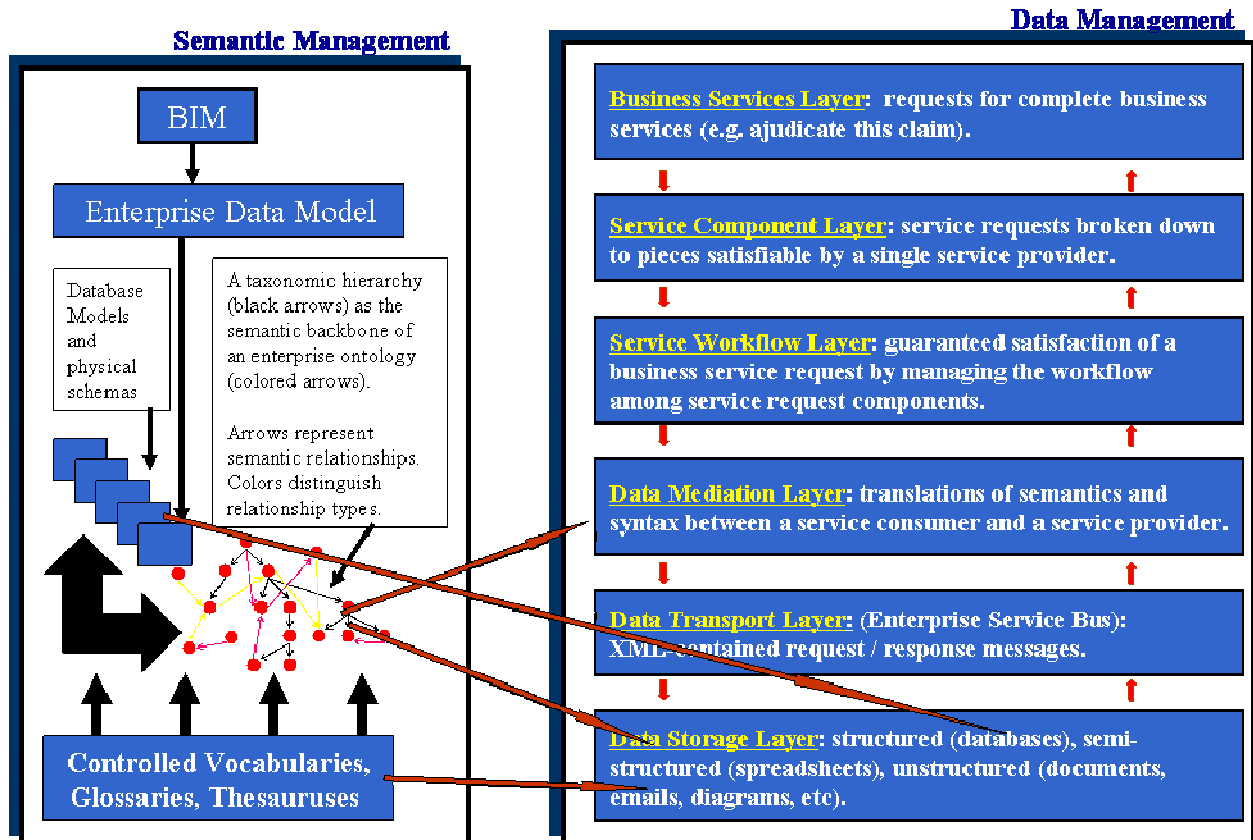


An Ontologically-Informed SOA Architecture.

Tom Johnston
February, 2007.

Enterprise Architecture: a Future State Vision



Service Oriented Architecture (SOA).

In this description of work being done in the context of a service-oriented architecture, the following concepts are used:

1. Consumers and Providers of services are business application systems, or business-function-bounded components of such systems.

2. The Broker and the Mediation Agents are middleware software components that dynamically link consumers of data to providers of data, and that translate messages between them, both syntactically and semantically.
3. Services are for either the provision of or consumption of data (e.g. SQL updates or SQL retrievals), or for transformations data of or derivations based on data (e.g. calculations of payments on claims).
4. The Service Contract is basically a table of service-level agreements for each service, which can be read and interpreted by the middleware broker searching for a provider to satisfy a service request.
5. The Enterprise Service Bus (ESB) is a layer of vendor-provided software which handles all communications among Consumers and Providers as XML-enabled asynchronous messages.

The SOA Messaging Paradigm.

The basic SOA messaging paradigm is as follows:

1. A Consumer requests a Service.
2. The request is based on a Service Contract, which specifies both the Service and its associated Service Level Agreement (SLA).
3. A Broker locates a Provider for that Service, who can meet the Service Contract's SLA.
4. An Enterprise Service Bus (ESB) transports request messages from the Consumer to the Provider, and response messages from the Provider back to the Consumer.
5. But Consumers and Providers don't necessarily "talk the same language". So Mediation Agents sit on top of the messaging layer of the ESB, and translate between Consumers and Providers. XML is the lingua franca, used in all ESB messages. Subject-matter-specific topics are assigned a specific template which is agreed-to by

Consumer and Provider and, usually, by a wider Community of Interest (COI).

6. If a better Provider comes along, the Broker can redirect requests to that Provider. The Consumer knows nothing about the change. There are no adjustments the Consumer has to make.

Three High-Level Architectural Patterns.

1. Point to Point. A requestor application sends a message (a transaction) to another application. It responds back. Hundreds of such messages link dozens of such applications in a point-to-point configuration that resembles the old tangles of “spaghetti code” from the mainframe days.
 - a. The main problem with point-to-point is not scalability, although that does become a problem. The main problem is that every transaction type an application is coded to respond to constitutes a “service contract” that it must honor. With often dozens of such contracts to honor, replacing or just changing any business application is difficult and risky, because the replacement or the changes must leave the service contracts unaffected.
2. Hub and Spoke.
3. Service Bus.

Architectural Pattern for the Messaging Layer: Service Bus.

1. Message bus (ESB) at the transport layer.

Architectural Pattern for the Service Component Layer: Hub and Spoke, Supplemented by Point to Point.

1. Hub and spoke, with the data warehouse complex (ODS, historical DW, marts) as the hub, at the application layer.

A Brief Description of Semantic Management.

Semantic Management components will enable semantic interoperability among business applications, both within the enterprise and among the companies with which the enterprise does business. Mediation agents will consult ontologies to determine how to map the schemas and values used in one database, or by one application system, into schemas and values used in other databases, or by other application systems. For each message between a pair of systems, the mediation agent will specify a source-to-target mapping.

The same paradigm applies to unstructured data. A search for customers in one set of documents may return the names of parties who have responded to marketing contacts, whether or not they have purchased a product or service. A search for customers in a different set of documents may return only names of parties who have made an invoiced purchase in the last five years. When comparing the two lists of names, mediation agents, using an the enterprise ontology, will either resolve the semantic discrepancies, or else make those discrepancies known, as part of the result set.

Terminological Taxonomies.

Semantics are expressed in an ontology, whose “backbone” is a hierarchical taxonomy. Some taxonomies are terminological. Their nodes are terms, and the hierarchical relationships express some of the semantics of those terms. Here is a brief example.

Suppose that "salesperson" is a node under "employee" in the the enterprise terminological taxonomy. This says that “A salesperson is a employee”.

In a terminological taxonomy, the “arcs” between a pair of nodes express the “KIND-OF” relationship. So, more formally, this piece of the taxonomy says that “A salesperson is a KIND OF employee”. In a relational data model, we would say that a Salesperson entity is a subtype of an Employee entity (although relational support for subtyping is very limited).

This taxonomy is a piece of *formal* semantics, and so it can be navigated by software. Consequently, a query language could ask questions like “What types of employees are there?” and get back "salesperson" as one of the

results. The term “inference engine” is used to refer to the data structures and software that make this kind of “automated inferencing” possible.

A “slot” is a piece of additional information about a node in a taxonomy or an ontology. The slot associated with the "salesperson" node in this hierarchy says, let's suppose, that “a salesperson is a employee, working within a designates territory, whose job is to sell products to customers”.

There is a lot of semantics in this slot, but it is *informal* semantics. It is a description of "salesperson" that human beings can read and understand, but to software it is just a string of characters. No automated inferencing is possible with it. No queries can discriminate its contents.

Terminological Ontologies.

Now suppose we have a more complex network of nodes and arcs. Suppose our nodes are now "employee", "salesperson", “product”, “territory” and “customer”, and our arcs are “KIND-OF”, “SELLS”, “WORKS IN” and a few others (each arc linking the obvious pair of nodes). Now we can translate the definition in the slot into a set of nodes and arcs like these.

That translation now makes the full semantics described in the slot *formal*. The full semantics of "salesperson" is now in a form where queries could be written against that terminological network. Automated inferencing is now possible against the full semantics of "salesperson".

The inferencing engine that would support that inferencing, and would present an appropriate query language to the user, is an ontology management tool. The hierarchy is a taxonomy structure. The network is an ontology structure.

With the semantics of "salesperson" formalized in this way, we can write complex queries against PDFs and other documents, against spreadsheets, and even against diagrams (looking for labels with "salesperson" in them).

The Semantics of Structured Data.

But those artifacts are instances of unstructured or semi-structured data. How does all this semantic management get applied to *structured* data as well, i.e. to our company's databases?

On the structured data side, we express semantics in a hierarchical set of data models. At the enterprise, the highest level data model is an enterprise data model (EDM), although in some companies, the EDM exists one level beneath a more generalized model called the Business Information Model (BIM). Under the EDM, there are the subject area or application-specific data models and databases.

What does it mean to say that one data model is “under” another? In practice, it means that the developers of the lower level data models have looked at the higher level models, and maybe even used the same names for entities, attributes and relationships, and that these names appear to mean the same thing in the higher and lower level models. In practice, then, it doesn't mean very much.

In theory, on the other hand, one thing it should mean is that the highest-level entities in a model should be subtypes of the lowest-level entities in the parent (next higher) model. This is a clear-cut criterion. But “under” should have additional constraints if it is to fulfill its intended meaning as “more detailed than but semantically consistent with”.

Taxonomies and ontologies supply those additional constraints.

1. Our taxonomy said that a salesperson is a kind of employee. For the enterprise data models to be semantically consistent with that taxonomy, salespersons should be modeled as a table which is a subtype of a table of employees.
2. Our ontology (although we didn't spell this out) says, let's suppose, that a salesperson is permitted to sell product only to customers who reside in his sales territory. For the EDM to be semantically consistent with that ontology, there should be a relationship from a salesperson entity to a customer entity, a sales territory entity to a salesperson entity, and a salesperson entity to a customer entity. In addition, full semantic consistency of the EDM to the ontology also requires stored

procedure code which constrains the relationships as indicated (e.g. that a salesperson to customer relationship cannot exist unless both are in the same sales territory).

In this way, we can begin to manage the enterprise semantics – to enforce semantic consistency – across structured, semi-structured and unstructured data, in object or relational databases and their data models, in Word and PDF documents, emails, spreadsheets, Powerpoint and Visio diagrams, etc.“

Left-Over Semantics” in Structured Data: Relationship Labels.

So far we have been talking about managing *formal* semantics, semantics expressed in structures that software can access to update and to query. We have been talking about semantics expressed and enforced by inferencing engines (whose implementation, currently, is shared by both data structures and DBMS-enforced constraints, and also developer-written code).

But there will always be semantics “left over” after formalization. Above, we called this “informal” semantics. Informal semantics are expressed in data which human beings can understand, but which are just bits and character strings to software.

For example, when data models are being developed, the labels given to relationships are informal semantics. In a relational data model, most relationships are one-to-many relationships, e.g. a customer may have many employees, an invoice may have many lines, a CSR may respond to many customers, etc.

May have, may file, may respond to, etc. These are all different labels for relationships in the data model. But as far as the inferencing machine we call a RDBMS is concerned, there is just the relationships and their cardinalities (zero or one, one or many). The labels on the relationships aren’t even passed to the DBMS as text strings. In all these cases, all the DBMS understands is that one instance on one side of the relationship can be related to any number of instances on the other side.

So in data models, relationship names are informal semantics. They help human beings understand the model, but they don't help the DBMS maintain and query it.

“Left-Over” Semantics in Structured Data: Data Dictionaries.

As a data model is being constructed, we define (or should define) its components. These are the definitions of entities, attributes and relationships that appear in the data dictionary for the data model.

But clearly these definitions are also informal semantics. The DBMS doesn't know anything about them. Therefore, we can be as careful or careless as we like in writing these definitions, and it won't affect the database itself, or the queries we write against it.

Managing Left-Over Semantics.

How can we make these definitions, and these relationship labels, as clear, accurate and semantically consistent as possible, if we have no formal semantic ways of doing it?

We start with a *glossary* of business terms used in these labels and definitions. The glossary lists and defines these terms. So if a business term is used in an entity definition, for example, it should already exist in the glossary, and it will be assumed to have the meaning given to it in the glossary.

While this repository of basic terms is called a glossary on the structured data side of the house, it is called a “controlled vocabulary” on the unstructured data side of the house. Regardless of what it is called, there should be only one such repository for the enterprise, one set of basic terms and their definitions. This is the most important tool we have for managing informal semantics.

Another tool is the *thesaurus*. Like an ordinary thesaurus, its purpose is to suggest terms that are close in meaning to a given term. So a thesaurus is a *heuristic* tool. It is not a glossary or a dictionary. It is not a taxonomy or an

ontology. It is an informal network of “similar in meaning” relationships between the terms in the glossary.

Definitions appear in the data dictionaries for data models. They also appear in slots for nodes in taxonomies and ontologies. *Semantic clarity* is enhanced with the rule that in any definition, the business terms used must come from the corporate glossary either directly, or indirectly by being terms that themselves have this kind of controlled definition. This prevents someone from defining a term in a way that is meaningful to him, but uses terms not understood by others.

This same restriction also enhances *semantic consistency*. Although we cannot guarantee it in any automated fashion, our rule is that all business terms used anywhere in the enterprise can be traced back to the terms in a glossary / controlled vocabulary, and mean what they are said to mean in those semantic repositories.

As shown in the diagram at the beginning of this essay, semantic management cuts across all the layers of an enterprise architecture. Wherever the terms of a controlled vocabulary, or of terms defined on the basis of a controlled vocabulary, appear, whether the semantics of these terms do or do not also appear in the formal structures of data models, taxonomies or ontologies, they must be used consistently with the stated enterprise defined semantics.

In this way, the enterprise can continually improve the clarity and consistency of what we say