies. Since the characteristics of securities vary widely, the parameters and procedures for calculations also vary widely. In some cases, security cash flow must be projected using a predictive model and financial assumptions. For example, an attribute of the *PyMort* subclass controlled the choice of prepayment model used in projecting cash flow for *SecMGen* and *SecPool* securities. If no value was set by the application, the value

for that attribute was obtained from the user's setting in the *Assumptions* object, allowing the prepayment model to be varied without the application's knowledge.

The calculation of price from yield, and yield from price, are so common and important that regulators and industry standards bodies have set down precise rules for computations and precision of results. For bonds, bills, strips, and other straightforward securities, there are closed form analytic formulas for computing present value. Mortgages and mortgage-backed securities require prediction of the cash flow stream using a prepayment model followed by a general discounted cash flow method. Calculating yield from price requires a root-finding algorithm for most securities. Industry standards specify the root finding algorithm and the initial and terminating conditions. As some programmers have learned the hard way, a "more accurate" result is worthless if it differs from what everyone else in the industry expects.

3.2. Integrating databases.

DCS did not include a full query-writing capability for databases. Instead, it relied on pre-coded SQL statements provided in database "retrieval packages." Each package was associated with a class in the conceptual model. Attribute references were specified using path names relative to that base class (e.g., "coupon.rate" based on Security). Each package included a list of attribute paths for required parameters, which could be associated with parameter slots in the SQL statement. Each element in the SELECT clause of the SQL statement, mapped to a corresponding attribute path where retrieved data would be stored. The mediator would not execute the SQL in a package until all required parameter values were present.

3.3. Integrating computational functions

Computational functions and simple arithmetic expressions could be associated with attributes of classes in the domain model. When the value of an attribute was requested, the mediator would first check to see if a value had previously been set or derived. If not, the expression or function would be evaluated and the result value assigned to the requested attribute. Inputs to the calculation were specified as attribute paths based on the containing object. Intermediary results were held in "value packages" along with metadata describing the DCS domain model type. In preparing a computational function call, the mediator consulted the function declaration and performed necessary data conversions. Independently supplied functions did not need to be changed when integrated into DCS, since the mediator would

automatically handle conversion of input data from its original context to the context of the called function. The value returned was held in a value package until the context of its destination was determined.

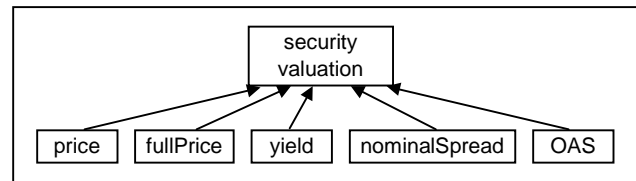


Figure 8. Representations of valuation

DCS required the domain model designer or the function integrator to specify each calculation alternative for mutually dependent attributes. The five attributes of *PyCalc* shown in Figure 8 are financially equivalent representations of security valuation. For example, *fullPrice* is equal to the sum of *price* and *accruedInterest*. Given any two values, the third can be calculated. To represent this situation, all three expressions are declared. If values are specified for none of these attributes, the mediator needed to keep track of attributes that it had visited to avoid being caught in a cycle.

3.4. The DCS API.

The DCS API supplied four basic operations: create or delete object and get or put attribute value. The create operation returned a handle to a runtime object of an entity class in the Conceptual Model. Get and put operations accessed the value of an attribute referenced by a dotted path name relative to an object handle. Programs using the C API would perform a series of put operations to initialize the runtime object with parameters or search criteria. Get operations would then be used to obtain the results of calculations or data retrieval. For the spreadsheet interface, the sequence of put operations was combined into a single object declaration function, which accepted a range of name-value cell pairs. Individual get functions were used in each cell where an output value was required. The C API provided variants of the get and put operations for C data types as well as specially defined types, such as "percent." Where possible and necessary, data values were automatically converted between the C data type used in the application and the representation in a source.

4. An illustrative example.

Figure 9 is a C API code fragment illustrating the use of the DCS mediator. This program finds a security using an industry standard identifier and obtains a descriptive string. Then the valuation is specified as a spread from an

assumed benchmark yield curve and the yield and price calculated. The *dcsNew* function creates a generic *PyCalc* object to serve as a template for accessing the mediator. The *dcsPutString* function sets the CUSIP number of the desired security. CUSIP 887315AT identifies the Time Warner 7.48% debentures due January 15, 2008. To reach the *cusip* attribute in the *Security* object, the programmer uses an attribute path ("sec.cusip"). The string "887315AT" is copied into the *cusip* attribute of a generic *Security* object automatically created by the mediator. The determination of *Security* subclass is deferred.

```

hCalc = dcsNew( "PyCalc" );
dcsPutString( "sec.cusip", hCalc, "887315AT" );
pDescr = dcsGetString( "sec.description", hCalc );
dcsPutBasisPoints( "nominalSpread", hCalc, 275 );
yield = dcsGetPercent( "yield", hCalc );
price = dcsGetFraction( "price", hCalc );
dcsRelease( hCalc );

```

Figure 9. API example

"Put" operations provide constraints on DCS objects in a similar fashion to the WHERE clause in SQL. Values are stored into objects, but no calculations or data retrieval is done until the programmer uses a "get" operation. When *dcsGetString* for *sec.description* is called, the mediator uses the retrieval package requiring the known *cusip* attribute to obtain *assetID*. From *assetID*, *source* and *type* are derived and the *Security* object is subclassed to *SecBond*. A second package based on *assetID* retrieves bond data from the identified source. The wrapper uses an internal put function to store values for attributes retrieved into the security object. The value for description is then returned to the caller.

The *dcsPutBasisPoints* function stores the value of 275 in the *nominalSpread* attribute with metadata indicating basis points representation. Then *dcsGetPercent* asks for *yield* expressed in percent. Finding that nominal spread was given, yield is calculated by adding the spread to the value found at a point on a benchmark yield curve. A yield curve expresses yield as a function of time to maturity. In this example, the yield curve and the point on the curve need to be determined from context provided by a global *Assumptions* object. Our program need know nothing about what assumptions have been set in the Control Panel. Before adding the benchmark yield to the nominal spread, the mediator detects the difference in representation and automatically converts basis points to percent.

The final query in the example is for *price* expressed as a fraction given the security and the yield. The subclass of *PyCalc* is derived from the type of security referenced. In this example, the security is a bond. The *PyBond* subclass specifies calculations for *accruedInterest* and for

fullPrice from *yield*, each requiring data about security characteristics obtained by gets from the *Security* object. The mediator sets up each call, after converting the representation to meet the requirements of the called function, computes *price* as the difference, and converts the final value to the fractional form required by the application programmer.

The *dcsRelease* function tells the mediator that the objects created here are no longer needed and may be deleted.

5. Conclusions.

DCS proved the concept of context mediation in the pragmatic world of an information-intensive and highly competitive industry. When the original design was presented to clients, response was enthusiastic. Developers liked the idea of integrating data and calculations, allowing them to avoid the work of connecting components together in software. They found the notion of treating computational results as data to be appealing.

Producing a mediator for production business use required a major focus on quality assurance and training. The major quality assurance effort was not directed at the mediator itself. Clients saw the results delivered through the mediator as the responsibility of the mediator. In consequence, the mediator development team spent most of a year testing and fixing calculation software as it was integrated. Considerable effort was also devoted to documentation and training, including a manual on domain model design using a tennis shoe sales example.

The DCS context mediator was similar in some ways to an object broker approach. The difference was a design based on declarative knowledge used to dynamically decide on methods to materialize elements of the conceptual model.

For some users the structure imposed by the standard conceptual model was helpful. More advanced developers and client shops, however, wanted to be able to specify their needs as user views or in their own conceptual models. The mediator would need to go beyond context conversion for individual pieces of data, to be able to map the central domain model into the structure of client views and models. Clients also wanted to be able to treat the components of the domain model as building blocks that could be arranged to suit their purposes. For example, clients would have liked to pose queries that combined entities as needed rather than using pre-planned relationships. A query language like SQL or OQL would have been helpful in place of the cumbersome API.

The DCS mediator offered a new paradigm for financial application development. Traditionally, the

application programmer retrieves data and sets up calls to calculation libraries. With DCS the application programmer works at a higher level of abstraction, in terms of the results of applying analytic methodologies. The mediator takes responsibility for finding and meshing the right programs and data to fulfill the application's needs.

By decoupling applications from the methods and data used in computing analytic information, DCS allows plug 'n play dynamic substitution of functionally equivalent components. For example, mortgage prepayment models are critical to projecting performance on portfolios holding derivative securities. When a Wall Street firm refines its prepayment model, it wants to make that new knowledge available to its customers quickly. Buy-side portfolio managers may have preferred portfolio analysis application tools that can gain immediate benefit from the new model delivered through the mediator. The mediator can also facilitate the sell-side analyst in developing proposed trades by bringing buy-side portfolio holdings data into the analyst's applications. Improving information flow between buy-side and sell-side can also improve capital market efficiency.

Experience with DCS suggests the potential value of further research on query processing reasoning algorithms for mediating among application developers' conceptual models and component supplier models using industry-specific abstract knowledge.

References

- [1] G. Wiederhold, "Mediators in the architecture of future information systems." *IEEE Computer*, March 1992, pp. 38-49.
- [2] G. Wiederhold, "Mediation in information systems," *ACM Comp. Surveys* vol. 27, no.2, June 1995, pp. 265-267.
- [3] S. Madnick, "Are we moving toward an information superhighway or a Tower of Babel? The challenge of large-scale semantic heterogeneity," *Proc. IEEE International Conference on Data Engineering*, 1996, pp. 2-8.
- [4] C. Goh, "Representing and reasoning about semantic conflicts in heterogeneous information systems", Ph.D. Thesis, MIT Sloan School of Management, 1996.
- [5] C. Goh, S. Bressan, S. Madnick, and M. Siegel, "Context interchange: representing and reasoning about data semantics in heterogeneous systems," MIT Sloan School of Management Working Paper #3928, Dec. 1996.
- [6] M. Siegel and S. Madnick, "A metadata approach to resolving semantic conflicts," *Proc. 17th International Conference on Very Large Data Bases*, 1991, pp. 133-145.
- [7] Y. Arens and C. Knobloch "Planning and reformulating queries for semantically-modeled multidatabase," *Proc. of the Intl. Conf. on Information and Knowledge Management*, 1992.

-
- [8] A. Levy, A. Rajaraman, and J. Ordille, "Query answering algorithms for information agents," *Proc. 13th National Conf. on Artificial Intelligence*, Aug. 1996, pp. 40-47.
 - [9] M. Genesereth, A. Keller, O. Duschka, "Infomaster: an information integration system." *Proc. 1997 ACM SIGMOD Conference*, May 1997, pp. 539-542.
 - [10] Y. Papakonstantinou, A. Gupta, and L. Haas "Capabilities-based query rewriting in mediator systems," *Proc. 4th Intl. Conf. on Paralled and Distributed Information Systems*, 1996.
 - [11] H. Garcia-Molina "The TSIMMIS approach to mediation: data models and languages," *Proc. Conf. on Next Generation Information Technologies and Systems*, 1995.
 - [12] D. Woelk, P. Cannata, M. Huhns, W. Shen, and C. Tomlinson, "Using Carnot for enterprise information integration", *Second International Conf. on Parallel and Distributed Information Systems*, Jan. 1993, pp. 133-136.
 - [13] D. Woelk, W. Shen, M. Huhns and P. Cannata, "Model driven enterprise information management in Carnot", in Charles J. Petrie Jr., ed., *Enterprise Integration Modeling: Proc. of the First International Conference*, MIT Press, Cambridge, MA, 1992.
 - [14] M. Benaroch, "Toward the notion of a knowledge repository for financial risk management," *IEEE Trans. on Knowledge and Data Engineering*, vol. 9, no. 1, Jan. 1997, pp. 161-167.
 - [15] Standard & Poors CUSIP Service Bureau <http://www.cusip.com/>