The role of semantic models in smarter industrial operations

Tim Hanis  
Dave Noller  
March 30, 2012  
(First published October 11, 2011)

Solution architects and developers have many architectural options at their disposal. Architects may leverage architectural patterns based on data-oriented design, service-oriented design, message-oriented design, and others. Naturally, these are not mutually exclusive patterns, and we often use components together in solution design. In this article, we discuss semantic model architectures and describe the semantic model approach and how it fits in context of other architectural patterns. We highlight the value of semantic models as a core component in solution design and show how IBM Integrated Information Core enables creation of model-integrated solutions.

Introduction

We often describe three critical elements in the discussion around smarter planet solutions. The three "i"s, as they are sometimes labeled, are "instrumented", "intelligent" and "interconnected". These support the idea that there is much data in the world that we can collect and use to derive intelligence, and from that we can help drive optimization around critical business tasks. What is important in this approach is the analysis and understanding of data from a wide variety of sources in a wide range of formats and context.

Solutions need to be designed to handle this kind of disparate data, including structured and unstructured data, sensor data (current value and historical), images, audio, and video. Not only does this data not fit well into standard relational persistence structures, it also presents a challenge to make sense of it in context.

Consider a smarter city traffic application. Traffic light sensors, speed sensors from the transportation department, and video cameras provide real time traffic data. Additional data that is critical to accurate traffic flow predicting can come from a wide variety of sources including weather reports, accident reports, transit interruptions, calendar events like holidays, seasonal trends like beach traffic, special events like parade, festivals or major sporting events, emergency dispatch events, and significant news events. We need to understand all this data in context and understand the relationships between the events.

In addition, we need a common understanding of the events that we can reference from these various sources. For example, basic terms such as vehicle may become ambiguous between data...
source providers as we consider distinctions like cars, light trucks, semi’s, buses, or motorcycles. Some characteristics like axles or occupants may take on important distinctions. And, of course, the relevant data that we may need to gather is continually changing.

Semantic modeling can help define the data and the relationships between these entities. An information model provides the ability to abstract different kinds of data and provides an understanding of how the data elements relate. A semantic model is a type of information model that supports the modeling of entities and their relationships. The total set of entities in our semantic model comprise the taxonomy of classes we use in our model to represent the real world. Together these ideas are represented by an ontology - the vocabulary of the semantic model that provides the basis on which user-defined model queries are formed. The model supports the representation of entities and their relationships and can support the constraints on those relationships and entities. This provides the semantic makeup of the information model.

From our example, a semantic model could help us understand relationships such as traffic light sensors with the intersections they monitor, any given traffic light sensor with other sensors on the same road, or the relationship of the roads for which we have specific sensor data to other intersecting roads and collectively as feeders to major highways. The model might also yield similar information about bus lines or subway lines. It could describe the types of services available with the locations serviced. The relationships between stations and street addresses, and service lines and surface road routes, would provide the basis for understanding the implications of specific disruptions in mass transit service on road traffic.

As an additional complication, it's possible that a single application will need to interact with multiple domain models (or domain ontologies). One method to achieve that is to merge existing ontologies into a new ontology. It is not necessary to merge all the information from each of the original ontologies since that integration may not be able to be satisfied logically. In addition, the new ontology may introduce new terms and relationships that serve to link related items from the source ontologies. We look more closely at how semantic models fit in an example later in this article.

The understanding provided through semantic models is critical to being able to properly drive the correct insights from the monitored instrumentation that ultimately can lead to optimized business processes or, in this case, city services. As a result, semantic models can greatly enhance the usefulness of the information obtained through operations integration solutions.
Towards semantic model-based operations integration

Over the years, a number of architectural approaches have been defined for the integration of systems and for the representation of the information and processes. These included data-oriented, message-oriented, service-oriented and information-oriented approaches. We want to explore how these various approaches differ and relate, where semantic models fit from an architecture perspective, and what value do they provide as a key component of operations integration architecture. In the next section, let’s look at how architectural approaches have evolved and positioning semantic model integration relative to those various approaches.

Centralized ownership of data

**Centralized application owns data**

In this case, a centralized application owns the data and other applications make direct calls to obtain information or to request that the called application perform some action. Historically, this involved direct invocation of another application through application program interfaces (APIs) or remote function calls (RFCs) that were contained in a client library to which the calling application would be linked. The calling application, in this case, is responsible for understanding the semantics of the called application and it is responsible for all data transformations. Although fast, from a performance perspective, this approach proved costly from a maintenance perspective (and brittle since a failure in one application has a ripple effect through all applications directly connected to each other).

One example of this type of information sharing would be a client application directly invoking a Service Advertising Protocol (SAP) Business Application Programming Interface (BAPI) through a remote function call.

**Data-centered architecture**

An alternative to the centralized application owning the data is the data-centered architecture, arguably a step forward from the direct connectivity approach in that applications are not directly connecting to each other to exchange information. Data-centered architecture is anchored through the definition of relevant business data around which systems are integrated and applications...
are developed. Put simply, data-centered architectures establish a common data model for a centralized data store and client applications interoperate through this centralized data store.

One early example of such an approach, again, was SAP's enterprise resource planning (ERP) application. Anyone who worked with SAP, in earlier incarnations, realized that it was (or at least seemed to be) basically a suite of applications that had been developed to interact around a central data model. Although SAP supported other integration approaches for external applications, SAP itself had taken a data centered approach to intra-application communication.

This approach to integration, although inherently simple, still results in a tightly coupled system where all system components are affected by changes in the data model (and have a single point of failure in the shared data store).

**Distributed ownership of data**

**Messaging oriented architectures**

A message-oriented architecture will typically rely on two complementary components:

- An interaction model that defines patterns such as commands, request/reply, or pub/sub
- A data model of the content to be exchanged

Both of these could leverage industry standards. Java™ Messaging Service (JMS), for example, which is part of the Java EE specification, defines an API that you can use for standardization of the interaction model for applications. JMS says nothing, however, about the content of the information to be exchanged between applications.

Message-oriented architecture is for exchanging information (documents) where there is no implied semantics about what should be done with a received document. What does this definition mean? It means that message-oriented architecture is for broad-scale information sharing. An example would be stock ticks. A financial services firm will have a message-oriented architecture backbone (for example TIBCO, MQ, or MSMQ) to distribute changes in stock values to any application that is interested. It doesn't dictate what someone does after they know a stock has changed - it just informs them that it has happened. By this definition, message-oriented architecture is used primarily for data synchronization and event notification. As a result, message-oriented architecture would often be pub/sub-based.

We can base the data model to exchange information on industry standards as well; examples of that include EDI (electronic data interchange), B2MML (XML implementation of the ISA-95 standard) and BatchML (XML implementation of the ISA-88 standard). Note that the data model used here can also be used for the data model that we discussed previously in the "Data-centered architecture" section.

Another data model example is the Open Applications Group Integration Specification (OAGIS). It defines Business Object Documents (BODs) for information such as Items, Bills of Material, Production Orders, etc. Each of these BODs has an Application Area (header) and a Data Area. Both sections of a BOD are comprised of fields and data structures based on a standardized vocabulary (based in part on UN/CEFACT Core Components and published as part of the OAGIS...
standard). The BODS are extensible (in a standardized way) by users and are represented (for data exchange) as XML documents (similar to BATCHML and B2MML). Thus, the OAGIS standard itself can serve as the information model for data exchange (and could be used as the content for a JMS based interaction model) as shown in Figure 2. Put differently, one can use the OAGIS standard to supply the nouns in ontologies as we describe later in this article. (The standard implies the relations, but does explicitly define them.)

Figure 2. OAGIS "production to manufacturing execution" scenario

Data-oriented architecture

Rajiv Joshi, in his article "Data-Oriented Architecture: Loosely Coupling Systems into Systems of Systems" (see Related topics), argues that data-oriented architectures are the best way to integrate real-time systems. He describes the data bus as a key component of the architecture to support this approach. The data bus is an adjunct to the enterprise service bus (ESB), which is a foundation component of a service-oriented architecture.

The Object Management Group (OMG) published a specification that outlines an approach for realization of a data-oriented architecture called the Data Distribution Standard (see Related topics). The specification defines APIs for exchange of real-time in a platform independent, pub/sub model.

The OLE for Process Control (OPC) specification is aimed at the same issue, which is to provide a vendor/device neutral means of obtaining real-time data on the status of production and associated assets, and has much more traction in the industry.

Enterprise application integration (EAI) (brokered data exchange)

The EAI architectural approach builds on messaging oriented approaches to further address the problem of applications needing to contain too much knowledge about issues such as the following:
• Applications that need to be communicated with for specific requests/information
• Protocol for interacting with another application
• Data transformation requirements for interacting with another application

EAI introduces additional integration infrastructure to separate those concerns from the participating applications such as a message broker (for example WebSphere Message Broker or Microsoft® BizTalk) that can handle message routing, transformation, and transaction management on behalf of the applications being integrated. Normally, this is combined with using messaging standards (for example OAGIS) to introduce a canonical form for integration, further addressing the concerns we previously identified.

Service-oriented architecture

Services provide a standardized approach for interoperation between applications or application components. The services and the applications can be deployed on different systems and running on different platforms (J2EE, Windows®, Linux). The service is meant to be an abstraction layer, similar to CORBA IDL in the past, which allows applications to interact in a platform independent manner, without needing to know implementation, or even the location of the service provider. Key elements a service-oriented architecture (SOA) will often include are:

• Service Provider – a component of the architecture providing services to consumers
• Service Consumer – a component (client) that is consuming services
• Enterprise service bus – the integration bus through which services are invoked and through which information flowing between components is mediated
• Service registry – provides a registry and look up service for the services existing within a SOA based system. This can include book look-up and invocation services.
• Process Choreography – a key element of SOA is that composite business processes, flows of services, can be choreographed in a managed way across a number of applications.

SOA differs from both the data-centered architecture and the message-oriented architecture in that there is really no focus on the information flowing between services, and there is not a predefined model. Rather, the goal of SOA is to provide an architectural approach for creation of composite applications that are comprised of a set of composed business processes spanning the applications being integrated.

Within SOA the consumer is interacting with a provider for a well-defined purpose (for example processing an order). Information is very task specific and doesn’t change often. Information changes often require new versions of service providers to support new consumers with new types of information.

Information-oriented architecture

The preceding architectural approaches can complement each other and make sense to use together. The information-oriented architecture extends SOA to include a canonical view of, and access to, the information in the system being integrated to service as the basis for business intelligence and analytics in support process optimization and enhanced decision making. This type of architecture gives us the foundation for composing business processes that collectively
create composite applications around an information model. That model defines the canonical form for data exchange – put differently; it defines canonical side of data mediations.

The information-oriented architecture typically includes master data management (MDM) and business intelligence tools as a complement to SOA. Robin Bloor, in his Data Integration Blog post (see Related topics), points out that an information-oriented architecture might also include a semantic data map, which can help to provide context to the information being accessed in MDM and the integrated applications. This idea is consistent with the basic premise of this article, that however useful the previously described architectural approaches have been over many implementations, they do lack, to one degree or another, "context" for the information being acted on. SOA, combined with standards based messages (for example OAGIS, B2MML, and BATCHML) provides the ability to create and integrate composite processes and applications for services like order management or production tracking. But, there is still no overlying context for the information that can be requested by client applications.

Information-oriented architecture can provide this context by an overlying model of the real world that provides a context for information requests. This way, requests, associated services, definitions of data and more can be associated to an object in the model that will define its meaning and provide its context. As an example, a model can be created for an enterprise based on industry standards, such as ISA-95 and ISA-88, which can be used to define the enterprise hierarchy of an oil drilling platform. That model, at the lowest level of the hierarchy, can contain instances of equipment, such as pumps or motors, to which information requests and actions can be associated. That association then provides the context to support queries such as, "Find the available work orders for this pump", "Report the current temperature of this motor", or "Calculate the average value of pH in this tank over the last week".

One could obtain all of this information, one way or another, with any of the previously described architectures. What the model centered approach does is to introduce context into the discussion in a way that is meaningful to the business, thus simplifying the task of accessing the information and of associating meaningful actions with events related to the modeled objects, which in the example are oil drilling equipment.

**Model-driven architecture**

We're discussing using semantic models to support operations systems integration and, arguably, creation of composite/integrated applications through SOA, middleware, and a common information model. This might sound similar to what is known as model-driven architecture, but really it is very different. Model-driven architecture, explained in detail in Alan Brown's excellent paper "An introduction to Model Driven Architecture" is about using models in the context of application design to drive the development of the application, perhaps including generation of the application code itself (see Related topics). Here, in contrast, we 're talking about using models, in conjunction with SOA and appropriate middleware, to provide context and a common view (and access method) for information available in the enterprise.
Why semantic models?

What exactly are semantic models and how are they helpful for this type of operations systems integration? First, for clarity let's compare models in Unified Modeling Language (UML) versus OWL. UML is a modeling language that is used in software engineering to design artifacts largely around object-oriented systems. When we talk about operational system integration based on information-oriented architecture, in this context, we are really referring to leveraging semantic models as the functional core of an application to provide a navigable model of data and associated relationships that represent knowledge in our target domain.

Semantic models allow users to ask questions about what is happening in a modeled system in a more natural way. As an example, an oil production enterprise might consist of five geographic regions, with each region containing three to five drilling platforms, and each drilling platform monitored by several control systems, each having a different purpose. One of those control systems might monitor the temperature of extracted oil, while another might monitor vibration on a pump. A semantic model will allow a user to ask a question like, "What is the temperature of the oil being extracted on Platform 3?", without having to understand details such as, which specific control system monitors that information or which physical sensor is reporting the oil temperature on that platform.

Therefore, semantic models can be used to relate the physical world, as it is known to control systems engineers in this example, to the real world, as it is known to line-of-business leaders and decision makers. In the physical world, a control point such a valve or temperature sensor, is known by its identifier in a particular control system, possibly through a tag name like 14-WW13. This could be one of several thousand identifiers within any given control system, and there could be many similar control systems across an enterprise. To further complicate the problem of information referencing and aggregation, other data points of interest could be managed through databases, files, applications, or component services with each having its own interface method and naming conventions for data accessing.

A key value of the semantic model then is to provide access of information in context of the real world in a consistent way. Within a semantic model implementation, this information is identified using "triples" of the form "subject-predicate-object"; for example:

- Tank1 <has temperature> Sensor 7
- Tank 1 <is part of> Platform 4
- Platform 4 <is part of> Region 1

These triples, taken together, make up the ontology for Region 1 and can be stored in a model server, as is described in more detail later in this article. This information, then, can be easily traversed using the model query language to answer questions such as "What is the temperature of tank 1 on Platform 4", much more easily than was the case without a semantic model relating engineering information to the real world.

Another advantage of semantic models for this type of application is maintenance. Consider Figure 3.
The real world model we described here can be implemented with any of the types of models shown in the Figure 3. The relational model has relations between entities established through explicit keys (primary, foreign) and, for many-to-many relationships, associative entities. Changing relationships in this case is cumbersome, as it requires changes to the base model structure itself, which can be difficult for a populated database. Querying for this kind of data based on a relational model can also be cumbersome since it can result in very complicated where clauses or significant table joins.

Hierarchical models have similar limitations when it comes to real world updates and are not very flexible when it comes to trying to traverse the model "horizontally".

The graph model, which is how semantic models are implemented, makes it much easier to both query and maintain the model once deployed. For example, if a new relationship is needed to be represented that had not been anticipated during design, with a triple store representation that additional representation is easily maintained. A new triple is simply added to the data store. A critical point is the relations are part of the data, not part of the database structure.

Likewise, you can traverse the model from many different perspectives to answer questions that you had not thought of at design time. In contrast, other types of database design might require structural changes to answer new questions that arise after initial implementation.

Semantic models (based on graphs) allow us to easily make inferences in a nonlinear way. As an example, consider an online service for purchasing books or music. Such an application should be very good at making additional purchase suggestions based on your buying patterns. This is very common for e-tail sites, which provide recommendations such as "Since you liked this movie, you might also like...", or "Because you liked this music, you would probably also like the following...".

One way to accomplish this is to use a semantic model and to add relations such as the following:

Enya <is similar to> Celtic Women
You could also establish in the ontology that both Enya and Celtic Women are part of the music genre called "New Age". These relations, once established in the model, make it simple to offer up those types of suggestions when needed.

Now let's look at the details of semantic models and an example model server deployment approach.

**Semantic models**

As defined by the World Wide Web Consortium (W3C), the Semantic Web "provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries." While the web had generally been about the ability to share documents, the Semantic Web provides the framework so that machines can share, interrogate more readily, and understand data. The Semantic Web supports the notion of common formats for data that a variety of different sources can present. It also provides the structure for understanding the data relationships. This supports the interrogation of web-based data relying on semantic meaning rather than on explicit (or implicit) links and references.

The Semantic Web architecture, as defined by Tim Berners-Lee, is a layered structure with an XML foundation for namespace and schema definitions to support a common syntax. The next layer above the XML foundation supports a Resource Definition Framework (RDF) and RDF Schema. RDF is a framework for a graph representation of resources. While it was created to represent information about web resources, we can use it for a variety of other data types, as we discuss later. The core definition of a RDF element is based on triples in subject-predicate-object form. The machine-readable format for RDF is XML (RDF/XML).

An RDF model essentially defines a graph as described through triples. An RDF Schema (also known as RDF Vocabulary Description Language) provides additional knowledge to the RDF, such as the terms that can be used, restrictions that apply, and what additional relationships exist. You can create an RDF Schema to describe taxonomy of classes (as opposed to just resources in RDF) and formalized relationships between resources (typing and sub classing) to define simple ontologies. You can create more complex ontologies using Web Ontology Language (OWL). The ontology vocabulary is the next layer in the Semantic Web architecture.

As we referenced earlier, an ontology provides an understanding of concepts (terms and relationships) within a domain through a defined vocabulary and model taxonomy. Within a specific industry domain, we can use an ontology to support multiple applications. Additionally, an ontology could support generally applicable terms and relationships that can span multiple domains. Ontologies define entities and relationships to represent the knowledge that we want to share across industries, domains, and applications as appropriate. In order to facilitate this, ontologies support inheritance. Therefore, more generalized knowledge can be captured (referred to as upper ontologies) that can then be further refined to support a specific domain (domain ontologies). As we discuss later in this article, the IBM Integrated Information Core Reference Semantic Model provides an example of an upper ontology.
Semantic understanding of data depends on a common vocabulary that defines terms and relationships. RDF Schema provides a framework for a vocabulary that supports typing and subtyping and the ability to define datatypes. You can create more detailed ontologies using OWL, which relies on RDF Schemas but provides additional language terms in its own namespace. OWL is defined through species or profiles. Providing profiles that restrict the use of terms can make implementations simpler including the inference engines that you can use. We will discuss inferencing and inference engines (reasoners) later in this article. You can use OWL Lite for taxonomies and simple constraints, OWL DL for full expressiveness, and OWL Full for no expressiveness constraints.

The Simple Protocol and RDF Query Language (SPARQL) is an SQL-like language for querying RDF (including RDF Schema or OWL). We use SPARQL to query RDF graph patterns and return results from selected subgraphs. (See Related topics.) You can use SPARQL for querying ontologies and instantiated model data.

Next, we explain the role of the model server as a run-time "host" for the semantic model.

Model servers

The model server (or model manager) provides the run-time framework on which to deploy the model. A model server needs to support a number of key functional services to persist and manage the model (ontology) and the model instance data. It also needs to provide tooling and application interfaces for model and instance data queries and updates. Let's look at this capability in more detail using open source projects like Jena, Joseki, Sesame, and Pellet as examples.

Model servers can support a number of different persistence layers that include database and file (typically in RDF/XML format; although N3 and Turtle are two other popular notations). While you could use relational databases to support RDF data persistence, querying of RDF data (graph-based data) stored in an RDB is often inefficient and you may lose the ability to change the data model without changing the db schema. A triple store is a special purpose database designed specifically for storage and querying of RDF data. The data structure is optimized for data stored in a triples structure that corresponds to the RDF subject-predicate-object form. Both Jena and Sesame provide triple stores.

When we think about model servers at this level there isn't yet any requirement to understand the structure of the persisted data. However, as additional model server function is considered, an understanding of the data becomes relevant. Jena and Sesame provide good examples.

First, we should note that Jena provides a Java framework for building Semantic Web applications rather than providing a complete model server. Specifically, Joseki, an open source sub-project to Jena, provides server capability through both an HTTP interface to the RDF data and an interface for SPARQL querying and updating. In addition, Jena provides a programming interface to the RDF data and an inference engine. With this additional capability, Jena does need to understand the RDF ontology. Reasoning or inferencing means being able to derive facts that the ontology does not directly express.
Jena provides an inference engine to support reasoning in RDF, RDFS, and OWL, but some instances are incomplete. Jena provides a pluggable interface so that additional inference engines can be integrated. For example, Pellet is an open source Java reasoner fully supporting OWL DL that can be plugged into Jena. With this type of extensibility, Jena supports languages such as RDFS and OWL and supports data inference from instance data and class descriptions.

Like Jena, Sesame provides a Java framework that supports persistence, an interface API, and inferencing. However, the inferencing capability within Sesame supports RDF and RDFS but not OWL. For a set of RDF or RDFS data, you can query Sesame and find the implicit information. Because anything that can be inferred can also be asserted, one approach to supporting inferencing is to explicitly add the implicit information to the repository as the data is initially created. This is the Sesame approach.

Next we’ll talk about the semantic model provided with IBM Integrated Information Core that draws on a number of industry standards to create a meta model that provides asset definitions integrated with enterprise operations structure. That model, in the form of an ontology and manifested in an RDF, will be deployed on a model server provided with Integrated Information Core that supports some of the capability described here.

**Semantic models and IBM Integrated Information Core**

The purpose of IBM Integrated Information Core is to provide a framework that makes it much simpler to create applications that are centered on a semantic model of the real world, and that support integration of real-time operational data and related enterprise applications. The key component of the Integrated Information Core architecture supporting this goal is the semantic model which, based on industry standards (centered largely on ISA-95 and ISA-88), supports the definition of an enterprise model down to assets and associated measurements.

The information model included with Integrated Information Core is the Reference Semantic Model. It meets our definition of semantic models because it provides a real world abstraction of the enterprise and assets in a graphical model. Through it, applications can access information from disparate systems with various access methods. The information model in Integrated Information Core contains named entities based on industry standards (today, primarily including ISA-95, ISA-88, and ISO15926) and relationships either defined by those standards or implied by combining the standards into one, homogenous model. The Reference Semantic Model can be queried through services or (based on the deployment) through a SPARQL interface.

Another key component of the Integrated Information Core architecture is the model aware adapters layer that support integration of various types of endpoints (OPC, databases, and web services accessible applications), and maps of the information flowing between those endpoints and elements of the model.

There are really two views of the Integrated Information Core semantic model:

1. **Reference model** (the ontology)
This view defines the classes that exist in the model and the relations between them, but does not correspond to any particular enterprise or asset.

2. **Instantiated model**

This view includes instances of the classes that have a direct mapping reference to real-world entities. They are populated with a set of properties (for example, s/n, location, temperature) and with relationships to other instantiated entities in the model.

As an example of how the industry standards based model in Integrated Information Core are used to model the real world, consider the following example (based on a project for manufacturer of paint).

As an example of how the industry standards based model in Integrated Information Core is used to model the real world, consider the following example based on a project for manufacturer of paint.

First, as shown in Figure 4, classes from ISA-95, Enterprise, Site, Area and Production Unit (found as reference classes in the RSM model) are instantiated. These, along with an additional Work Equipment class, are used to define a physical model starting from an enterprise level down to the level of specific pieces of work equipment.

**Figure 4. Enterprise hierarchy based on industry standards**

![Enterprise hierarchy based on industry standards](image-url)

(View a larger version of Figure 4.)

It is at the work equipment level, typically, that measurement classes can then be attached and mapped to end point data adapters and specific data sources.

After the model has been instantiated and mapped to endpoints through the adapter layer we can use it in a number of ways to achieve the previously described business benefits:

- Applications in the paint manufacturing enterprise that need to obtain information about an asset, such as a tank, can now go to a single location, that is, the model server hosting the instantiated model, to access that information. This can include real-time information on the
tank (for example, temperature), historical information (for example, average temperature this week), or more complex types of information (for example, open work orders for this tank, or tanks of this type).

- The queries made by the applications to get operational information about the tank can be made using a consistent interface method (for example, SPARQL) regardless of the true source of the information such as SCADA systems, operational database, or an application (for example, IBM Maximo or SAP).
- The representation of the tank and the enterprise hierarchy around the tank is consistent and based on industry standards. That canonical form remains intact regardless of the underlying format used in the endpoint systems.
- The tank information can easily be extended to introduce new information that is deemed to be useful in the future. For example, a new requirement to relate to equipment failures in an external asset management system can easily be tied to equipment in the model so that the failure information can be queried through the same model context. The model also provides a canvas, based on real world context, to simplify configuration for aspects of production control such as calculation of KPIs (key performance indicators), definition of actions needed for operational events and generation of alerts for detected problems. That type of information can now be associated with an object in the model and it can then easily be made sensitive to context in the model.
- Likewise, the relations in the semantic model now make it much easier for applications to look at this information across the model laterally to answer questions that were not anticipated in the initial creation of the model. As an example, it might be that our Paint enterprise contains similar types of motors that can serve the same function, but which come from different suppliers. Through relations in the model such as "Motor type A <is equivalent to> Motor type B" we can easily produce a report showing performance characteristics of all the similar motors currently being used in production (across locations, if need be) so that we can make better supplier decisions in the future. We might also conclude, in doing so, that we need a maintenance action to replace one type of motor because another type is performing much better. Note in this example that the relations showing equivalency need not have been in the originally implemented and deployed model, these could be added later based on new knowledge.

In summary, Integrated Information Core extends the capability of application integration based on a semantic model.

- **Model business entities**
  Model business entities (for example, tanks, pumps) and their relationships so that we can support data queries, which might be contained in a number of different systems, in a real world context. This is a powerful concept and it allows us to establish intelligence across the entities (and underlying systems) to support analytics and optimization aimed at things like failure prediction, detection of abnormal behavior, detection of and prevention of product quality problems.

- **Establish global namespace**
  Establish a common naming definition, and information access method, so that an application can reference entities such as assets that might be named and identified differently by
multiple enterprise subsystems in a way that protects the application from knowing the details of those subsystems (for example, SCADA/DCSI Systems, OPC Servers, SAP, or Maximo).

- **Define canonical form**
  Define a canonical form to reference information associated with business entities in the enterprise. For example, a tank being used for mixing of paint might have temperature information that can be obtained from lower level OPC servers, or work orders that can be obtained from SAP or Maximo. As was previously mentioned, you can use industry standards to supply definitions for that canonical form, which has the advantage of building on accepted definitions and vocabulary for common entities such as equipment, locations, personnel, and more.

- **Provide enterprise application interface**
  Provide a global interface for applications to query and update business entities and their associated data so that the application does not need to know which subsystem owns any given entity or associated data (for example OPC servers, SAP or IBM Maximo). The application will be provided with a full enterprise view of the data, based on the model of the real world that corresponds to the information. This makes addition of new underlying systems much simpler, since the specifics of that are hidden behind the model.

### Conclusion

In this article, we looked at the value of semantic models in building solutions. We discussed this architecture in context of a number of widely used and well known solution architectures that center on data, messaging and services. We described semantic models in general terms and then discussed how IBM Integrated Information Core delivers on the value of providing a semantic model based foundation to build solutions that drive business insights and efficiencies.

As described here, semantic models, play a key role in the evolving solution architectures that support the business goal of obtaining a more complete view of “what is happening” within operations and then deriving business insights from that view. Semantic models based on industry standards take that one step further, especially as application vendors adopt those standards (which, as always, will happen more rapidly through pressure from the user community).
Related topics

- **IBM Integrated Information Core**: Find more information about this product.
- **IBM Integrated Information Core Information Center**: Access information to install, maintain, and use this product.
- **IBM Service Oriented Architecture**: Learn more about this business-centric IT architectural approach.
- **W3C Web Services Architecture**: This document defines the architecture, identifies the functional components, and defines the relationships among those components to effect the desired properties of the overall architecture.
- **Open Applications Group**: Explore this community that focuses on building process-based business standards.
- **Core Components Technical Specification – Part 8 of the ebXML Framework**: This document from the United Nations Centre for Trade Facilitation and Electronic Business contains information to guide in the interpretation or implementation of ebXML concepts.
- **Ontologies and Semantic Web**: Access more information from Marek Obitko.
- **Jena – A Semantic Web Framework for Java**: Check out this Java framework for building Semantic Web applications.
- **Introduction to Jena** (developerWorks 2004): In this article, developer Philip McCarthy shows you how to use the Jena Semantic Web Toolkit to exploit RDF data models in your Java applications.
- **W3C - RDF Vocabulary Description Language 1.0: RDF Schema**: Learn more about this general-purpose language for representing information in the web.
- **W3C – RDF Semantics**: This is a specification of a precise semantics, and corresponding complete systems of inference rules, for the Resource Description Framework (RDF) and RDF Schema (RDFS).
- **Simple Protocol and RDF Query Language (SPARQL)**: This specification defines the syntax and semantics of the SPARQL query language for RDF.
- **Fern Halper on semantic models**: Read what this expert on data analysis, business analysis, and strategy development has to say about semantic models.
- **Data-Oriented Architecture: Loosely Coupling Systems into Systems of Systems**: Here's the full article by Rajiv Joshi, which was published through RTC January 2008.
- **An introduction to Model Driven Architecture** (developerWorks 2004): Alan Brown examines the importance of models and modeling, introduces four key principles of model-driven architecture, and looks at the leadership role that IBM has played in defining the approach and its supporting standards.
- **Explore IBM Integrated Information Core model aware adapters** (developerWorks, February 2012): Learn about the data adapters provided with the product and how you can extend them to support customer-specific data sources.
- **developerWorks on Twitter**: Join today to follow developerWorks tweets.
- **IBM product evaluation versions**: Get your hands on application development tools and middleware products.