

# The information perspective of SOA design, Part 4: The value of applying the canonical modeling pattern in SOA

Skill Level: Intermediate

[Brian Byrne \(byrneb@us.ibm.com\)](mailto:byrneb@us.ibm.com)  
Industry Models and Integration Architect  
IBM

[John Kling \(jkling@us.ibm.com\)](mailto:jkling@us.ibm.com)  
Consulting and Services Architect  
IBM

[David McCarty \(davidmccarty@fr.ibm.com\)](mailto:davidmccarty@fr.ibm.com)  
IT Architect  
IBM

[Dr. Guenter Sauter \(gsauter@us.ibm.com\)](mailto:gsauter@us.ibm.com)  
Senior IT Architect and Manager  
IBM

[Peter Worcester \(pworcest@us.ibm.com\)](mailto:pworcest@us.ibm.com)  
Services Solution Marketing Manager  
IBM

13 Mar 2008

Discover the approach and value of canonical modeling in SOA design. See how the canonical data models can be aligned in SOA with canonical message models. In this fourth article in the "Information Aspect of SOA Related Design" series, learn about the concept's underlying data and message modeling regardless of the technology and tool choices. A future article in this series describes how various IBM® software products can be used to implement the concepts described here.

## Introduction

**Read all the articles in this series**

1. [Introduction to the information perspective of an SOA](#)
2. [The value of applying the business glossary pattern in SOA](#)
3. [Use the IBM WebSphere business glossary in SOA design](#)
4. [The value of applying the canonical modeling pattern in SOA](#)
5. [The value and use of Rational Data Architect in SOA](#)
6. [The value of applying the data quality analysis pattern in SOA](#)
7. [Use of IBM WebSphere Information Analyzer in SOA design](#)

The first three articles in this series describe the role and implementation of a business glossary in an SOA. The business glossary defines the individual terms that describe an organization's information. The canonical data model defines the structure of an organization's information. The objective is not to limit data modeling to a single database and its related physical data model. Rather, the canonical data model is the reference for all of the entities and their relationships across all of the databases and related legacy applications that the SOA encompasses.

The data model may be developed top-down solely based on business requirements or bottom-up through reverse engineering of existing data structures. Either approach may use industry data models such as the IBM Insurance Application Architecture (IAA) for the insurance industry or the ARTS data model created by the National Retail Federation for the retail industry. Regardless of how the data model is created, it represents the common structure of the data in the SOA.

The canonical data model is developed incrementally during the SOA project. Its information domain coverage corresponds to that which is in scope for the SOA project. During business analysis, requirements gathering, and use-case design, the model deliberately lacks detail, showing only the information concepts that are most important for the business. As the project moves into technical modeling and detailed design, the canonical model is developed to fully support these activities. In subsequent SOA projects, the canonical data model is enhanced to include additional business areas and information types as they come into scope for that organization's SOA.

This article describes the motivation for using canonical models. It goes on to provide additional detail about how each is developed and used in an SOA.

## Motivation

Implementing successful SOA projects requires a careful balancing of conflicting factors. For example, broad and detailed business analysis is needed to find widely reusable business services. At the same time, experience has shown that a series of short, incremental projects delivering specific business objectives is the most manageable and successful approach for developing an SOA. The price paid for this pragmatic approach is often to create "quick and dirty" service implementations, to focus only on the requirements at hand and to speed up implementation. Over time,

these services restrict the flexibility and extensibility of the overall SOA as new requirements drive the creation of an increasingly large and complex portfolio of services.

One symptom of this approach is revealed by the inspection of the data structures being exposed through service messages. If there are many services exposing arbitrary variations of the same business information, or many instances of data mapping running in the enterprise service bus (ESB) and business processes, then it is likely that a simpler and more reusable solution could have been developed with appropriate attention to the data architecture of the solution. Left unchecked, this approach can lead to a solution so complex that all the benefits of the SOA approach are lost.

A canonical data model provides a common format for the information content of the messages of the individual services. This leads to a horizontal alignment of message formats across services.

Canonical message formats often require data from more than one physical table. This means that the messages must contain the structural information to allow the service to join data from separate tables. For this to work, a canonical message model must be based on a canonical data model. Many companies have already developed a canonical data model for the most important entities across their enterprise. This can be leveraged and extended to support the SOA analysis and design process.

It is important to consider the vertical alignment of data formats in the SOA architecture layers as well as horizontal alignment. If the message format of the services is very different -- without a specific reason -- from the canonical data model and ultimately the data as persisted in a database, the SOA needs to implement potentially complex transformation operations. They are required to map between the different formats when the data flows from the service interface down to the database and back.

In addition to avoiding the problems caused by a lack of horizontal and vertical alignment, the canonical data model can be used to improve developer productivity by improving communication between the business users of the data and the application developers. Eventually, all of the work that a business does comes to rest in its data. Understanding the business nuances that are inherent in the data is one of the quickest and surest ways for an application developer to obtain an understanding of the business. In developing application services, the application developer needs this understanding to ensure that applications meet the information needs of the business users.

Finally, the canonical data model is a critical foundation for effective data governance. The model helps to address such key questions as: "How many systems store portions of each common entity?", "Which systems maintain common

data?", "What is the lineage of data? ", "Can the quality of the information be trusted?", and so on.

Issues related to data governance can be some of the most complex and important issues that a project has to address. Data governance issues can have a significant impact on IT strategy since key initiatives can succeed or fail based on data availability and data quality.

## The canonical data model

Depending on the complexity of the project, a canonical data model may be specified at two different levels of abstraction and granularity. However, in the majority of cases, these are simply two phases of definition within the same actual model. Conceptual data modeling drives the initial broad specification within the canonical data model, and logical data modeling adds further detail within that same model.

### The conceptual data model

A conceptual data model is a canonical data model viewed at the highest level of abstraction. This aspect of the model represents the strategic information requirements of the enterprise within the scope of the SOA project.

Business entities are the fundamental building block of the conceptual data model. An entity represents a grouping of attributes around a single, key, unifying information concept, for example, person, product, order or payment. Each attribute represents an aspect of the information concept at a more granular level, for example, first name, expiration date, or SKU number. Both attributes and entities are based on and aligned with the business glossary regarding terminology and semantic definitions. The conceptual data model is created during the initial phases and typically includes only entities and their most important relationships.

While not always a part of the conceptual data model, a few major attributes may be defined for illustration purposes. All of the attributes that comprise each entity are added later as part of the logical data modeling phase.

The development of a conceptual data model has several purposes within the SOA project. The main reasons for developing a conceptual data model in the SOA project are:

- To provide a shared representation of the major high-level groups of information used in all aspects of business analysis, business process design modeling, use case modeling and candidate service identification within the scope of the SOA project.

- To provide a shared starting point and boundaries for subsequent modeling and design activities of the project. It sets the baseline information context in which all services will operate for the project across all participating lines of business.
- To provide a useful vehicle for Data Governance regarding responsibilities for information creation, update, deletion, distribution and use. The policies for data ownership and use are tightly coupled to similar responsibilities defined against service ownership and use in the SOA governance model.

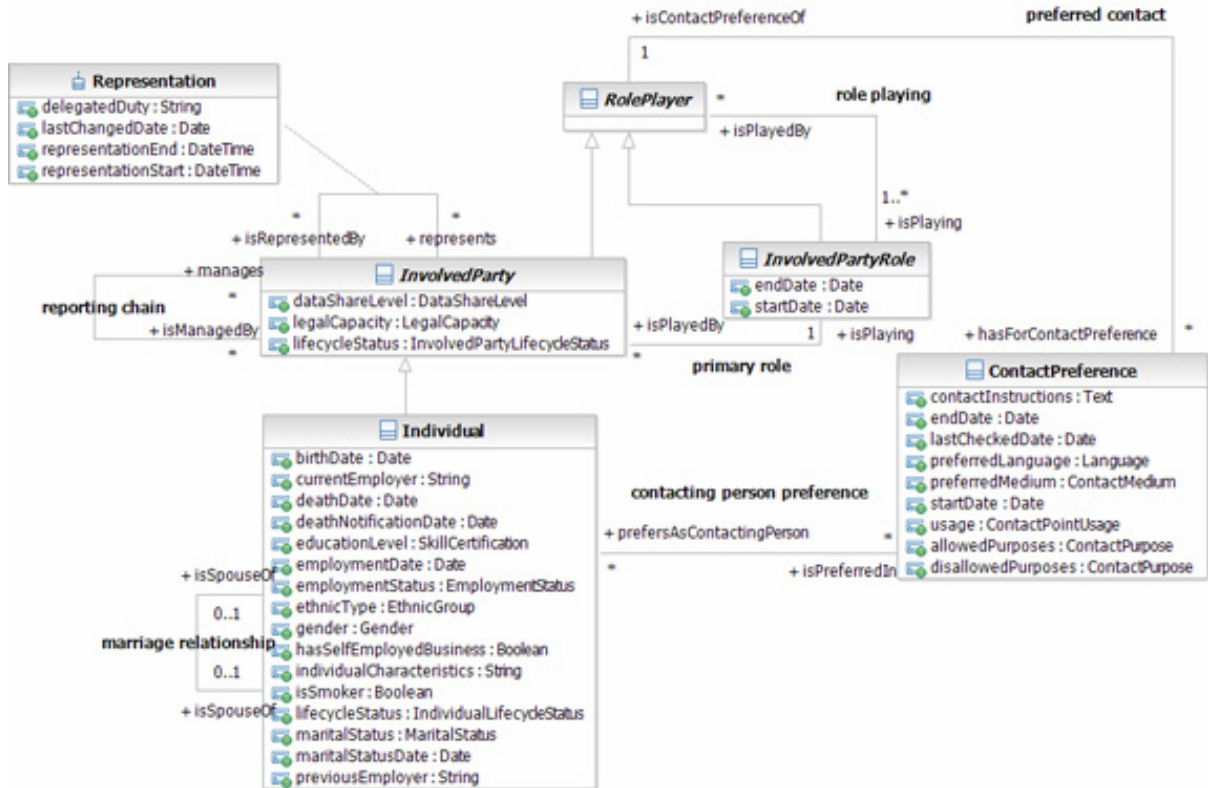
The conceptual data model typically is coarse-grained and is intended to show the broad set of entities and relationships under consideration. It is not intended to be granular enough in perspective for service specification, but rather to provide a high-level view of the information universe under consideration.

This model enables the SOA team to demonstrate an understanding of the structure and content of the customer's information requirements. This provides:

- A basis for identifying information-sharing opportunities between candidate services in support of asset reuse
- A means for reducing the risk of missing information needs or failing to align the SOA management of information with the wider information needs and strategy for the enterprise
- A starting point for analyzing information-sharing requirements and information quality analysis, across the business units within the enterprise including the impacts of future changes on the SOA solution. This means that the accuracy of the estimation of subsequent projects is less likely to be affected and there is less potential for delay in sequential projects.

While information modeling is most often documented using information-engineering notation, it is important to note that this same model can be transformed into Unified Modeling Language (UML) and exposed to service analysts, driving consistency across multiple modeling domains. To highlight this, the example below shows a conceptual model expressed in UML. You can see that the same model content can readily be interchanged between UML and IE notations.

### **Figure 1: Example of a conceptual data model**



### The logical data model

The logical data model is based on the conceptual data model developed earlier, and is usually a further expression of detail within the same model, rather than an entirely separate model mapped to a conceptual model. The role of the logical data model is to provide input to the development of data structures or data representations across the architecture layers: in physical data models, message models, and component models. In addition, the logical data model may be used in service analysis and exposed to service consumers such as business process models.

The following define the most significant differences between the conceptual and logical models:

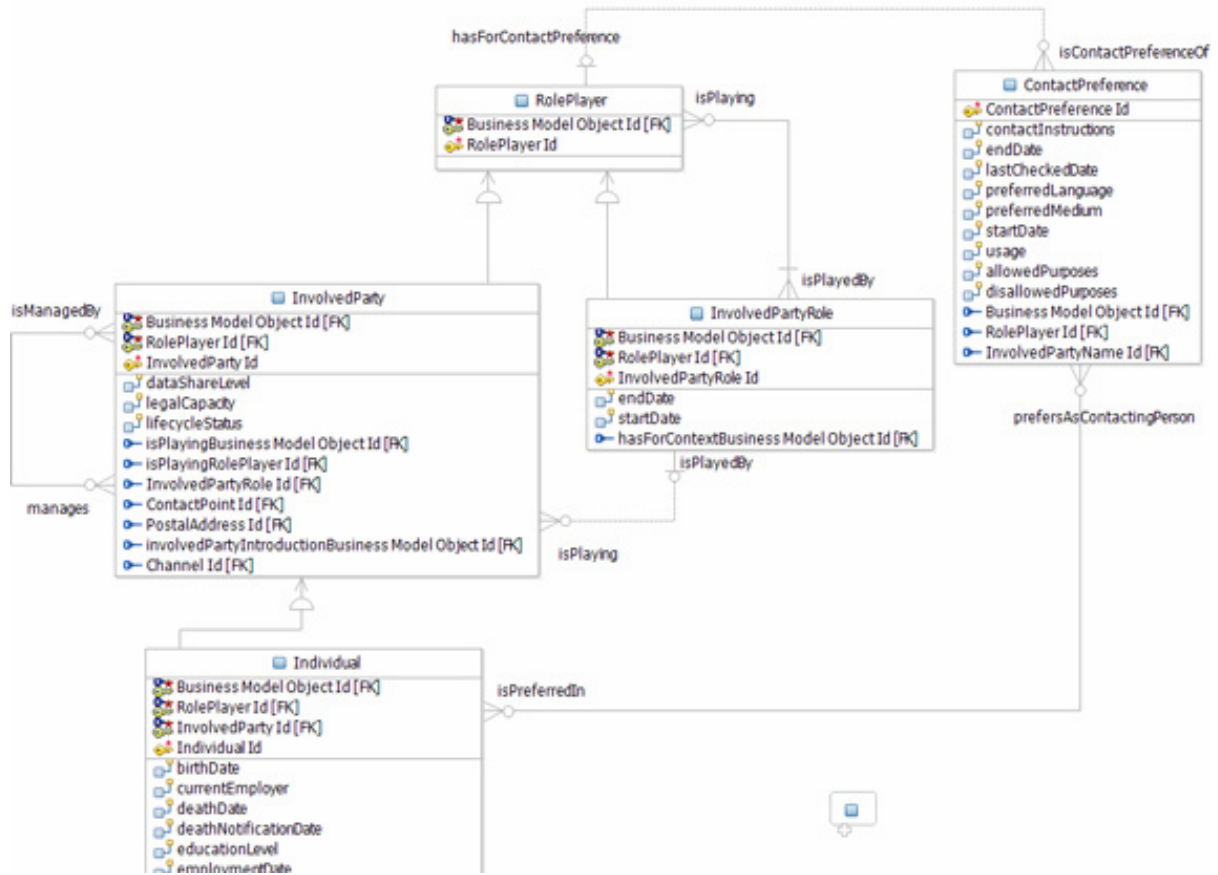
- Each entity in the logical data model is assigned a primary key. An entity's *primary key* is the attribute or set of attributes that distinguish one instance of the entity from another. The main characteristics of the primary key are that it is unique and stable -- it is unchanging over time. Multiple attributes in the entity may be combined to form a composite primary key. For example, if Account Numbers are assigned across all types of accounts, Account Type may need to be used along with Account Number to form the primary key of the Accounts entity.
- Unlike the conceptual data modeling phase, where a few attributes may

be shown for illustration purposes only, all of the attributes for each entity must be included during logical data modeling. The attributes included in the logical data model must be sufficient to support all aspects of the information structures being specified. Attributes should use business names that are consistent with the business glossary and a consistent naming convention.

- Relationships between entities in the logical data model are represented through foreign keys that are associated to the primary keys of the referring entity. If a Sales Order entity refers to a Customer entity which is identified by the Customer Number primary key, then the Sales Order entity will have a Customer Number as a foreign key. A Customer instance is related to a Sales Order instance when the primary key in Customer and the foreign key pointing to Customer in Sales Order have the same value. Sometimes, relationships expressed during conceptual data modeling (in particular many-to-many relationships) are shown in the logical model as entities themselves. For example, the conceptual model may show a simple recursive relationship from Person to itself. In the logical model we add an entity called RelatedPerson which holds the keys of the two related people along with any attributes defining the relationship itself such as RelationshipType="spouse".
- Normalization decisions are finalized in the logical data model resulting in the final normalized representation of entity-to-entity relationships as well as supertype-to-subtype hierarchies.
- Attributes in the logical data model are assigned a physical data type. For vertical alignment, this should be consistent with the physical data model that is used to generate the database schema. For horizontal alignment, this must be consistent with the message model.
- Rules governing domains of specific allowed values, ranges of allowed values or other logical restrictions on data values are specified. This includes cross attribute restrictions (e.g. ship date must be greater than an order date). Just because these rules are part of the logical data model, one should not assume that they must be enforced by a database. Enforcement of data rules is the responsibility of the service that maintains these entities. The service may use a database service or use application components to enforce these logical data rules. Decisions regarding how rules are enforced and by which services are a key aspect of data governance.

The last two bullets in the above list are beyond the scope of what is normally associated with a logical model. However, in the context of an SOA, they are essential due to the stringent requirements for consistency and quality in the data shared between services. Figure 2 shows an example of a logical data model:

## Figure 2. Example of a logical data model

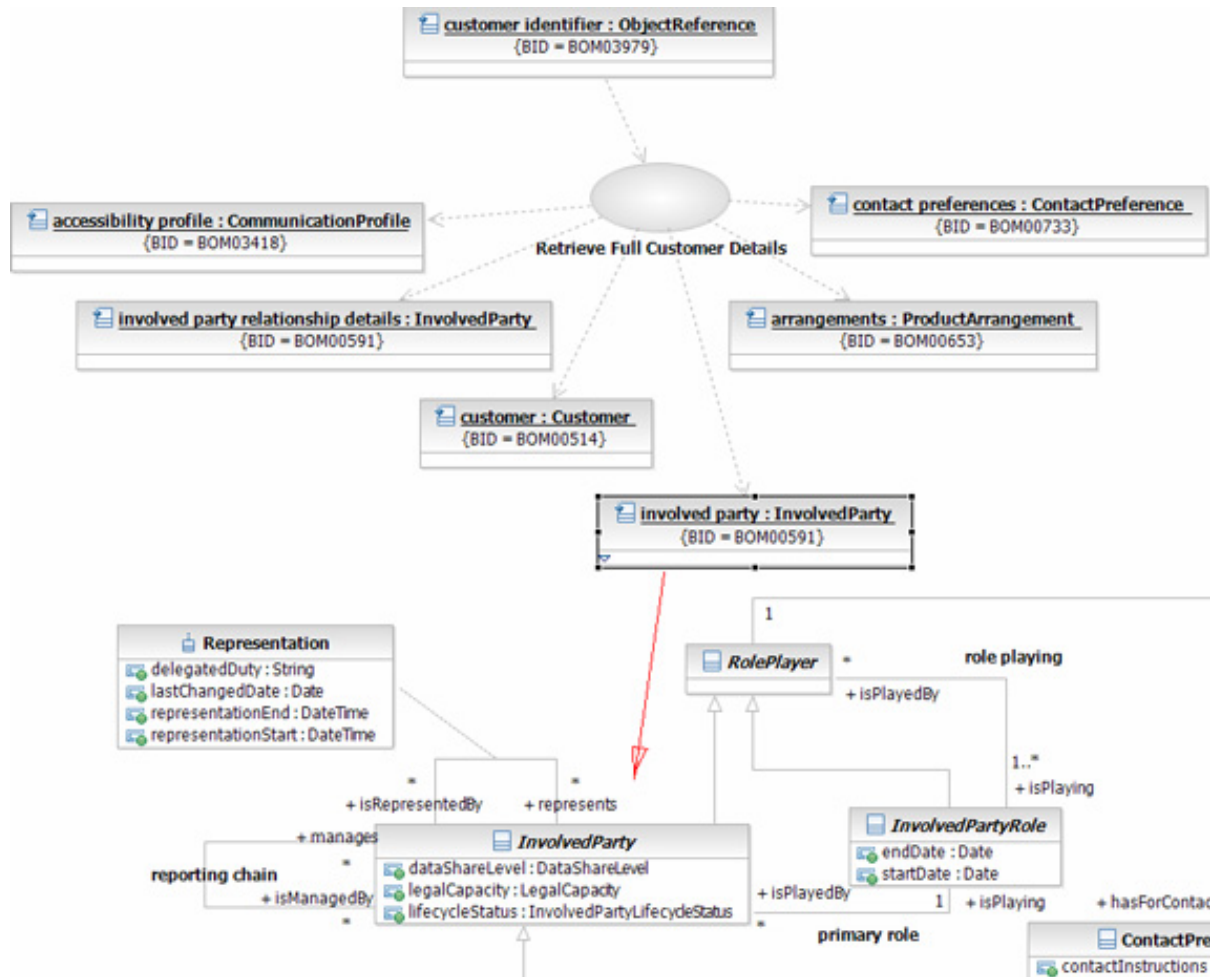


### The service analysis model

The service analysis model represents the data structures for service and component specification. The structures expressed within the service analysis model are a direct reflection of those within the conceptual model (or logical data model where no conceptual model formally exists). In many cases, the canonical data models employ information-engineering techniques and the service analysis models employ UML techniques with business definitions being interchanged between the two. The specifics of this interchange of conceptual definitions is discussed in the next article in this series.

Figure 3 shows an example of a service analysis model:

**Figure 3. Example of a service analysis model**



The goal of a service analysis model is to guide the specification of software components and services in a way that is compatible with the overarching data architecture. While the canonical data model describes the business entities, attributes, and relationships, in a normalized form structured to reflect their business use, the service analysis model results in defined aggregations and de-normalizations of this canonical model based on reuse characteristics of this data and the needs of service consumers.

In many cases, a single component model may be defined from which canonical message models (in XML form) and component specifications are derived. In this case, the conceptual model still guides the definition of business types within the component and of those passed through the interfaces of the components. This brings the additional benefits of alignment of data structures across data persistence, software components and service definitions.

## The canonical message model

The canonical message model represents the standardized format used for exchanging business information on a service bus. Not every data structure passing through the different layers of the architecture necessarily complies with this model. But rather, the model provides the default business data interchange formats so that all components need only to know (at most) their own data format (perhaps within a software component) and the default data format (that shared on a service bus). While based on the logical data model, the canonical message model is often significantly de-normalized to reduce the complexity of both relationship graphs and type hierarchies in order that service messages are more easily understood and digestible by service consumers.

The canonical message model may be arrived at using a variety of methods depending on each project. For projects driven predominantly top-down, the service analysis model leads through service design modeling to eventually arrive at technical service specifications, component specifications, and a design-level class model which includes the data types and structures making up the canonical message model. For projects focusing on ESB integration and bottom-up service exposure, the message model may be created directly in XML either manually, or as a result of message flow design and service code generation tools.

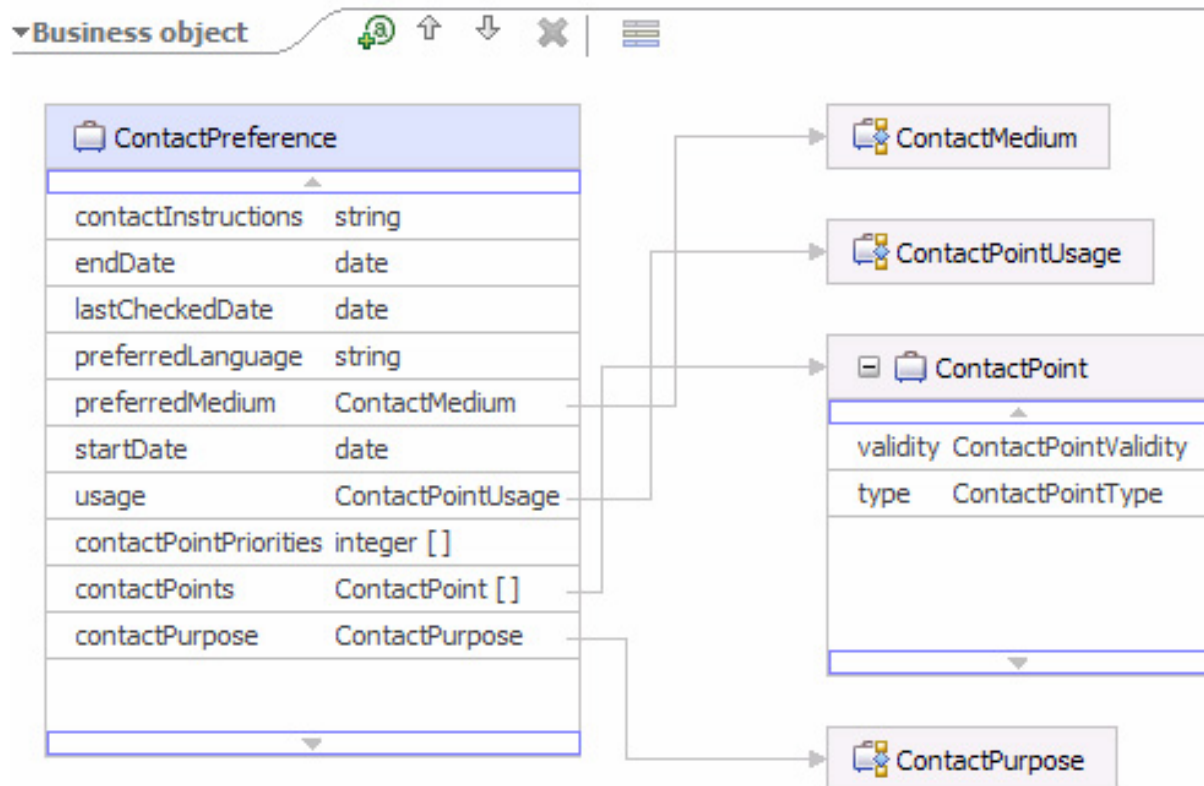
The most common representation used for the canonical message model is a set of XML schemas. This has the benefit of making the type and message definitions directly reusable in the WSDL schemas that describe the exposed services.

The canonical message model consists of the following:

1. A defined set of types, elements and attributes representing the business entities and their business attributes used in all messages. Each definition includes:
  - Technical data types, formats, structures and names
  - Rules governing the allowable values of the type
  - Business semantics of the type
2. A defined set of messages, each including a related set of the previously defined types, elements and attributes structured to provide a business document with a specific semantic meaning and context.

Figure 4 illustrates a canonical message model:

#### **Figure 4. Example of a canonical message model**



The canonical message model is ultimately based on the logical data model and business glossary, via a service analysis model. The canonical data model describes the business entities, attributes, and relationships in a normalized form structured to reflect their business use. The canonical message model provides a set of views over the canonical data model, each exposing a small subset of related entities as a reusable message, but always following the entity and relationship structures defined in the canonical data model.

## Conclusion

In summary, there are a range of interrelated models used to analyze and define the data structures involved in service contracts, software implementations of those services, and the data platform. Consistency across these expressions of data type is critical and is driven through the specification of a canonical data model. A canonical data model is a common representation of entities, their attributes and relationships based on the business requirements in the SOA project. It should be aligned with the business glossary, the process, the service/message model and the physical data models. This ensures semantic and structural interoperability when the data flows through the architecture layers during service execution.

If this concept addresses problems that you have in your project and you are interested in understanding how a tool could help you with your data modeling

challenges, stay tuned for the next article in this series titled "Use of Rational Data Architect in SOA."

# Resources

## Learn

- Check out the rest of this [series](#).

## Get products and technologies

- Create, manage and share an enterprise vocabulary and classification system with [IBM Information Server](#) and in particular [WebSphere Business Glossary](#).
- Use [Rational Data Architect](#) to simplify data modeling and integration design.
- Utilize [IBM Industry Models](#) to accelerate projects and reduce risk.
- Explore [Data Studio](#) for a more comprehensive data management solution to design, develop, deploy, and manage data-driven applications.

## Discuss

- Participate in [developerWorks blogs](#) and get involved in the developerWorks community.

# About the authors

## Brian Byrne

Brian Byrne has over 10 years experience in the design and development of distributed systems, spending 7 years driving the architecture of Industry Models across a range of industries. Brian is currently an architect within IBM's Information Management organization.

---

## John Kling

John Kling is an architect in the Information Services Practice within IBM's Global Business Services. He is responsible for leading large client engagements that focus on data quality, data integration and master data management. He is currently the data team lead for the SAP implementation of a Fortune 500 industrial company.

---

## David McCarty

David McCarty is based at IBM's European Business Solution Center in La Gaude,

France and has 20 years experience designing and developing IT systems with IBM customers. He is currently a member of the Information as a Service Competency Center developing techniques and best practices for leveraging data systems in SOA solutions.

---

#### Dr. Guenter Sauter

Guenter Sauter is an architect in the Information Platform & Solutions segment within IBM's software group. He is driving architectural patterns and usage scenarios across IBM's master data management and information platform technologies. Until recently, he was the head of an architect team developing the architecture approach, patterns and best practices for Information as a Service. He is the technical co-lead for IBM's SOA Scenario on Information as a Service.

---

#### Peter Worcester

Peter joined IBM three years ago after almost 25 years at institutions like the US Dept. of Defense, GE Corporate and Morgan Stanley where he held technical leadership positions and gained valuable experience in Enterprise Architecture and Enterprise Data Integration. He initially joined IBM as a Sr. IT Architect as part of the architect team for Information as a Service. Currently he is a Solutions Marketing Manager for the IPS Global Services organization, specializing in MDM solutions.