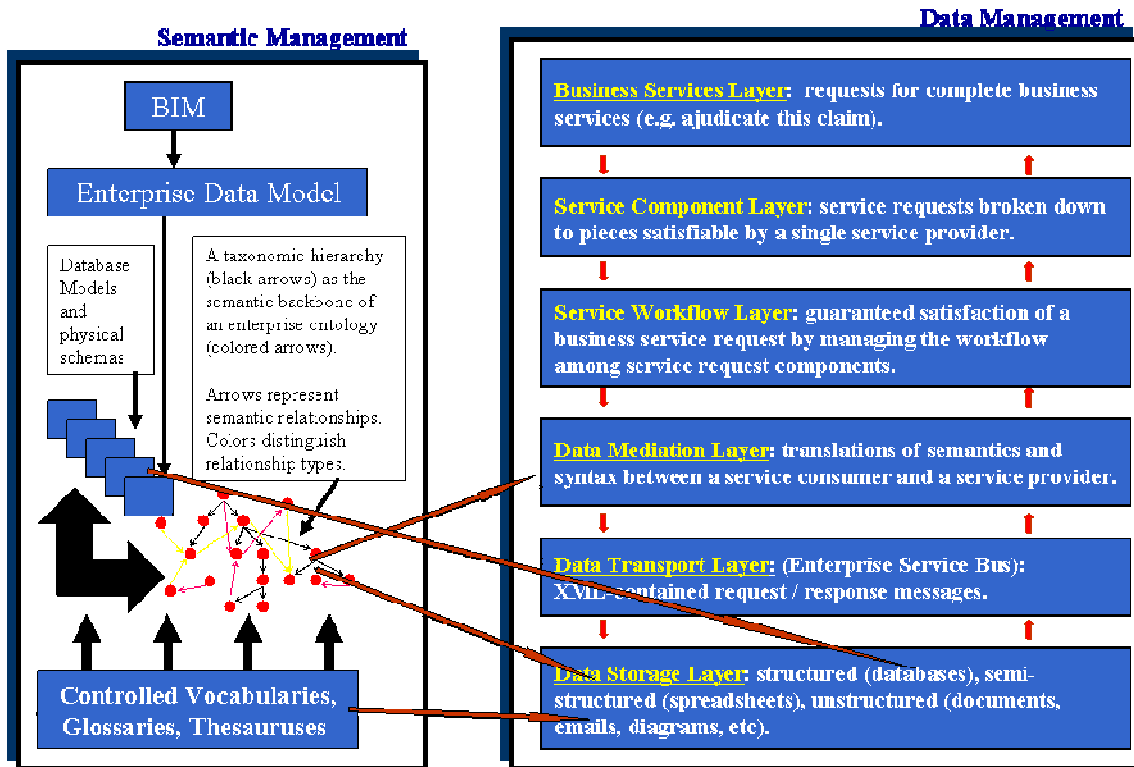


How to Develop and Evolve to an Enterprise IT Architectue at Both the Physical and Semantic Levels: a Proposal.

Dr. Tom Johnston
MindfulData.com
June, 2008.

How to Develop and Evolve to an Enterprise IT Architectue at Both the Physical and Semantic Levels: a Proposal.	1
Service Oriented Architecture (SOA).....	2
Components.	2
Process.	3
A Layered Data Management Architecture.	4
Business Services Layer.	4
Service Component Layer.....	5
Service Workflow Layer.....	5
Data Mediation Layer.	6
Data Transport Layer.	6
Data Storage Layer.	6
Recommendations.....	7
Conduct a Business Process Re-Engineering Study.	7
Integrate the Semantics of the Business.....	7
Buy, Don't Build, the Middleware Layers.....	8



Service Oriented Architecture (SOA).

Components.

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 to providers, 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 of data 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.

Process.

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. Consumer and Provider are loosely-coupled, which is the foundational principle that supports inter- and intra-systems flexibility.

A Layered Data Management Architecture.

The illustration on page 1 shows a six-layered data management architecture.

Business Services Layer.

If business services are (as they should be) large-grained run-the-business functions, then most companies don't know what their business services are. Work gets done; transactions get generated. Money flows in; less money (hopefully) flows out. We all get paid, and life moves on.

That is not a set of business services. It's a bunch of transaction-acquiring or transaction-generating systems, supplemented by some report-generating and query-responding databases, exchanging data with one another as needed, resulting in a patchwork quilt that almost defies logic.

To create a business services layer, a business process re-engineering study should be carried out. The objective is to identify large scale business functions based around the core objects with which the business interacts in the process of doing business. These functions include supplier relationship management, HR (employee relationship management), customer relationship management, regulatory agency relationship management, partner relationship management, etc. The next step is to break these high-level functional units down one or two more levels. At that point, business functions start to become service components.

At this point, top-down decomposition into logical categories gives way to bottom-up groupings from platform and software implementations of business functionality. An as-is business services layer results from bottom-up groupings based on existing platforms and systems. A to-be business services layer results from bottom-up groupings based on an enterprise IT architecture which describes the redistribution of database and application level functionality into a more rational set of groupings, one which reduces both the duplication of functionality and also the point-to-point interconnectedness of the implementations of those functionalities. A prioritized list of the deltas constitutes a roadmap for the evolution of the information processing resource of the enterprise.

Note: a great deal of policy and procedural infrastructure can be created around this simple description, but it is nothing more than a support structure. Describe where you are; describe where you're going; then figure out how to most efficiently and effectively get from here to there. Do so in terms of a map whose topography is defined by a hierarchy of business functionality defined by the business itself. I would personally time-box the development of a business service layer and a high-level as-is and to-be IT architecture, to no more than three months of work by a team of two or three people. The result doesn't have to be

perfect, or even fine-grained. It just has to be good enough to get us started on the right track, and keep us on that track until the next iteration of this process refines our map and our travel plans.

Service Component Layer.

Service components are business services in small enough “chunks” that a request for service from one service-consuming component can be satisfied by one other service-providing component.

The point of breaking business services into service components is to get down to the level of transactions which involve a single consumer and a single provider. These “atomic level” transactions can then be managed by a service “broker” who can dynamically route a request transaction to the provider that can best respond to it.

In other words, the point of breaking business services into service components is to make SOA possible!

Service Workflow Layer.

It follows that an upper-level business service is made up of lower-level services and, ultimately, of service components. But just as reliable telecommunication transmission protocols need a layer in the protocol stack that will re-send or re-request a block of data whenever needed, this application architecture needs a workflow management layer which will do the same thing.

There is a standard state-of-the-art pattern for workflow management, and it is not the state transition table pattern familiar to many developers. It is, rather, a pattern known as typed Petri Nets. The principal advantage of typed Petri Nets over state transition tables is that they can manage multiple concurrently executing asynchronous processes, including call-and-wait or call-and-continue coordination among those processes, where necessary. Thus, no implementation of this layer should be considered by a business unless it is based on typed Petri Nets.

The management routines in this layer are aware not just of specific requests made by service components, but also of the highest-level requests made by top-level business services. In that way, they have the knowledge of all service components involved in a highest-level request, and the knowledge to manage a workflow among all components for that request which will result in the satisfaction of the request made by the highest-level requesting business service.

Data Mediation Layer.

Service consumers and service providers often don't speak the same language. Usually, there are differences in both syntax and semantics. Differences in syntax include different data types or lengths for the same piece of information (e.g. policy type), different sets of values (meaning the same thing), etc. Differences in semantics include differing criteria for creating an instance of a concept (e.g. a policy), or different sets of values meaning different things, etc. At a more basic level, platform differences must also be resolved, e.g. EBCDIC to ASCII, OSx DB2 varchars vs. AIX DB2 varchars, etc.

Data mediation agents provide this translation, using either the consumer, the provider or a third "neutral" format for the messages passed between consumer and provider. Usually, XML provides the basic framework for the messages, with the specific semantics agreed to by the local requestor/provider community (i.e. the business itself) or by a wider Community of Interest (e.g. various EDI standards, the ACORD insurance standard, etc.)

Data Transport Layer.

The concern of the data transport layer is to guarantee delivery of messages from one point to another. This is not a point-to-point architecture, however. It is a bus architecture. The difference is that a bus architecture is a *physical* pattern that dynamically supports point-to-point connectivity at the *logical* level. That is, via a physical bus, with mediation agents, the transport layer makes it seem, to every requestor and provider, that they are directly connected.

ESB (Enterprise Service Bus) implementations of a data transport layer are replacing EAI (Enterprise Application Integration) implementations. One reason to prefer ESB is that it is based on open standards, while EAI tends to be proprietary to its vendor. Another is that the implementation of ESB tends to be distributed, while EAI implementations tend to be centralized hub and spoke models. This means that ESB transports will be more scalable and fault-tolerant than EAI transports.

Data Storage Layer.

IT traditionally manages structured data, i.e. data in databases (or VSAM files, etc.). But important shared concepts exist across structured and unstructured data. The concept of a purchase order, for example, will appear as one or more tables in one or more databases, but also as a term used in emails, documents, Powerpoint presentations, etc. All of these different formats in which we talk about purchase orders, unless they are relegated to an ephemeral existence (an email read and immediately deleted, for example) are components of the data storage layer.

From a *semantic management point of view*, there must be integrated management across all of them. From a *data management point of view*, traditional IT management tools and techniques will continue to apply to structured data.

Recommendations.

Conduct a Business Process Re-Engineering Study.

This study will create a two- or three-layer hierarchical structure of business services, and a business-user-level workflow among those services. Each service is a cohesive, single logical unit of work from a business perspective.

1. Some services are provider services only. They interact only with consumer services.
2. Other services are consumer services. They take requests from business users, parcel out the work to lower level services and, ultimately to service components. Consumer services are thus the interface points between business users and the IT processes that support them.

Integrate the Semantics of the Business.

1. Do not create ontologies and taxonomies in isolation, as IT projects. In particular, do not engage in a series of projects that will create an ontology and taxonomy for unstructured and semi-structured data, and then, later on (if ever) a different one for structured data (the data already contained in databases).
2. Develop semantics top-down and bottom-up concurrently.
 - a. Top-down, restructure the business' enterprise data model to incorporate an upper ontology, such as the SUMO (standard upper merged ontology) created by the IEEE1600.1, and use its links with WordNet to construct and/or validate any taxonomic hierarchies or hierarchy fragments created by specific IT projects.
 - b. Bottom-up, continue IT project-level development of structured vocabularies, taxonomies and ontologies. Review all IT logical and physical data models, and their associated data dictionaries and glossaries. Standardize the formats in which their semantics are expressed, and resolve semantic inconsistencies across them.
 - c. Continually compare and integrate results produced by top-down and bottom-up semantic management work.

- d. Iterate. There is no end state. There is just a process of things getting better.

Buy, Don't Build, the Middleware Layers.

1. Proposed must define its own objects for the upper two layers of the architecture. It must also define its own structured and unstructured data artifacts (the bottom layer).
2. But the workflow, data mediation and transport layers are middleware functions, and should be purchased, not developed. The greatest risk in going forward will be the ability to integrate Proposed formal semantic artifacts (specific tool-based vocabularies, taxonomies and ontologies) with the Data Mediation Layer.
3. This is a critical risk that must be managed. If the mediation layer cannot talk to the formal semantic management tools, then the rules for mapping data from one service to another will have to be entered into the mediation layer COTS product. For batch processing into an ODS, data warehouse or data marts, that mediation layer tool is known as an ETL (Extract, Transform and Load) tool. Examples are Informatica and AbInitio.
 - a. But these rules are based entirely on the formal semantic descriptions of the two components between which the mapping will take place, and that formal semantic description is what our taxonomy and ontology management tools are responsible for.
 - b. Therefore, the semantic descriptions of those two components would be *duplicated*, once in the formal semantics management tools, and once in the mediation layer COTS product. This duplication of complex semantic structures would inevitably create mis-alignments, and that would wreak havoc with the attempt to insure *semantic consistency* and *semantic interoperability* across all communicating parties within and without Proposed.