An Active Conceptual Model for Fixed Income Securities Analysis for Multiple Financial Institutions

Allen Moulton, Stéphane Bressan, Stuart E. Madnick, Michael D. Siegel

Massachusetts Institute of Technology, Sloan School of Management,
Cambridge, Massachusetts 02139
{amoulton, smadnick, msiegel}@mit.edu
steph@context.mit.edu

Abstract. The practical implementation and use of a mediator for fixed income securities analysis demonstrated the potential for extending the application of conceptual modeling from the system design stage to providing query access to both data and computational resources. The mediator product was designed as a general interpretive engine specialized and controlled by the declarative knowledge from the conceptual model. The fixed income conceptual model included securities ranging in complexity from Treasury bills to collateralized mortgage obligations, as well as standard and proprietary analytic methodologies and calculations. All information, whether computed or retrieved from databases, was offered to clients in the form of attributes of conceptual entities. Client preference entities and attributes were used to control selection among conceptually interchangeable source data and procedural components. Experience from this implementation provides insight into the requirements for successful application of a conceptual model in mediating among heterogeneous, autonomous users and information resources.

1 Introduction

Effective fixed income investment decision-making involves integrating information from internal firm sources with industry standard information and specialized proprietary knowledge, all within a common framework of abstract analytic methodologies. Representing the information structure of abstract methodologies in high level conceptual models enabled the Data and Calculation Services (DCS) mediator to combine heterogeneous, autonomously controlled information resources needed by decision-makers. Decision support application developers could write queries directly against the conceptual model, with the mediator responsible for finding and combining component resources. The conceptual model provided a database-like framework for integrating databases, analytic calculations, and predictive models. End-users could independently control implicit assumptions and selection of resource components used by their applications. The mediator enabled the interchange and combination of information resources across organizational boundaries, while responding to the rapid

evolution of financial products, transactions, and markets characteristic of the fixed income securities industry.

The challenge of the DCS mediator 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. Government regulators and industry associations, such as the Securities Industry Association[4] and the Public Securities Association[5], develop standards for terminology and generic analytic measures. Publications, such as Fabozzi's series of books (*viz.* [6]) disseminate theories and practices. Beyond the basics, however, firms add value by developing richer, more complex predictive models for security valuation and portfolio management. Context mediation can help integrate industry and regulatory standards with proprietary data and analytic methods to serve the needs of the investing public.

In COIN[7,8,9], 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 traditional use of conceptual modeling is in the planning and design of information systems and databases[10,11]. The DCS mediator was a practical experiment in making a conceptual model actively available for query as envisioned by Markowitz and Shoshani [12] and Parent *et al.*13]. The DCS conceptual model served as a domain model for fixed income decision-making information. SIMS[14], Information Manifold[15], Infomaster[16], and Garlic[17] employ a domain-specific ontology. As in these projects, DCS integrates data sources through a shared model of knowledge specific to the problem domain. DCS differs from the approach of TSIMMIS[18], which uses an ontology-free approach where data is integrated and delivered in its own terms, from Carnot[19] which draws on an ontology of general common-sense knowledge, and from Benaroch's[20] 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 issuers, information about markets, information about

economic conditions and events, and information about methodologies and models (see Fig. 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.

Fixed income securities, such as bonds, 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. Optional events may affect the cash flow stream and there may be risk of default. In essence, however, all fixed income securities are interchangeable investment vehicles. 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. Repackaging, or "financial engineering" produces "derivative" securities. 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"

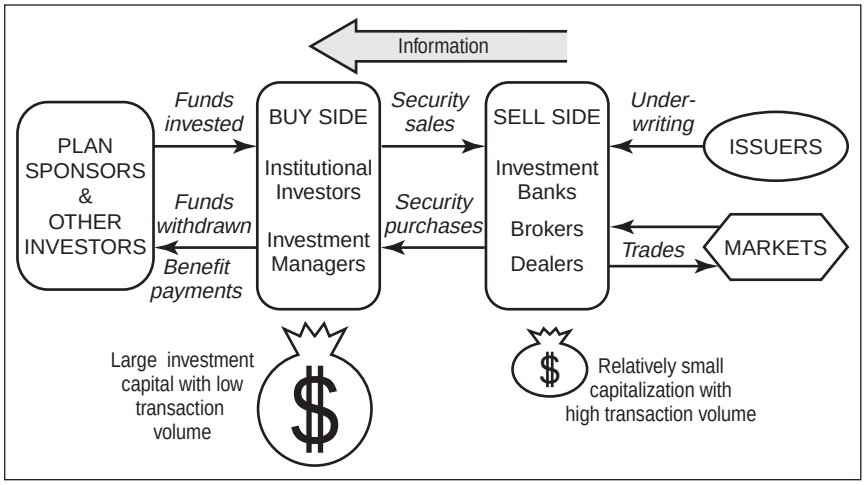

Fig 1. The fixed income securities industry.

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 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.

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.

A Venture partner 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. 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.

3 Architecture Of The DCS Mediator

The DCS mediator (see Fig. 2) was designed as a demand-driven general interpretive engine controlled by a static declarative knowledge. The core of the declarative knowledge was a conceptual model of fixed income data and analytic methodologies. Action specifications provided the means for realizing entities and attributes of the conceptual model from heterogeneous resources supplied by autonomous organizations. Declarative knowledge was prepared in a definition language script compiled into a

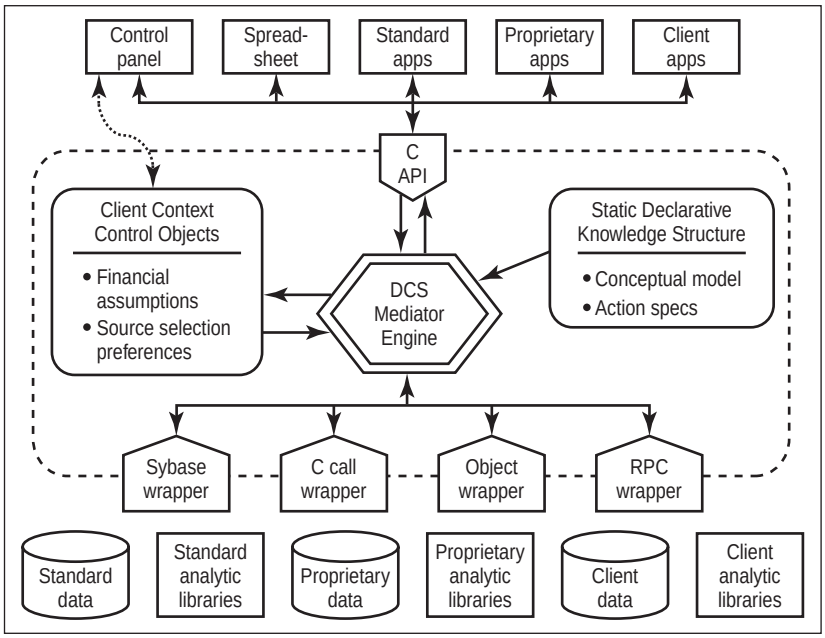

Fig. 2. Architecture of the DCS Mediator.

binary structure for run-time 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 financial assumptions were represented in context control objects that could be viewed and manipulated though a Control Panel application. By changing values in context objects, the user could control 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 Conceptual Model Structure

The DCS conceptual model defined the entities, attributes and relationships available through the mediator and served as a shared ontology for integrating heterogeneous, autonomous data and calculation resources. A high level conceptual model for fixed income securities analysis was published as the principal documentation of the functionality available through the mediator.

Fig. 3 shows an example of conceptual model elements offered in DCS. Entity class A is a subclass of a parent class P, depicted by a hollow arrow. DCS supported single inheritance and polymorphism with subclass determined dynamically from an attribute

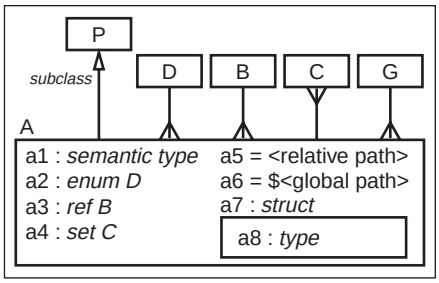

Fig. 3. DCS conceptual model elements.

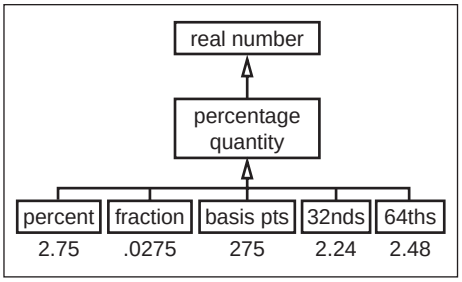

Fig. 4. Semantic types for percentage quantities.

in the parent class. Class A has six attributes (a1…a6) and a substructure a7 containing an additional attribute a8. Attribute a1 illustrates a data attribute. Semantic type captures properties of data at the conceptual level, while a data type stands for its physical representation. Both types are associated with data values during mediation. Fig. 4 shows a semantic type hierarchy for several representations of percentage quantities. The five different numbers at the bottom all represent 2.75%. The mediator provided automatic conversion among compatible semantic and data types.

Attribute a2 in Fig. 3 refers to an enumerated domain D. Arbitrary codes are often used to represent properties of entities. An enumerated domain could list the codes used in the conceptual model or codes used in a particular resource context. Mapping relations associated context-specific codes with conceptual model codes, allowing the mediator to convert codes across contexts. An attribute declared on an enumerated domain may also be used as a role name for a many-to-one relationship.

Attribute a3, a reference to an object of class B, is a role name for the many-to-one (or one-to-one) relationship with class B. Attribute a4, similarly, is a role name for the one-to-many relationship with class C. Many-to-many relationships require an intervening entity. Attributes a5 and a6 are equated to other attributes reachable by a relative(a5) or global(a6) path.

A path is a sequence of attribute names separated by dots, representing the traversal of a relationship. Each step, except the last, in a path must be the name of an attribute which refers to another object or to an enumerated domain. A relative path begins with the current object. For example, attribute a8 can be reached from an object of class A with the relative path “a7.a8.” A global path (indicated by a leading “$”) begins with the user’s global object G. Every object has an implicit many-to-one relationship with the global object.

3.2 Action Specifications

Action specifications provide procedures for deriving output attribute values from input parameters (see Fig. 5). An action may be a database retrieval, a computational function, or an arithmetic expression. Each action is associated with a class in the conceptual model (the base class for the action). Attribute reference paths may be either relative to the action base class or global. Each action may have enabling conditions which serve

as constraints that must hold for an action to be applicable. For the initial implementation, enabling conditions were limited to equality constraints specified as pairs of attribute paths and constant values.

Database retrieval actions are defined by an SQL statement with slots for input parameter values. Output attribute paths in the same order as the SELECT clause are specified in an INTO clause. For single row retrievals, the action sets values into attributes of a single object of the base class and related objects. For set valued retrievals, an additional output attribute defined as a set of base class objects must be declared. Semantic types of attributes obtained from a database retrieval action are defined by the conceptual model. Data types are determined from the database query output.

Computational functions may be legacy or new code, available in a local library or through a remote procedure call on a server. Functions can return a single value to multiple values through output arguments. . A function must return the same output values for each call with a given combination of inputs. Internal state may not affect outputs. Functions are declared to the mediator with both data and semantic types for each value and parameter. Action specifications match input and output attribute paths to function arguments and value. Expression evaluation actions use a built-in interpreter for simple arithmetic expressions. Independently supplied functions need not be changed when integrated into DCS, since the mediator automatically converts of input data to the semantic and data types required by the function. Values returned retain the function-defined semantic and data types until conversion is required where used.

The network of action inputs and outputs constitutes a dependency graph among attributes of the conceptual model. At a high level of abstraction, there may well be several different ways of representing functionally equivalent information. To allow heterogeneous sources, receivers, and computations to each use their preferred representation, the conceptual model may need to include functional redundancies resulting in potential cycles in the action dependency graph. Fig. 6 illustrates a cycle resulting from mutual dependency of the *fullPrice* and *price* attributes. To cope with some frequently occurring cycles, the DCS mediator provides for designation of groups of mutually determined attributes. This information is used to detect overspecified criteria in a query and to avoid trying actions known to result in cycles.

3.3 Mediation Algorithm

The mediator algorithm provided the appearance to the user of a database capable of materializing all elements of the conceptual model. Queries are based on a single object with attributes of related objects reached using relative paths. The mediator processes queries of the form:

```
SELECT <list of attribute paths>
FROM <class>
WHERE <conjunction of path=value criteria>
```

A query proceeds by creating a working object of the given class, “putting” values specified in the criteria into attributes of the object and related objects reached through it, then “getting” values for each selected attribute path from the object. Related objects

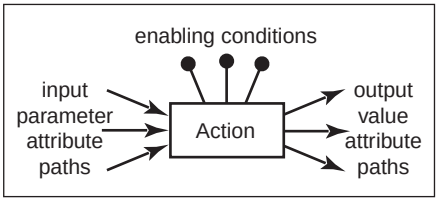

Fig. 5. Action specifications.

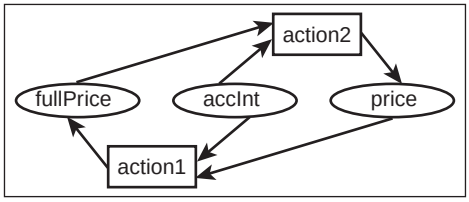

Fig. 6. Dependency graph cycle.

are automatically created when referenced. Each intermediary and final result value is held in a "semantic value package" structure together with its semantic and data types. The query logic follows a depth-first approach checking for cycles to avoid infinite recursion.

The logic of the mediator can be understood by describing the "get" algorithm. Get is a recursive procedure that takes a reference object and a path and follows a depth first search strategy to find a value. The attribute path is processed in sequence, getting each object reference and applying get recursively to the remainder of the path until only one element remains. When the path has only a single attribute name, get first checks the conceptual model for an equivalence declaration. If the attribute is as equated, a recursive get is performed using the equated path. If the value of the attribute has previously been determined, it is returned. If not, the mediator attempts to find an action that can supply the requested value. Since the dependency graph for actions may well have cycles, the mediator marks the attribute as "in use" before proceeding. If another get operation is attempted on an attribute that is so marked, the operation fails.

To derive a value for the attribute, the mediator uses the dependency graph to attempt each action capable of providing the required attribute as an output. Actions are tried in the order they are declared. When an action succeeds, the value is returned and alternative actions ignored. Each action may have enabling conditions, which require an attribute to match a specified constant value. For each enabling condition, a recursive get is performed on its attribute path based on current object. If the get fails or the returned value does not match, the action is cancelled and the mediator moves to the next action.

If all enabling conditions hold, the mediator attempts to obtain values of all input parameters by calling get recursively for each. If the get on any input parameter fails, the action is discarded. If all input gets are successful, the mediator compares the semantic data types of the values returned to the types required for the action. Incompatible types cause the action to be discarded. If either semantic or data types are compatible but different, the mediator calls an appropriate semantic or data type conversion.

Once all the inputs are collected, the mediator calls the appropriate wrapper to execute the action. A failure in the wrapper causes the action to be cancelled. The database wrapper prepares the SQL statement by substituting input values where marked. The statement is then sent to the database instance specified in the action. Each value returned is put using the path from the INTO list in the action specification. Data type is determined by the type returned from the database; semantic type, from the conceptual model. The function wrapper sets up and executes a local or remote procedure call. On

return the function value and all output argument values are put to the path specified in the action specification. For functions, semantic and data types are given by the function declaration.

When any action succeeds, get returns the value after storing the attribute value in the containing object. If all actions fail, get returns failure. In either case, the attribute is marked "not in use." After the recursive call stack is unwound, the mediator converts the result value to the user-specified semantic and data types.

4 Applying the DCS Mediator to Fixed Income Securities Analysis

Fixed income investment decision-making involves buying, selling, and swapping securities in order to improve the performance of an investment portfolio while meeting the sponsor's objectives. Decision-makers use both general methodologies applied across all kinds of securities and specialized analytic techniques applicable to particular market segments. Analytic support tools must be extensible for new kinds of securities invented by financial engineers and for new analytic models within the structure of common methodologies. Decision-makers also need access to and control over assumptions, models and analytic components used.

The DCS mediator offered a new paradigm for analytic application development. Traditionally, the application programmer is responsible for meshing database retrievals and calculation libraries. With DCS the application programmer works at a higher level of abstraction focusing on the results of applying analytic methodologies. The mediator takes responsibility for finding and integrating the right data and programs to fulfill the application's needs.

4.1 Developing the Fixed Income Conceptual Model

After about a year of design and development, a beta version of the DCS mediator was made available to application developers in 1991. Product release was in mid-1992. After the initial development of the mediator engine, the efforts of the DCS development team focused primarily on extending the domain model, verifying the correctness of calculations and data integrated through the mediator, and documentation. Documentation included "placemat" diagrams of object classes similar to the small samples in Fig. 8, along with explanations of the semantics of each attribute. A glossary of fixed income terminology (still available on the web at [21]) clarified definitions and semantics as used in the domain model.

Fig. 7 illustrates part of the abstract conceptual model for fixed income securities and analytic methodologies. The Security entity represents a generic concept of a fixed income security. The PyCalc (price-yield calculator) entity represents an actual or hypothetical trade in a security together with internal rate of return (IRR) and option adjusted spread (OSA) analytic methodologies applied to the trade. At the abstract level, the Security has an issuing financial entity represented by Issuer and end-of-day prices from various sources in a set of Valuation entities. The global Assumptions class

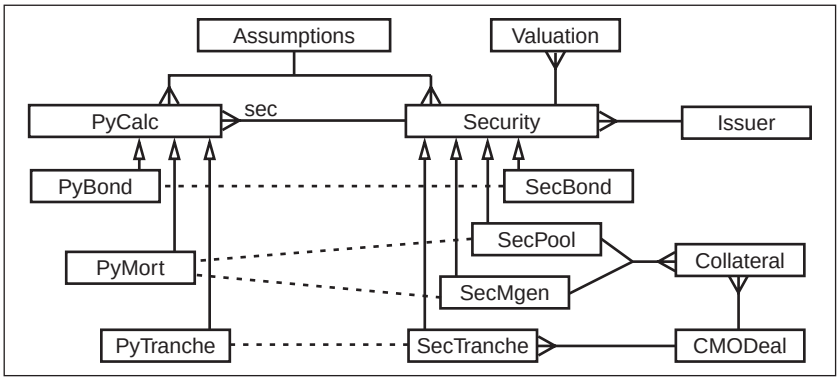

Fig. 7. Fixed income conceptual model fragment showing polymorphic subclass interrelationships.

provides context information on financial assumptions and client selection of data sources and analytic models.

Polymorphic subclasses (shown with hollow arrows pointing to the parent generic class) represent different information structures for Security and different analytic implementations for PyCalc. In each case, the subclass is determined dynamically from the value of the meta-attribute *type* 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 a collateralized mortgage obligation (CMO). For the SecTranche subclass, the security is related to a CMODeal 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 action specifications for performing the functionality of the abstract class. The subclass of PyCalc is derived from the type of Security entity referenced: PyBond for bonds, PyMort for mortgage generics and pools, and PyTranche for CMO tranches. In some cases, a subclass may extend the generic PyCalc functionality with additional measures specific to one type of security.

Fig. 8 shows some of the attributes of the generic PyCalc and Security classes in "placemat" form used in the documentation. The *sec* attribute in PyCalc provides a reference to a Security. The *type* attributes in each class control subclassing, with PyCalc *type* derived from Security *type* by an enumerated domain relation. Security *type* may

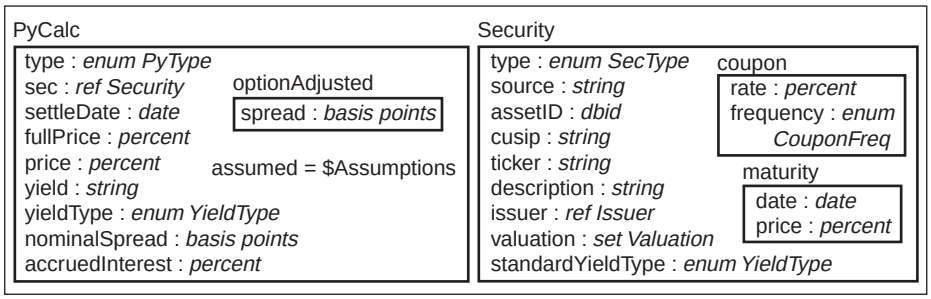

Fig. 8. Fragments of the "placemat" documentation for the generic PyCalc and Security classes.

be implied (because a source is limited to a single type) or derived from stored information. The *optionAdjusted* part of PyCalc is also polymorphic based on security type and selection of OAS model in Assumptions.

A Security object may be used to represent 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[22] is an industry standard identifier used in the United States and Canada. An *assetID* was an artificially constructed globally unique key for all security data objects accessible through DCS. Venture-supplied security databases were designed with *assetID* attribute for each security. 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.

4.2 The DCS Mediator at Work

The following simple query illustrates an application request for a description string, yield and price for a trade in a security identified by CUSIP, settling on November 19, 1998, with valuation specified as a nominal spread of 275 basis points from an assumed benchmark yield curve.

```
SELECT sec.description:string, yield:percent, price:fraction
FROM PyCalc
WHERE sec.cusip="887315AT" AND settleDate="11/19/98"
    AND nominalSpread=275
```

The DCS mediator begins by creating a generic PyCalc object and putting the values given for the three attributes in the WHERE clause. A generic Security object is automatically created in traversing the path *sec.cusip*. Attempting to obtain *sec.description*, the mediator must determine the security type and source. The mediator uses a database action requiring the known *cusip* attribute to obtain *assetID*. From *assetID*, *source* and *type* are derived. Since CUSIP 887315AT is a bond, the Security object is subclassed to SecBond. A second package based on *assetID* retrieves bond data from the identified source and puts it in attributes of the SecBond object. If the source does not provide a value for the *description* attribute, a function action constructs a description string from other attributes. The value returned would be:

TIME WARNER 7.48% DEB 01/15/2008

As illustrated in Fig. 9, *yield*, *price*, and *nominalSpread* are alternative representations of security valuation, each mutually computable from one another. The example query specified *nominalSpread*. The action for computing *yield* from *nominalSpread* involves adding the value given for the spread to the yield value found at a point on a benchmark yield curve, which is derived from a scenario object in the global Assumptions object. By setting a scenario using the Control Panel, the user determines the contextual meaning of nominal spread in all applications. After calculating *yield*, the mediator converts the value, if necessary, to the percent form requested in the query.

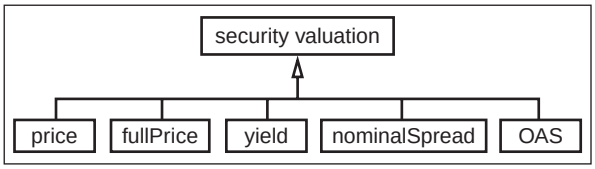

Fig. 9. Alternative representations of security valuation.

Computing *price* from *yield* requires an algorithm specific to the type of security. Based on the security type of bond, PyCalc was subclassed to PyBond. The PyBond subclass has action specifications for computing *accruedInterest* and for *fullPrice* given *yield*. Each action requires *settleDate* and security characteristics attributes from the SecBond object. The mediator sets up each call, after converting the representation to meet the requirements of the called function, then computes price as *fullPrice* less *accruedInterest*. As a final step, the value for *price* is converted, if necessary, to the fractional form required by the user query.

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 has the added benefit of improving capital market efficiency.

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 requires quality assurance and training, yet 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 verifying data and 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.

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. Use of UML[23] modeling capabilities and OQL[24] as a query language would have been helpful.

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. Since the DCS mediator was built, Goh[7] has examined abductive logic programming for context conversion of semantic types. Our present research involves extending the abductive reasoning paradigm to multiple levels of abstraction in conceptual models.

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. Securities Industry Association. New York. http://www.sia.com
5. Public Securities Association. New York. http://www.psa.com
6. F. Fabozzi, ed., The Handbook of Fixed Income Securities, 3rd ed., Irwin, Homewood, Illinois, 1991.
7. C. Goh. "Representing and reasoning about semantic conflicts in heterogeneous information systems." Ph.D. Thesis, MIT Sloan School of Management, 1996.
8. C. Goh, S. Bressan, S. Madnick, and M. Siegel. "Context interchange: new features and formalism for the intelligent integration of information." MIT Sloan School of Management Working Paper #3941, Feb. 1997.
9. 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.
10. C. Batini, S. Ceri, and S. Navathe. Conceptual Database Design. Redwood City, Calif.:Benjamin/Cummings, 1992.
11. T. Teorey. Database Modeling and Design. San Mateo, Calif.: Morgan Kaufmann, 1990.
12. V. Markowitz and A. Shoshani. "Abbreviate query interpretation in entity-relationship oriented databases." Proc. 8th Int'l. Conf.. on Entity-Relationship Approach, 1989.
13. C. Parent, H. Rolin, K.Yétongnon, S. Spaccapietra. "An ER calculus for the entity-relationship complex model." Proc. 8th Int'l. Conf.. on Entity-Relationship Approach, 1989.
14. Y. Arens and C. Knobloch. "Planning and reformulating queries for semantically-modeled multidatabase." Proc. of the Intl. Conf. on Information and Knowledge Management, 1992.
15. 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.
16. M. Genesereth, A. Keller, O. Duschka. "Infomaster: an information integration system." Proc. 1997 ACM SIGMOD Conference, May 1997, pp. 539-542.

17. 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.
18. H. Garcia-Molina. “The TSIMMIS approach to mediation: data models and languages.” Proc. Conf. on Next Generation Information Technologies and Systems, 1995.
19. 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.
20. 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.
21. Bridge Info. Systems, “EJV Partners Financial Glossary” http://bondchannel.bridge.com/html/EJVGlossary.html
22. Standard & Poors CUSIP Service Bureau http://www.cusip.com
23. Rational Software Corp. http://www.rational.com/uml/index.shtml
24. R. Cattell (ed.). The object database standard: ODMG-93. San Francisco: Morgan Kaufmann, 1996.