

# Leveraging Data Models to Create a Unified Business Vocabulary for Service Oriented Architectures

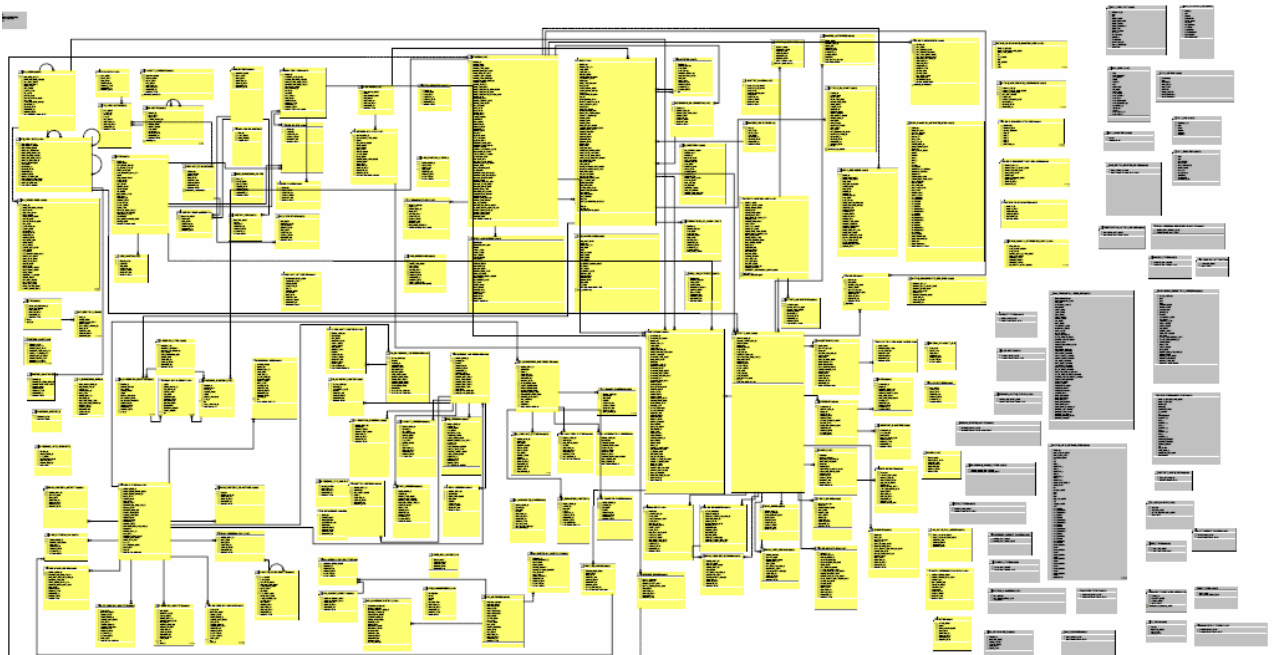
By  
Steven Lemmo  
CTO & Founder  
[ObjectRiver Inc.](#)

## Introduction

Early SOA initiatives generally focused on defining business services first, and worrying about the data later. Today, we acknowledge that services and data must work in concert to produce coherent and well-aligned service architectures. The unifying elements in the architecture include a shared definition of business information structure, validity and vocabulary – in other words, the metadata.

This article describes a best practice that shows how to leverage data architecture to create a vocabulary for Web applications. The practice works to ensure that industry standard terminology and business entity definitions are preserved through the layers of an SOA. Moreover, it is flexible enough to accommodate evolving business requirements.

## SOA This!



**Figure 1. Data Model ER diagram**

You would never see the above figure at an SOA conference (at full size, it covers a wall). These conferences talk about building SOA from the business side top-down approach. They typically never speak about the data. *So when does the miracle happen?*

By working from the bottom-up, we discover the business vocabulary contained in the data model and provide it to SOA developers who are working top-down from the business side.

## Data models contain the business vocabulary

Complex data models can be reduced down from potentially hundreds of tables into a manageable number of components. The components define a vocabulary in terms that are understood by business users. We find the elements of the business vocabulary in three areas.

- **Domains**

Domains define the types of all fields within the database schema. Typical domain definitions include **Indicator**, **Currency**, **Quantity**, etc. Domain types are very useful for establishing strongly typed application variables, thus taking some of the mystery out of propagation.

- **Code Tables and Validation Rules**

Code tables are an age old design pattern used by data architects. They are tables that hold the valid values (constraints) for fields within the database schema. A typical example of a code table would be a **PhoneCode** that contains the values **Home**, **Work**, **Fax**, **Cell**, and **Pager**. These values are not static; for instance, a new **PhoneCode** could be added called **Skype**. The constraints are validated through the use of foreign keys.

When exposed up to the business services layer, code tables are extremely valuable in SOAs. Web applications are detached from the database, meaning that constraints enforced at the database level need to be enforced in a remote environment as well.

If the code tables are encoded into XML schema primitive types, a Web application can validate a code on its own, thus minimizing error checking, exception processing and network requests.

- **Coarse Grained Business Objects**

Business objects are persistent representations of the entities of interest to a business. When well designed, these should reveal the nouns that are spoken by business users.

The first place to look for business objects is within the natural hierarchies of the relational tables in a normalized data model. Using primary keys to determine the hierarchies, you can typically find about 15-25 business objects within any given data model. We call these "coarse grained" because they are aggregations of entities that are linked through defined relationships.

The following table is an example of the business objects found in data models from three vastly different industries.

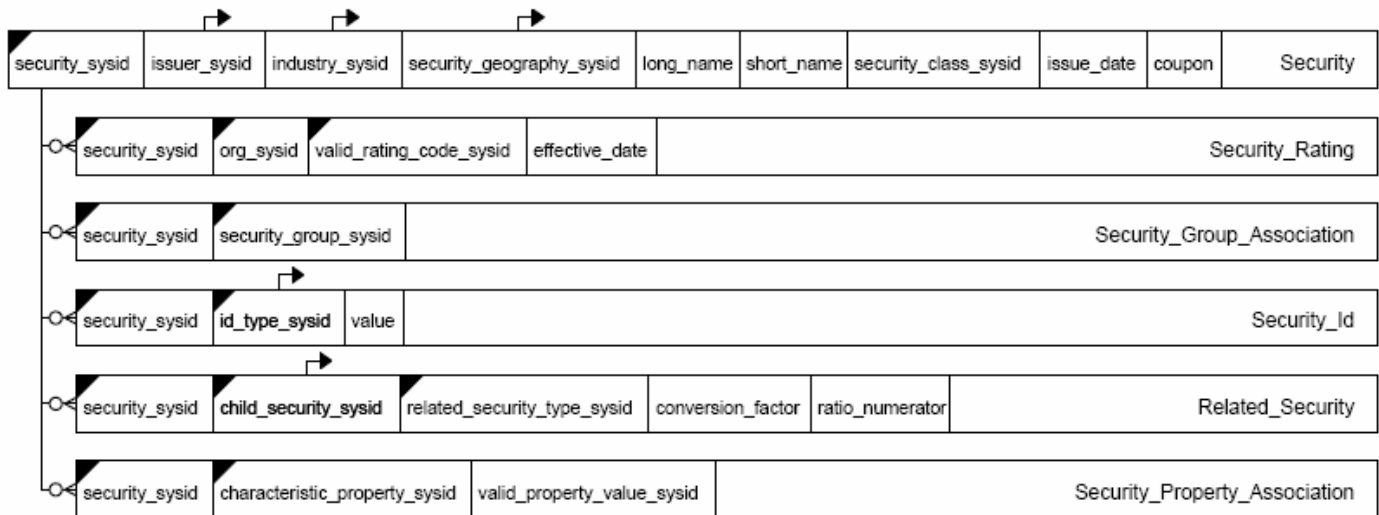
Retail	Financial Security Master	Insurance
Contact	Security	Person
Location	Security_Type	Organization
ContractRelation	Exchange	Agreement
RetailSale	Industry	Policy
Contract	Issuer	Claim
Order	IssuerGroup	Location
Product	SecurityGeography	Contact
Inventory	RatingCodeAssociation	Agent
	Account	
	Institution	
	Depository	

**Figure 2. Business Objects found within industry data models.**

The business objects discovered in these data models correspond to the nouns in the business vocabulary. If you were to eavesdrop on analysts from one of these industries, you would hear these nouns in their business conversations.

Once we have broken the data model down, we examine the structure of each business object and derive a data dictionary such as that shown in the financial security diagram below.

## Security Diagram



**Figure 3. Data dictionary diagram of Financial Security business object.**

Each business object is represented as a hierarchy, with each row corresponding to a relational table. The triangles are the primary keys, and the periscopes represent foreign keys to other business objects.

This practice effectively creates an object model within the data model, and captures the vocabulary in a data dictionary. Data dictionaries are readily consumable by Business Analysts, Business Rules Writers, Report Writers, and Application Programmers. A data dictionary that represents both the relational tables and the object model helps all the stakeholders to communicate more effectively.

## Metadata is incomplete for Web development

This practice relies on the data model as the originating source of metadata. Having worked with many industry data models, however, it is clear to me that while most metadata is suitable for use in reporting and business intelligence, it is incomplete for building Web applications.

Below is a list of inaccuracies and omissions we often find in metadata. A little analysis reveals that these problems can be addressed with User Defined Properties, domains, and logical constructs in the data model, without changing the physical database model.

- Sequences are routinely used as primary keys. The real primary key columns are present, but they are not marked as primary keys. This makes it difficult to find the natural hierarchies within the data model.
- Missing foreign key constraints confound attempts to discover relationships between business objects.
- Many models lack domain types that define database column types. When present, domain types are propagated to the Web services level to implement strong typing.
- When code table foreign key constraints are missing, codes need to be identified by their field names. With standard database applications this has not been a big issue, because these applications let the database validate codes or read the code tables directly. The same goes for BI and data warehousing systems, which also leverage code table definitions.

Web applications are different because they are detached from the database. Web data is passed around in XML documents that are validated by their own metadata, the XML schema. In this case, code table values should be included in the metadata.

All of the above problems can be found and corrected in the data model, without affecting the physical model.

## UML in reverse

If two things are unified they are strongly working with each other; they are in unison. The Unified Modeling Language unifies the data model into an object model. With the two parts combined, object based Web applications can be generated.

The best practice discussed in this paper does the reverse of UML. It effectively unifies the object model into the data model. This leverages the existing data model to define an object model, data types and constraints for Web data access. These components comprise the business data domain for processes and services.

## Model Driven Architecture

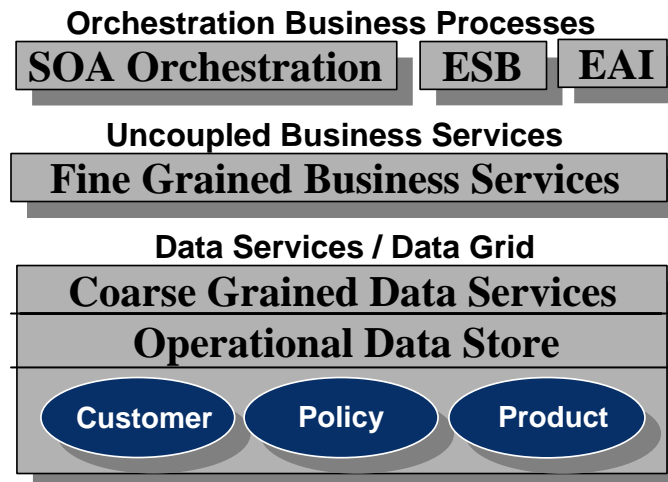
Model driven architecture is a practice designed to generate a system from a data model, change the model, and regenerate the system. Data architects have been using model driven architecture for at least the last fifteen years (the age of the ERwin data modeling tool), so this method is a logical extension of their data modeling process.

With the data and object models combined in a model-driven system, architects can continue to enhance both models in response to new business requirements

## Service Oriented Architecture

The next step in our best practice is to extend the unified model with its vocabulary and underlying data architecture into data services. This is a layered approach which promotes uncoupling, re-usability and adaptability for the agile business.

Services are language and operating system independent procedures that are assembled to create Web applications. The following diagram shows the mapping between the modeled business vocabulary and a simplified SOA stack.



**Figure 4. Service Oriented Architecture software implementation stack.**

### Data Services Layer

The data services layer exposes the business objects discovered in the data schema, and it defines the master set of codes and domains that will percolate up to the business orchestration layer.

The business objects are coarse grained and tightly coupled to the database, and contain all the attributes needed to run the business. These objects are versioned by the database to allow addition of new attributes and tables without disrupting running services. Although they are too large for direct consumption by business services, coarse grained data objects provide a basis for SOA developers to build their own representations within the context of their business goals.

The data services layer insulates the database from “free-for-all” data access. A free-for-all is where each application group writes their own data access code. They may use SQL, Stored Procedures, Hibernate, or TopLink. This kind of application development quickly leads to freezing the database schema due to the dependencies introduced by multiple data access methods. A data architect can not predict what data access dependencies will be affected by any proposed schema change.

A coarse grained data access layer that is generated from a data model will create consistent, documented data access for application architects. Application groups all access the same data services layer to implement their business services. No matter

now many applications are accessing the data services layer, it appears as one application to the database. This frees the data architects to continually make changes to the data model, because they are fully aware of the data access dependencies.

### **Uncoupled Business Services Layer**

The business services layer exposes its own business objects. The object definitions are more generic than those in the data services layer, and are uncoupled from the database implementation. In our best practice, these business objects incorporate the domains and codes from the data services layer into their field type definitions, and they are populated from the data services coarse grained objects.

Business services are fine grained, and are typically associated with business objects. Collections of services may reference and operate on specific business objects. For example, one set of services interacts with customer objects, another set interacts with product objects, and so on.

The goal of the business services layer is to define loosely coupled services that are reused across a variety of applications. The benchmark for testing whether services are sufficiently loosely coupled is the level to which the services are application agnostic. If they meet this test, much of the battle has been won. This will enable reuse in the SOA orchestration layer.

### **Orchestration / Business Processes Layer**

Once fine grained business services are created or purchased from packaged applications, the orchestration layer is where they are assembled to create coarse grained business services or business process applications.

This layer is the ultimate promise of SOA; however we are still early in its evolution. The goal is to build Business Process Management (BPM) applications from collections of uncoupled business services that are deployed in the enterprise. SOA orchestration actually overlaps with proprietary BPM products that are targeted toward business users. Though ESB and EAI vendors are showing initiative in this area, there is no clear leader with a pure play SOA orchestration product, because of the immaturity of the industry and standards.

One of the challenges to orchestration is the integration of vocabularies from different SOA implementations. It shouldn't be a surprise that a purchased customer hub does not interact with the services of a product hub due to inconsistent vocabulary. This is part of the problem with packaged applications; they have their own vocabulary that may be inconsistent with the business language.

### **Summary**

SOA implementations are developed both top-down and bottom-up, and those developments must meet to arrive at a shared definition of the business vocabulary and its supporting data infrastructure. The unifying elements are found in the data model, which contains the language of the business and is the true source of metadata. A model driven development process leverages the data architecture to create shared data services and a vocabulary for Web applications.