

# Integrating XML and non-XML Data via UML

David Carlson <dcarlson@ontogenics.com>

## Abstract

As the use of XML matures within our systems development toolkit, we need a better approach for integrating its schema definitions with other analysis and design activities. The Unified Modeling Language (UML) is described as a useful solution that breaks down walls separating development activities and technologies. A financial derivatives trade application is described where the FpML vocabulary is imported into UML from its XML Schema source, and this is integrated with a Trading Party vocabulary imported from a SOX schema included in xCBL. These XML data definitions are then linked with a relational database schema imported into the same UML model. All of these data definitions are integrated as part of a simple portal application for trade confirmation. The iterative design approach illustrates benefits of UML for rapid analysis and design of new e-business applications that include XML content in part of their design.

## 1. Introduction

XML has entered the mainstream. No one can dispute the fact that it is being designed into a wide variety of systems for large-scale integration within and between businesses. This is especially true in the financial services industry, where many different industry standards have been proposed and specified that use XML to exchange data about business reporting, financial products and transactions, insurance claims processing, etc. In addition, efforts such as ebXML are attempting to create large repositories of reusable components that are intended to support a general e-business infrastructure.

But these XML-based data are not used in isolation from the legacy systems that support front and back-office operations. Our current software development processes used for systems analysis and design must be augmented to enable these new XML data models to be understood as part of complete system architecture. This article describes how the Unified Modeling Language (UML) can support the analysis and design of large systems where one or more XML schemas are integrated with other information assets.

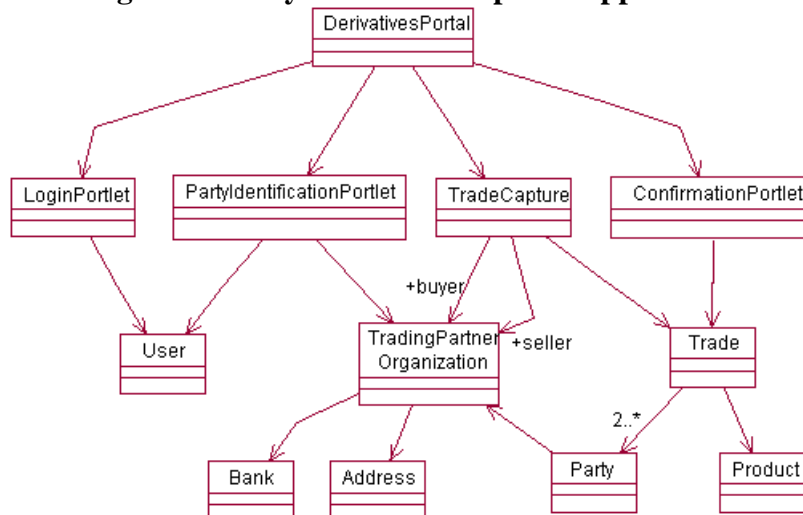
UML is the most recent evolution of graphical notations for object-oriented analysis and design, plus it includes a metamodel that defines the modeling language itself. Contrary to popular opinion, UML is not limited to its use as a tool for programmers. The UML defines several model types that span a range from functional requirements and activity workflow models, to class structure design and component diagrams. This article focuses on use of class diagrams to model the static structure of XML schemas and their integration with other non-XML sources of data and object interfaces.

## 2. Analysis of a Trade Portal

Many front-office financial service systems include a web portal that aggregates applications and information content for both internal business users and customer business partners. Several case studies have been described where the Financial Products Markup Language (FpML) XML vocabulary is used to create portals for trade confirmation and others are being considered for trade capture and pricing applications [FpML]. The hypothetical example described here was created to illustrate the potential for using UML to analyze and design a portal that integrates three different data sources.

We'll begin the example with a high-level UML class diagram depicting the initial analysis of our system requirements. This is a common use of UML models to get user feedback in a quick interactive development process. The entire analysis process described in this article might be completed in less than one week. The diagram in Figure 1 illustrates our initial concept, where a financial derivatives portal is composed of four "portlets", each of which depends on several domain data objects. A portlet implements the control and presentation for one piece of the application.

**Figure 1. Analysis model of a portal application**



As a next step we need to find sources for these data objects and refine this first analysis model into a complete design. We already know that financial derivative products represented by the FpML vocabulary will be used to refine the Trade, Product, and Party classes on the right-hand side of the diagram. However we also know that a Party in FpML is little more than a reference to a standardized identifier. We need to resolve this reference to a complete organization description drawn from another source.

For the organization information we'll turn to the Trading Partner schema definition from the XML Common Business Library (xCBL). At the time this was written, xCBL is being considered as a primary input to the OASIS Universal Business Language (UBL) [UBL] technical committee for possible inclusion in the ebXML core component library. It may become possible to send a web services message to a well-known repository, which queries an organization ID and returns a Trading Partner XML message. In anticipation of this design, we'll import the current

xCBL vocabulary definition into our portal design model. The most complete reference specification of xCBL 3.0 is available in the SOX schema language, so we'll need to reverse-engineer that language into UML.

Finally, to further underscore UML's potential for integrating diverse data assets, we'll import an Oracle relational database schema that supports user authentication and permissions. This design was created within an open-source Apache Jakarta project [[Turbine](#)] and is currently deployed in other portal systems, such as Apache Jetspeed [[Jetspeed](#)]. Thus, the resulting UML model contains data models from three diverse source languages -- W3C XML Schema, SOX, and relational DDL -- and all are mapped into a common representation and graphical notation.

This article places most emphasis on conceptual analysis of these data sources and their interrelationships, rather than on the specific details of how the implementations of these data models are mapped to and from UML. See the bibliography for more detail. The conceptual data model in UML includes the classes, attributes, and associations that define a static structural model for each schema. These classes are organized into UML packages that define clusters of related data, but associations may also exist between classes in different packages, thus representing inter-schema dependencies for system integration. The most obvious top-level mapping is from one XML or database schema to one UML package. However, it's often helpful to decompose an XML schema into reusable and optional modules, such as was done in the recent modularization of XHTML. Each schema module defines a distinct UML sub-package.

### 3. Reverse Engineering Schemas into UML

This article describes the benefits of reverse engineering existing schemas into UML, which is closely coupled with the approach used to generate schemas from UML, often called forward engineering. Both depend on a rigorous mapping between the source and target languages. The details of how this mapping is specified and implemented are beyond the scope of this article, but instead we review the resulting UML models and their ability to communicate the schema structures and use within a larger system integration project. We use the XML Metadata Interchange (XMI) [[XMI](#)] standard for UML models as the basis for reverse and forward engineering XML schemas. In principle, this enables tool independence, but there are still wrinkles to be worked out in a few compliance issues. Most important for reverse engineering, the XMI representation of UML graphical diagram layout is still under development and is not fully interchangeable across tools.

The reverse engineering process is part automated and part manual. Importing schemas into a UML tool (I'm using Rational Rose) is fully automated, however the automated diagram layout needs some manual follow-up work to create diagrams that are easily readable and that effectively communicate the model's purpose. The quality and style of auto-layout varies greatly across UML tools. Also, many XML schemas are very large (xCBL is huge) and need to be decomposed into several views based on a logical modularity that is present in most vocabularies. Each UML diagram communicates a view of the complete model, so either a large vocabulary must be decomposed into modules before importing them into UML, or the large diagram must be manually split into two or more views from within the UML tool. I used a combination of these approaches with the models presented in this article.

Returning to the portal application, we identified three different schemas that are implemented in three different languages. We'll apply two different mapping specifications for importing them into UML. The XML schemas are mapped according to the specification contained in Appendix C of *Modeling XML Applications with UML* [Carlson]. This mapping has been most frequently applied to the W3C XML Schema, but the same rules can be applied to SOX schemas used by the xCBL Trading Partner vocabulary. We have also used this mapping for DTD and RELAX NG schemas.

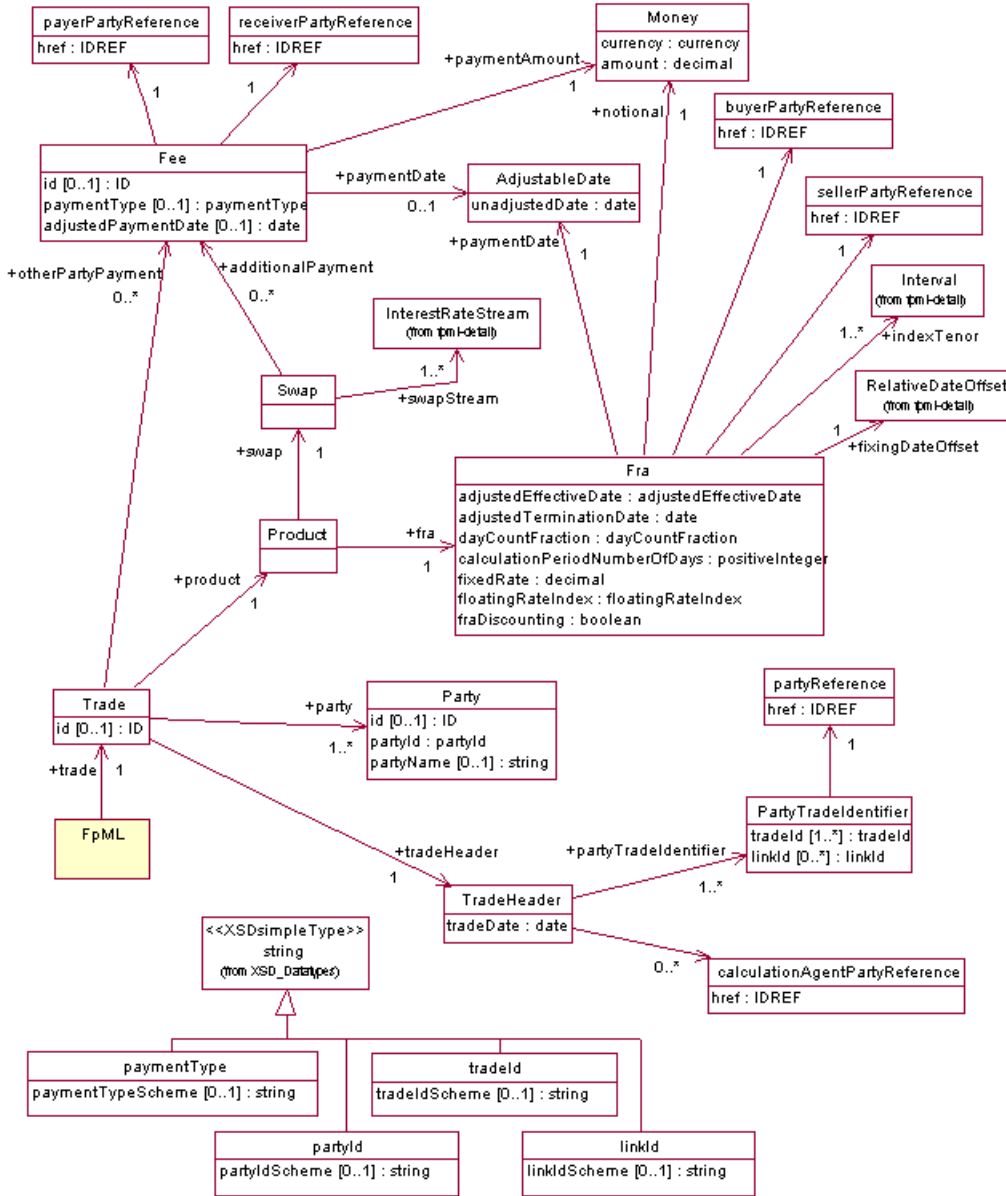
The mapping from a relational database schema to UML is accomplished via the built-in tool shipped with the Rational Rose modeling tool, version 2001A. This use of UML for database design is also described in a book written by two authors from Rational Software [Naiburg]. A two-stage mapping is executed, where importing a database schema produces a UML model that closely mimics the relational structure, including stereotypes for primary keys and methods for referential integrity triggers. A second transformation is then created within the tool to yield an object model, which could be used to produce outputs other than a database schema. This latter object model is used in the following discussion.

### 3.1. FpML Schema

The Financial Products Markup Language (FpML) is a new protocol enabling e-commerce activities in the field of financial derivatives. The development of the standard, controlled by market participant firms, will ultimately allow the electronic integration of a range of services, from electronic trading and confirmations to portfolio specification for risk analysis. All types of over-the-counter (OTC) derivatives will, over time, be incorporated into the standard, although the initial focus is vanilla interest rate derivatives.

To produce this UML model, I first imported the FpML 1.0 XML Schema into UML and reviewed the resulting diagram. Strictly from a visual perspective, I could see three clusters of related element types that were more heavily linked within each group, but had relatively few links between the groups. I then edited the schema file to divide it into three sub-schemas according to this rough modularity and re-imported them into three UML packages, each with its own class diagram that contains links to elements from the other packages. The "core" package is shown in [Figure 2](#), which contains the top-level elements used in all FpML messages.

**Figure 2. Subset of the FpML 1.0 vocabulary**



A primary strength of UML diagrams in this situation is that a person who has no knowledge of XML schemas and relatively little knowledge of financial products can still gain a quick grasp of this vocabulary's structure. This diagram notation is also standardized [UML], so many developers are already familiar with it. Because I intend for this model to be used at the level of conceptual analysis for system integration, I hid additional UML stereotypes and properties that fully specify the XML schema structure. In fact, I can use this model (with the hidden properties) to regenerate a W3C XML Schema that successfully validates all test documents distributed as part of the FpML specification.

A schema designer would need visibility of the detail to refine this schema's validation capability, but a business analyst can more easily critique and apply this model from the conceptual view. For the purpose of our portal design example, we can identify the Trade, Party, and Product elements and review the other data that are available to populate the portlet displays for trade

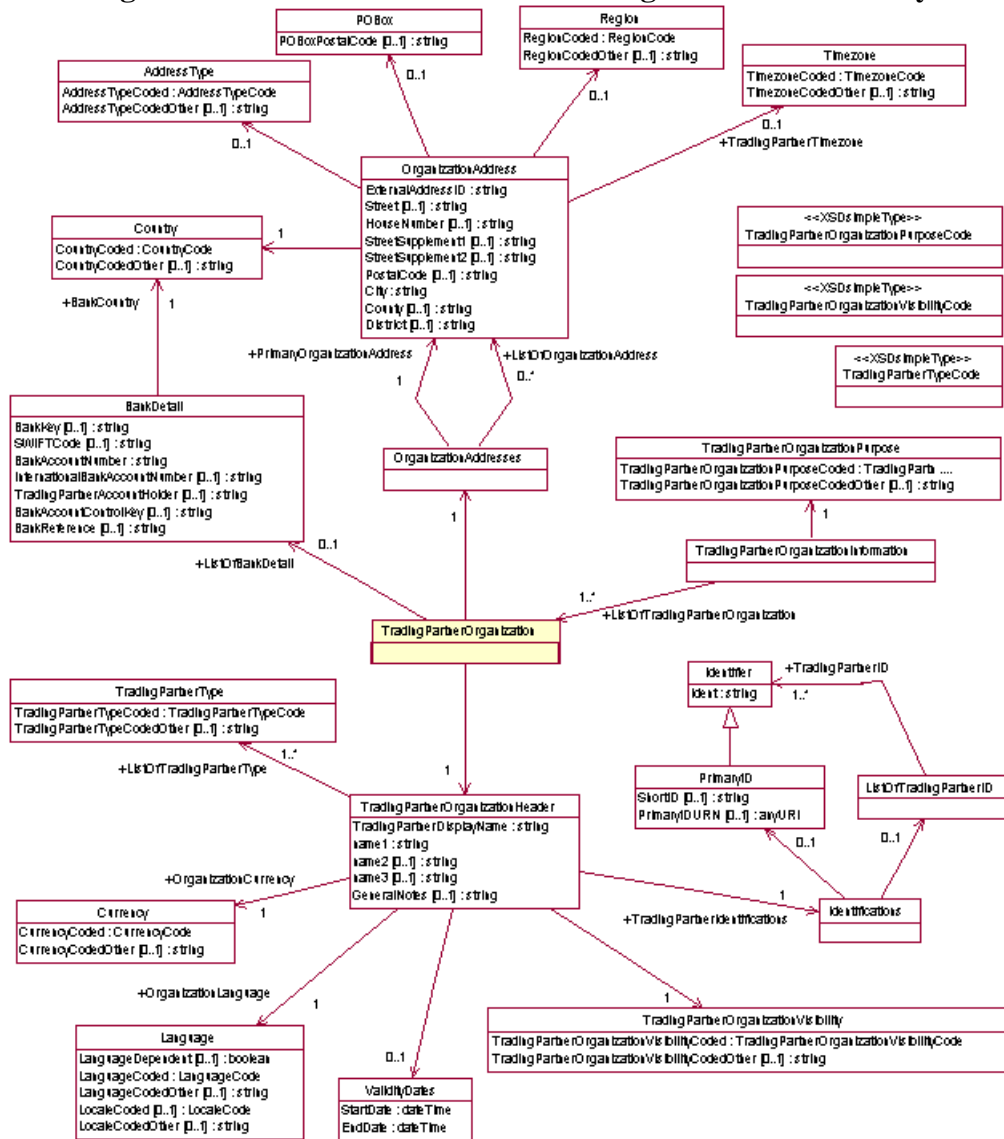
capture and confirmation. Be aware, however, that about 70% of this model's detail is contained in other views not included in this article. I use the convention that classes which hide all internal detail (i.e. contain no subcompartments) usually contain attributes or references to other model elements. For example, InterestRateStream includes many additional associations to detailed content shown in other diagrams.

### **3.2. xCBL TradingParty Schema**

The approach used to import the Trading Party schema is very similar to that used for FpML, with a few significant exceptions. The xCBL 3.0 specification is currently available in three schema languages: SOX, XDR, and DTD [[xCBL](#)]. The SOX schema appears to be most complete and has a structure very similar to the W3C XML Schema. I developed a SOX to UML/XMI transformation used to import this schema. The definition of high-level vocabulary modules (e.g. Trading Partner, Order, Catalog, etc.) is only informally specified using comments in the main xCBL schema file. I used these comments as a basis for creating a Trading Party module that I imported into UML .

The UML diagram shown in [Figure 3](#) illustrates the TradingPartnerOrganizationInformation message from xCBL. For our initial portal design, the TradingPartnerOrganization class can be used to retrieve the organization's addresses, bank detail, language preferences, and identifiers that we hope to match up with the Party ID used by FpML.

**Figure 3. Subset of the xCBL 3.0 Trading Partner vocabulary**



As a test of this model's completeness and accuracy, I generated a W3C XML Schema from this Trading Party definition in UML and used it to successfully validate the test document instances that were distributed along with the xCBL SOX schemas. This schema generation (i.e. forward engineering) used the hidden stereotypes and properties that were created by the reverse engineering transformation from SOX; for example to specify model group (sequence or choice), element position within a sequence, etc. This experiment provides convincing support for the claim that UML can be used to create XML schema vocabulary definitions that are independent of implementation language.

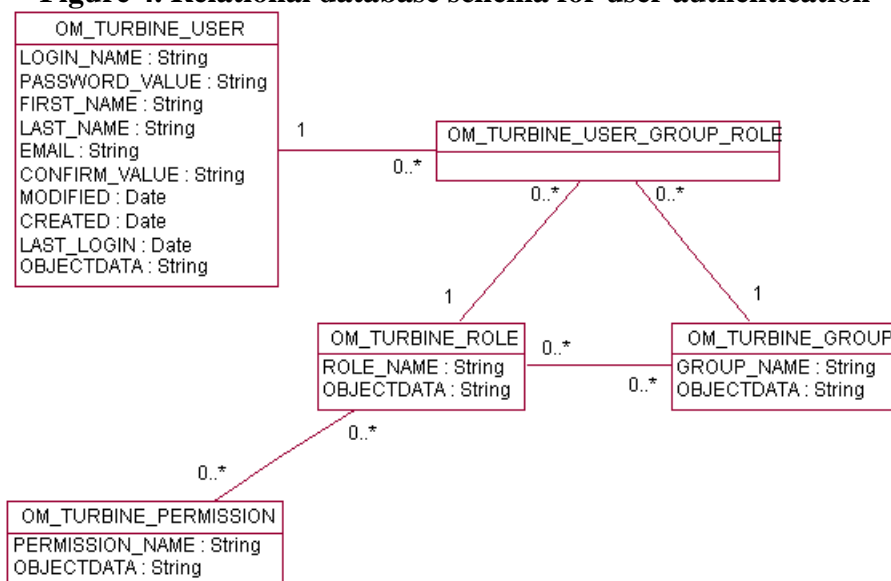
### 3.3. Portal Profile Database Schema

All portal management systems include a capability to authenticate user login and determine a user's permissions within the portal. Many portals also store user preferences for portal content, layout, and presentation. These profile data may be stored in files, a database, or a company-

wide LDAP directory. But regardless of the underlying implementation, there must be a model of the portal profile content and an interface for its use by portlets within the system. The example included here is very simple but it fulfills most of these objectives as part of an open-source portal server [Jetspeed]. Our portal at XMLmodeling.com <http://xmlmodeling.com> is implemented with this profile design.

The Turbine project [Turbine] within the open-source Apache Jakarta family includes a relational database design for user authentication. The Oracle SQL script from this project was imported into into a UML model using the reverse engineering mapping provided by Rational Rose. The resulting class diagram is shown in Figure 4.

**Figure 4. Relational database schema for user authentication**



This model is a direct representation of the relational database table structure implemented in Turbine, but it also provides an easily read view of the conceptual model for user authentication. From this diagram, a business analyst could evaluate and critique the applicability of this design in our portal application. If additional extensions are required, they could be added to this diagram and used to regenerate an updated database schema. Or the same conceptual model could be used to generate an XML schema for this information structure.

This database schema is very simple when compared to the XML vocabulary models. However it illustrates the approach and potential for integrating other relational database assets within a large heterogeneous system. For example, a complete design of this portal would require access to proprietary algorithms and historical data used to support pricing and risk management. These might be integrated as part of the trade capture portlet or as independent portlets within this overall financial derivatives portal application.

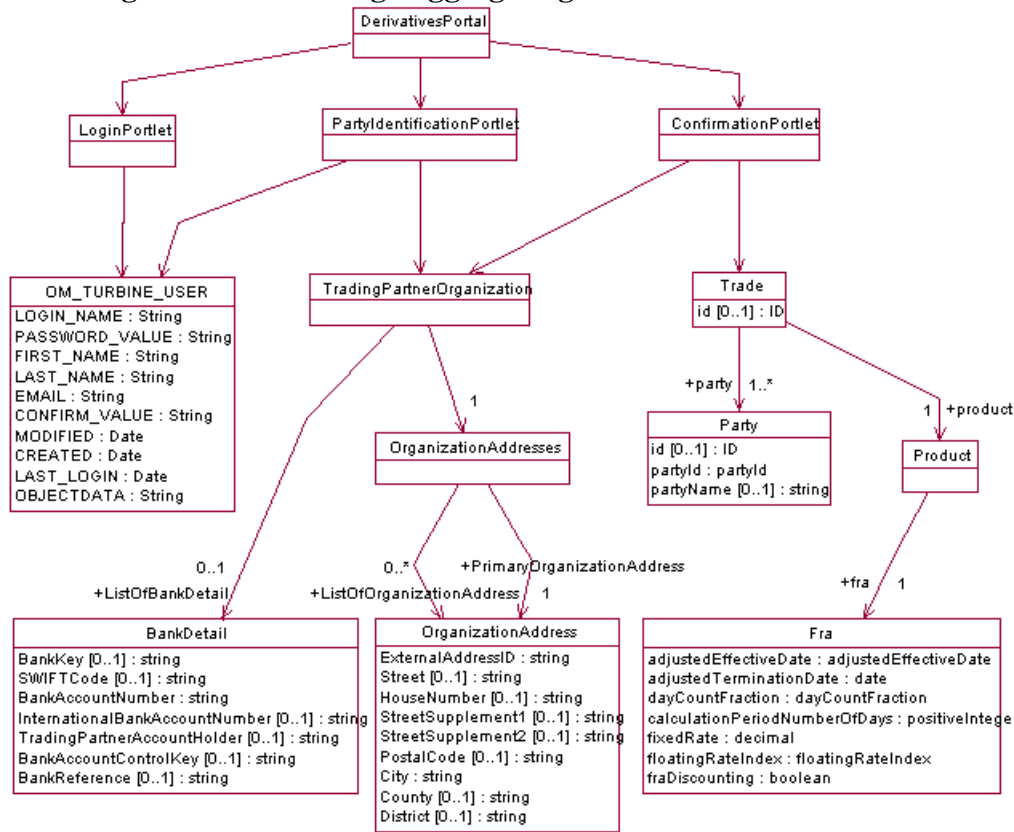
## 4. Refinement of Portal Integration Design

The concept of a "view" is very important in any modeling activity, with or without UML. Any model of a realistic system includes a lot of detail, but you rarely want to see all of it at one time. Different analysis activities with respect to the model require different views or subsets that are relevant in each context. The diagrams of FpML and xCBL vocabularies shown in previous sections were views showing subsets of those models. Returning to the initial task of designing a portal for financial derivatives trading, we'll construct a refined view of the portal analysis model that includes a small subset of the content from each XML vocabulary and database schema.

It's important to understand that a UML model is much more than the set of diagrams. The UML modeling language is defined by a standardized metamodel and diagrams are simply views of this model specification. When we imported the three schemas into UML, we actually created a rich model instance composed of several packages that divide this model into logical modules and namespaces. By default, we created a class diagram view for each package.

Every UML tool has a tree navigation view for displaying and reorganizing these packages and their contents. When I built this final design model using the Rational Rose UML tool, it was a simple task to drag & drop the required classes from the navigator tree into the new diagram. Most UML tools will automatically draw the model's associations to other related classes that are already on the diagram. So by picking classes from three different imported models and creating new links to the portlet classes, we get a quick integrated design model that is based on industry standard vocabularies for FpML and xCBL. The portal design class diagram is shown in [Figure 5](#)

**Figure 5. Portal design aggregating three information models**



## 5. Conclusions

Although considerable work remains before these schema mapping and modeling techniques are ready for company-wide deployment, significant benefits can be realized by beginning to adopt these approaches in current system integration projects. By establishing common system development processes that guide integration of XML data with other legacy sources, we can break down the walls and silos that often separate analysis and design using different implementation technologies.

For system integration, the schemas from XML and relational databases represent only part of a successful system design. I intentionally limited the diagram detail associated with the portal and portlet classes, but they could be expanded to more fully represent the control and presentation of data to the portal's users. The algorithms for pricing, risk analysis, credit approval, etc. should be included in this design, probably in their own packages within the UML model. All of these portal interaction and algorithmic classes could be deployed using either Java J2EE or .Net components, and both of these component-based architectures are supported by several UML tools.

Choosing a model-driven approach to system integration does not force you into a drawn-out waterfall development process. The approach described in this article illustrates an evolutionary and incremental process for system analysis and design. You may choose to terminate use of UML at this point and transition to coding in Java or other implementation languages. Or you

might continue to refine this design and use the UML tool's ability to generate many of the routine classes necessary to support EJB deployment. If you have used UML to create the original XML schema design, then those models would be used instead of reverse engineering existing schemas. Over time, a development group will accumulate a repository of model components that can be quickly combined to analyze new systems similar to the portal example described in this article.

## Bibliography

[Carlson] David Carlson, *Modeling XML Applications with UML: Practical e-Business Applications*, Addison-Wesley, 2001. This book contains an introduction to XMI and the specification of a UML profile for XML Schema. This is all described a part of a realistic e-business application analysis and design for product catalogs.

[Carlson-b] Modeling XML Vocabularies with UML, Parts I, II, and III. See this article series on XML.com, Sept-Oct 2001, or see <http://XMLmodeling.com> for the articles archived in PDF.

[UML] The Object Management Group maintains a site introducing the Unified Modeling Language (UML) at <http://www.uml.org>

[XMI] The XML Metadata Interchange (XMI) version 2.0 specification is available at <http://cgi.omg.org/cgi-bin/doc?ad/01-06-12> or see [Carlson] Chapter 6 and Appendix B for an overview.

[Naiburg] Eric J. Naiburg and Robert A. Maksimchuk, *UML for Database Design*, Addison-Wesley, 1001. This book describes a UML profile for relational database design.

[FpML] The FpML specifications and related presentations are available at <http://www.fpml.org>

[xCBL] The xCBL 3.0 specification is available at <http://www.xcbl.org>

[UBL] The purpose of the Universal Business Language (UBL) TC is to quickly develop a standard XML business library by taking an already existing library as a starting point and modifying it to incorporate the best features of other existing XML business libraries. See <http://oasis-open.org/committees/ubl>

[Turbine] Turbine is an open-source Java development framework for secure web applications that includes user authentication capabilities. See <http://jakarta.apache.org/turbine>

[Jetspeed] Apache Jetspeed is an open-source portal server written in Java, and which uses Turbine for user authentication. See <http://jakarta.apache.org/jetspeed>

## Glossary

FpML	Financial Products Markup Language
UBL	Universal Business Language

UML	Unified Modeling Language
xCBL	XML Common Business Library
XMI	XML Metadata Interchange

## Biography

David **Carlson**

CTO

Ontogenics Corp.

Boulder

U.S.A.

Email: [dcarlson@ontogenics.com](mailto:dcarlson@ontogenics.com)

Dave Carlson, the CTO of Ontogenics Corporation, is an experienced researcher, developer, author, instructor, and e-business consultant. He is a frequent contributor to several technical journals, and has been a speaker at recent OOPSLA and XML conferences. In addition, he has taught graduate level courses in knowledge-based systems and computer architecture, and was a Sun-certified instructor for Java 1.0. During his 20-year career, Dave's focus has been on creating practical applications of leading edge technology, including application modeling and XML vocabularies. He earned his Ph.D. in Information Systems from the University of Arizona and is the author of a book titled "Modeling XML Applications with UML: Practical e-Business Applications," published by Addison-Wesley in April 2001.