Implementing the TM Forum Information Framework (SID)

A Practitioner's Guide

Version 1.0 *September, 2011*

Implementing the TM Forum Information Framework (SID): A Practitioner's Guide

By John P. Reilly

Published by **TM Forum**

First printing: [insert printing date]

ISBN [insert ISBN number]

Table of Contents

List of Figures

Acknowledgements

When the number of books about TM Forum deliverables, such as Frameworx Distilled, reached five, I thought I was done. But then I started thinking about the members who have often asked about using the Information Framework (SID) as a starting point for database development. And, a few other SID-related topics. So, I thank you all for providing me with the inspiration to write a sixth and perhaps final book, at least from me!

I would like to thank the TM Forum, which permitted the use of content from various SID documents and models. And, to IBM who have kindly provided licenses to me for the use of Rational Software Modeler and InfoSphere Data Architect.

I can't forget thanking my wife, Jeannie, who puts up with my crazy traveling schedule and the somewhat odd hours I work during my time at home with her and Sara the cat.

John P. Reilly

Preface

The adoption of the TM Forum's Information Framework (SID) continues to increase within the information, communications, and entertainment industry. As a result, there is a need to provide advanced guidance on how the SID can be implemented by TM Forum members. Specifically, members expect guidance on implementing the SID for use in interface specifications as well as database design.

This book introduces more advanced concepts/techniques that can be used to successfully implement the SID within an application environment. They have been gathered and used by many SID practitioners from both teaching and consulting engagements. The culmination of this experience resulted in the practical guidance presented here for using the SID from an implementation perspective. Examples are used throughout the book to demonstrate the concepts and techniques presented.

In addition to providing an overview of the SID, this book's chapters provide guidance for:

- Implementing SID modeling patterns
- SID and database design
- Maintaining SID conformance
- Instantiating SID entities
- Applying other implementation considerations

Chapter 1 – The Information Framework (SID)

This chapter provides an overview of the SID, focusing on the its benefits, its structure, and its inherent extensibility.

The Information Framework (SID)

The Information Framework (SID) responds to the industry need for a shared information and data model. It represents an enterprise-wide information decomposition and model. An information model is independent of platform, language and protocol – today the Information Framework is an information model. One single information model can be used as the starting point for multiple technology-specific data models, such as technology-specific databases and technology-specific interfaces.

As a federated model, the SID draws on knowledge from other industry associations, standards bodies, authorities, and TM Forum member companies; it is not "home-grown". The origins of the SID come from member contributions as well as existing industry models, such as ITU-T M.3100 and Distributed Management Task Force (DMTF) Common Information Model (CIM).

The reasons for having an information model are also relevant from a shared information architecture perspective. The SID is one of the few industry models that cover the breadth of an entire enterprise. The SID provides a standard way of structuring, defining, and implementing information and also provides consistent, common terminology.

The model includes all concepts typical of an object-oriented model with the exception of behavior, or how things work. Behavior is provided not only by services and interfaces that make up the Integration Framework, but also the Process Framework, which also represents behavior.

The Information Framework, specifically Product and Service related entities, are now an ITU-T standard as of July 2008.

The framework assists in achieving a number of benefits:

- Reduce time to market
- Reduce cost of integration
- Reduce management time and cost
- Facilitate introduction of new technologies
- Support multiple technology implementations.

From a SID viewpoint, the first goal, reducing time to market, is related to the fourth goal, facilitating the introduction of new technologies and services that utilized the technologies. The inherent extensibility of the SID enables new technologies to be introduced without making major changes to the SID framework or its underlying models. Changes to information architectures that do not use SID modeling patterns typically require changes to the information architecture when introducing new technologies and the product offerings made available by these technologies. Additionally, sometimes substantial changes must be made to the applications that support the offerings and technologies. These changes could range from the need for new user interfaces to large fragments of code and/or database changes. The changes are minimized by employing the SID and result in the ability to reduce the time it takes to bring these new technologies and offerings to the market.

Implementing the SID also achieves the second goal, reducing the cost to integrate processes and applications. The SID facilitates this in a number of ways. First, the use of the SID as part of an integration framework provides a standard information vocabulary, framework, and model that reduce the time to translate the vocabulary, framework, and model among applications that are being integrated. Second, the SID can be employed as an operational data store (ODS). An ODS is a database designed to federate data from multiple sources to facilitate operations, analysis and reporting. Because the data originates from multiple sources, the integration often involves cleaning, redundancy resolution and business rule enforcement. An ODS is usually designed to contain low level or atomic (indivisible) data such as transactions and prices as opposed to aggregated or summarized data such as net contributions.

Using the SID also achieves the fifth goal, Support implementation of multiple application technologies. This goal is realized by the separation of the SID information model from the SID-based data models. This separation allows multiple data models to be developed and generated from a single information model. For example, an Oracle or DB2 based data model can be developed/generated based on a single information model, or the development/generation of Java interface classes or XML schema.

As a byproduct of achieving the other goals, the management time and cost is reduced. The time and cost associated with introduction of new offerings and technologies is reduced because the impact of these changes is minimized. The time and cost to integrate applications is reduced by having a standard information vocabulary, framework, and models. The time and cost to develop data models is reduced by employing a single information model upon which the data models are based.

The SID's Structure

The SID information model represents a logical view of things of interest (entities) to an enterprise such as customer, location and network element, and relationships (associations) between these things, such as a network element is situated at a location. Entities are further characterized by facts (attributes) that describe them and behavior (operations) that describe how the entities work. The SID as a data model represents a physical implementation of the SID logical view of things.

The SID links distributed and diverse information into a common structure and represents a holistic framework for controlling distributed problem solving. The SID supports interoperability across organizational, corporate, and regulatory boundaries.

The SID has been structured to allow for federation of information. The organization of the SID framework enables a "divide and conquer" approach to problem solving. For example, information that characterizes network resources is organized into a single domain (Resource) within the SID. The use of modeling patterns to capture common structures and relationships makes the SID inherently extensible.

The SID provides a standard way of structuring, defining, and implementing information and behavior. It provides consistent, common terminology and allows reuse of information technology investment. The SID provides a single information model from which information-technology-specific data models, such as Oracle, and DB2, can be derived.

Concepts key to the SID's structure are:

- Business Entity - something of interest to the business that may be tangible things (such as a Customer), active things (such as a Customer Order), or conceptual things (such as a Customer Account). Business entities are characterized by attributes and participate in relationships with other business entities. Business entity instances typically move through a well-defined life cycle
- Aggregate Business Entity (ABE) –a well-defined set of information that characterizes a highly cohesive set of business entities that are loosely coupled with entities in other ABEs
- Domain - a collection of Aggregate Business Entities associated with a specific management area. Domains that make up the SID Framework are consistent with eTOM level 0 concepts.

Similar to the Business Process Framework's (eTOM) framework that provides a decomposition of business processes, the SID is organized into a framework that represents a decomposition of information. The first level of decomposition represents large areas, called domains, which represent key concepts of interest to every information and communications enterprise. The domains include concepts such as Market/Sales, Product, Customer, Service, Resource, Supplier/Partner, and Enterprise as shown in Figure 1.01 – SID Domains. In the figure SID domains are superimposed on the eTOM Strategy Infrastructure & Product and the Operations processes to show that the SID supports both process areas, as well as Enterprise processes. One difference from the Business Process Framework is that the Information Framework domains, Market/Sales, Product, and Customer, are shown as separate horizontals.

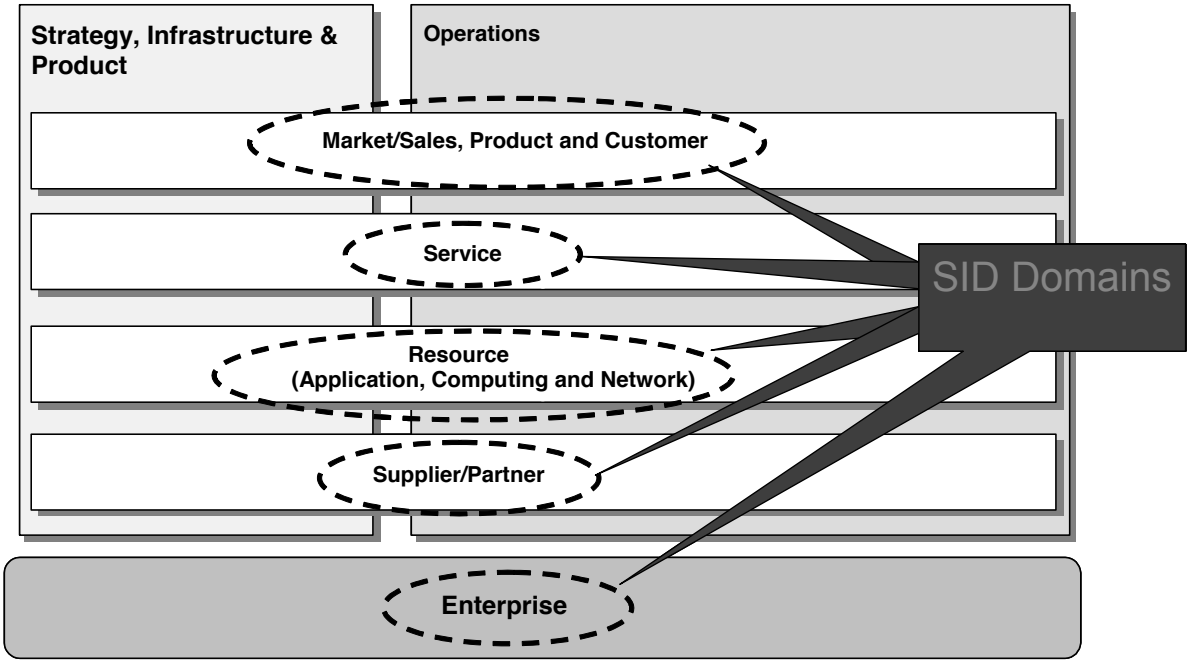

Figure 1.01- SID Domains

Each domain is further decomposed into cohesive collections of entities that characterize the domain. These collections of entities are called Aggregate Business Entities (ABEs). ABEs include such things of interest as Product Offerings, Service Specifications, and Supplier/Partner Orders and represent the first level of ABEs.

The source of the framework is based on previous work of another team in addition to analysis that was performed on an enterprise-wide information model. Previous work by the TM Forum Systems Integration Map team included an object oriented framework upon which the Information Framework is based. Development of the framework included performing affinity analysis on an enterprise-wide information model. Affinity analysis measures the degree to which entities are related and how they are used by processes, the result of which is cohesive groups (clusters) of entities that became the ABEs shown in Figure 1.02 – SID Level 1 ABEs. It is interesting to note that while the SID Framework is presented as a top-down decomposition, it was actually developed from the bottom up!

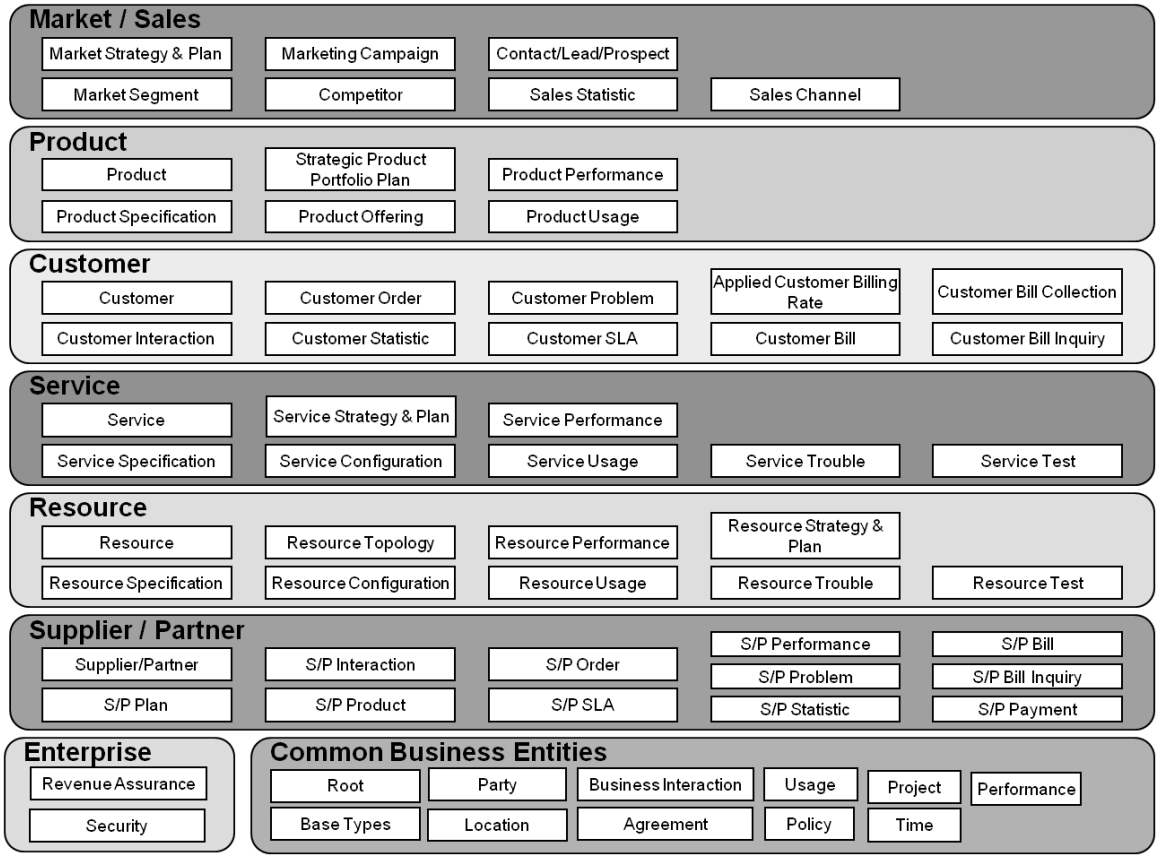

Figure 1.02 – SID Level 1 ABEs

Referring back to what affinity analysis measures, a direct relationship between these ABEs and Business Process Framework Level 2 core processes can be seen. For example, Customer Order entities contained within the Customer Order ABE are the focus of the Order Handling process; Service is the focus of the Service Configuration & Activation process; Service Performance is the focus of the Service Quality Management process. Additional details about this relationship and the relationship between the four Frameworks will be presented later in this book.

Note that an additional domain, Common Business Entities, which is not present as a concept within the eTOM, has been added. This domain contains ABEs that could be placed in two or more domains. One reason for the addition of this domain is that the Information Framework is a non-redundant decomposition of information, just as the eTOM represents a non-redundant decomposition of processes.

This domain also holds generalized ABEs, such as Usage and Performance. Many ABEs, such as Product, Service, and Resource Usage, have many entities in common. Rather than model these common entities redundantly in each ABE, they are modeled within a generalized ABE with the entities in the other domain-specific ABEs inheriting from the generalization.

Not all ABEs have been developed. For example, none of the Strategy & Plan ABEs have been developed as information associated with these ABEs is typically developed by ad hoc processes whose information requirements would be difficult to consistently define. Additionally, applications that support these ABEs are typically not part of an enterprise's suite of applications. The Supplier/Partner domain has not been developed, but other ABEs could be used if a SID implementer wanted to do so. For example, S/P Performance could leverage the Performance ABE. Also, the Product Offering entity has been related to the Party Role entity and could be used to support S/P Product requirements. And, during the writing of this book, the Problem/Trouble and Test ABEs are currently under development by the TM Forum Interface Program; the Configuration ABEs are work in process for the SID team.

Each ABE, based on its complexity, may further decompose into one or more subsequent levels of ABEs. The subsequent levels of ABEs possess the same characteristics of level 1 ABEs, in that they represent a cohesive collection of business entities. For example, Figure 1.03– SID Level 2 ABEs shows how a number of ABEs within the Market/Sales domain decompose into a second level of ABEs.

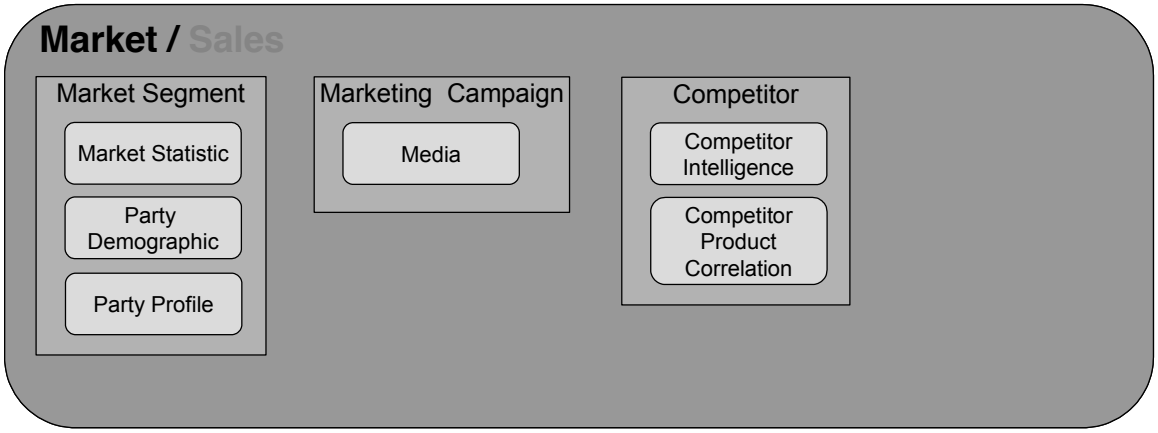

Figure 1.03 – SID Level 2 ABEs

In some cases, ABEs below the first level can be found by analyzing groupings of entities. This represents a "bottom-up" form of ABE discovery.

The SID UML Model

At some point in the decomposition or bottom-up discovery process the lowest level is found. This lowest level contains a group of closely related entities that define the ABE as shown in Figure 1.04 – Customer UML Model for the current Customer ABE. These entities are represented by a Unified Modeling Language (UML) diagram as shown in the figure.

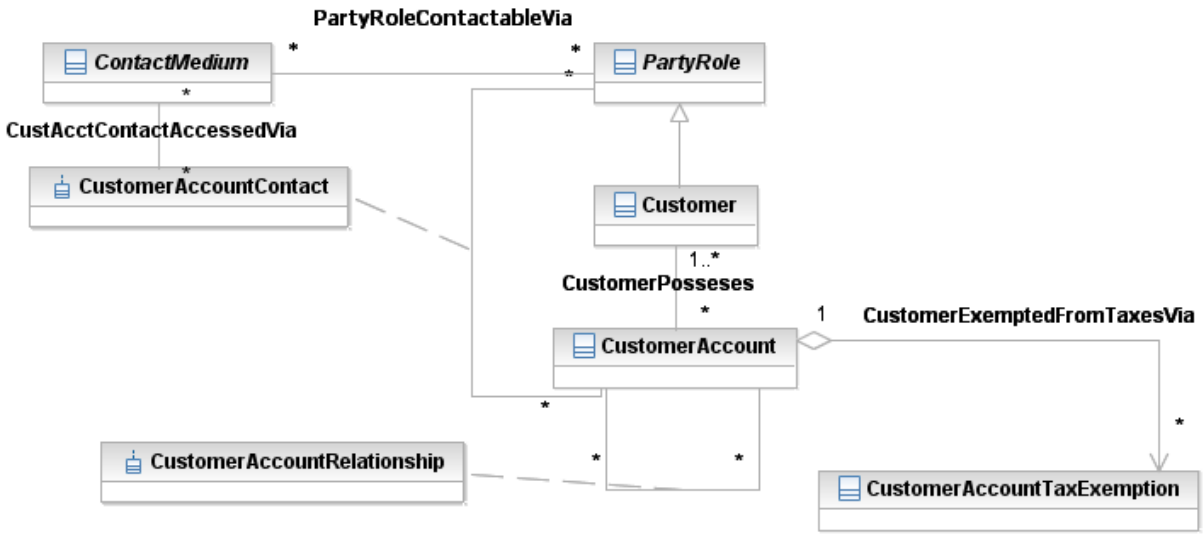

Figure 1.04 – Customer UML Model

UML classes are used to represent SID business entities, such as those shown in the figure. UML attributes and associations are used to represent the corresponding SID concepts. Associations are sometimes referred to as relationships or business relationships. Entities that represent a further refinement of an entity, such as a logical resource or a physical resource are represented by UML subclasses.

Each developed ABE is documented in a SID addendum. The addenda are organized by SID domain. Each addendum contains the design approach, rational for why the entities were modeled in a certain way if there were alternative modeling approaches, and use cases. These enable the model to be better understood and extended to model project-specific entities. Each addendum also includes references to other sources that provide more information about key concepts. Also included is a data dictionary that provides business definitions for all entities and their attributes, as well as UML models. Alias and cross-references to other industry models and concepts that were used as sources for the SID model are included.

Extensibility of the SID

The Information Framework can be viewed as a framework of frameworks, where each domain and each ABE are smaller frameworks within the overall Information Framework. The reason they can be viewed this way is the fact that the entire framework is based on the results of affinity analysis.

What this (framework of frameworks), and the fact that it is enterprise-wide**,** enables is the ability to explicitly add domain or ABE specific extensions, that represent new types of ABEs/entities**.** This facilitates the addition of components to the framework without major impact to other areas of the framework. This is a necessary characteristic of an extensible information framework and its associated model.

Explicit extensions can be added to the Network ABE and the Party Role ABE as new types of entities. New types of entities are typically added as subclasses of existing entities, such as the examples shown in Figure 1.05 – SID as a Framework of Frameworks. This technique is sometimes referred to as the "blade" concept, similar to adding a blade (new capability or functionality) to a network element. In some cases based on the amount of information associated with the new type, such as the NGN Access Network, a single entity is added. In other cases, the information associated with the new type of entity may require the addition of a new ABE, such as Competitor in the figure. In either case, the new type inherits all the attributes and associations of the existing entity.

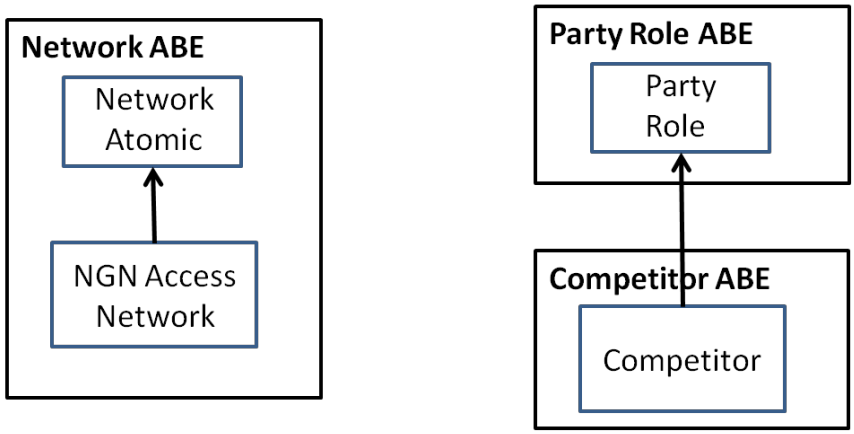

Figure 1.05 – SID as a Framework of Frameworks

Sub-typing (sub-classing) is one of the two ways that the Information Framework provides an extensible information model. The other way is the use of modeling patterns. These patterns were chosen to provide either dynamic or explicit extensibility of the Information Framework to achieve the goals of TM Forum Frameworx. Minimizing changes to an enterprise's information architecture and the applications which use the architecture means that applications are not barriers to an enterprise's introduction of new product offerings, new types of services, new types of resources, and so forth.

There are a number of other benefits to employing modeling patterns. There are quite a few patterns contained in a number of books and papers that could be used to model information; probably at least 50. Using a small number that were chosen because they provide built-in extensibility does represent a consistent approach that makes the model easier to understand. How consistent and easy to understand would a model be that used 10 or more patterns? And, the patterns should be used when extending the SID.

The five patterns used in the SID are:

- EntitySpecification/Entity
- Composite/Atomic
- Entity/EntityRole
- Business Interaction
- CharacteristicSpecification/CharacteristicValue.

Note that these patterns can be used in combination and that there may be cases when none of the patterns are used to model an entity or group of entities in the framework. They are also described in Using the SID (addendum GB922-U).

The following provides a short description of the business requirement satisfied by the pattern:

- EntitySpecification/Entity – many entities are described by specifications, which take the form of paper spec sheets or links to specifications that can be found when shopping on the web. The facts (attributes) for a specification, such as weight, dimensions, color, and so forth, are common to all instances of the entity related to the specification.
- Composite/Atomic – entities are often grouped together. For example, bundles of product offerings, network elements composed of physical and logical resources.
- Entity/EntityRole – an entity can often play many roles, while retaining a basic set of facts about the entity in general. For example, an individual can play the role of customer and employee, but the individual's name and address don't change from role to role.
- Business Interaction – as an enterprise carries out its mission, it interacts with individuals and organizations in a number of ways. This generalization pattern models the characteristics (entities, attributes, and associations) common to most interactions. As such it allows new interactions, such as customer orders, late payment notices, and so forth, to be easily added to the Information Framework.
- CharacteristicSpecification/CharacteristicValue – attempting to model every attribute for new and/or existing entities or attributes associated with new types (specifications) of entities is a near impossible task. This pattern enables new attributes to be dynamically defined, eliminating the need to explicitly model them.

The Information Framework program is considering the addition of two patterns. One, a temporal pattern, can be used to provide a time dimension to the relationship between entities. It has been approved by the team, but not yet applied in the framework. The second is a Version pattern that can be used to expand the basic versioning information, such as validFor attributes that represent from and to effective dates/times for an entity.

A generalized UML view of the five patterns can be found in GB922-U – Users Guide.

The fifth pattern is the CharacteristicSpecification/Characteristic pattern. Some modeler's call this the Attribute pattern, since it is used to enable the dynamic addition of attributes that describe an entity without changing the model.

This pattern is often described as enabling dynamic sub-classing of an entity.

It is very useful in the Product, Service, and Resource domains (to name a few!) because often with the introduction of a new type (instance of a specification) of product, service, or resource, there are additional attributes (characteristics) that represent properties and/or features not supported by the current information architecture.

The CharatcteristicSpecification/Characteristic pattern is often used with the EntitySpecification/Entity and Composite/Atomic patterns to enable near complete freedom (dynamic extensibility) in defining any type of entity and its attributes together with various ways to group the entities.

It's about choices. There is nothing wrong with using the "blade" concept to explicitly model new types of entities (specifications and entities). The considerations described below should help make the choice.

Dynamic attributes are those that are not known until a new information requirement arises, such as a new Product Specification, is made known. Or, when a new Product Specification is just created and the attributes may be in a state of flux. Informational attributes are attributes that only convey some type of information, such as height, weight, color, and so forth, and are not typically used in an application's logic.

Also, note that it is often convenient to start out using explicit modeling as this is a way to document the attributes and their properties. Information modelers often refer to this type of modeling as the construction of a business object model and keep a historical copy of it to be used when populating instances of characteristic specification entities. Note that the explicitly modeled entities would not be present in the information model if characteristics are chosen to support the entity and its attributes.

These considerations are further discussed in Chapter 3 – Implementing Applied SID Modeling Patterns.

Chapter 2 – Implementing SID Patterns

This chapter reviews the current techniques for implementing class hierarchies patterns and other patterns that were described in an earlier book, *Getting Started with the SID* and that are taught in the SID Modeler's workshop. The techniques regarding patterns that employ class hierarchies described here also apply to any area of the SID where there are several levels in a class hierarchy, such as in the SID Resource domain. It provides guidance on an implementation technique to choose and considerations that must be taken into account based on the chosen technique.

Special Note: The techniques described here act on the SID information UML model. It is suggested that, if possible, a database design tool that transforms the logical information model (SID UML) to a logical data model (as a first step towards implementation) be employed. A tool such as this should support the specification of transformations described in this chapter that are automatically performed when transforming the logical data model to the physical data model. Many of the considerations described in this chapter that must be taken into account when transforming the SID UML are then not applicable. These will be pointed out at the point where they are discussed in this chapter. The next chapter provides more detail on using the SID as the basis for database design.

Class Hierarchy Implementation Techniques

In some cases, such as when the SID is to be used as starting point for a physical data base, the logical perspective of the SID is modified to improve performance. In other cases, such as SID-based interfaces, the physical perspective may be generated or developed directly from the SID model as-is or from a subset of the SID model, without employing any of the techniques presented in this section. However, these techniques can also be employed to the SID before interfaces are generated or developed. The considerations presented in this chapter will assist in making the decision to expose the SID as-is or after these techniques are employed. These techniques are not unique to the SID, but can be applied to any information model that contains class hierarchies.

A SID implementer must balance impact of new releases of the SID with practicalities of implementation. Changes made to the SID when transforming the logical perspective to the physical perspective may have to be reconciled manually when adopting new versions of the SID as no comprehensive tools exist today that provide automated reconciliation assistance.

Presented here are techniques that can be employed when moving to the physical perspective. The use of these techniques minimizes the impact of adopting new versions of the SID, while supporting improved performance and a more consolidated, simplified view of the SID. The techniques also ensure that the logical perspective of the SID can be exposed via interfaces when employing the SID as part of an integration framework.

The techniques include:

- Consolidating entities from the "top" down
- Consolidating entities form the "bottom" up
- Consolidating entities from the "middle" - top down and bottom up
- Consolidating entities using a "type" attribute.

The first technique is used to consolidate a SID class hierarchy by explicitly moving attributes and relationships from abstract super-classes to concrete subclasses. This is referred to as the "top" down consolidation technique. Figure 2.01 - Consolidating Sub-Classes – Top Down - Before shows a class hierarchy from the Business Interaction ABE before consolidation. Note: The examples presented here do not always show repositioning of relationships, but the same technique applies.

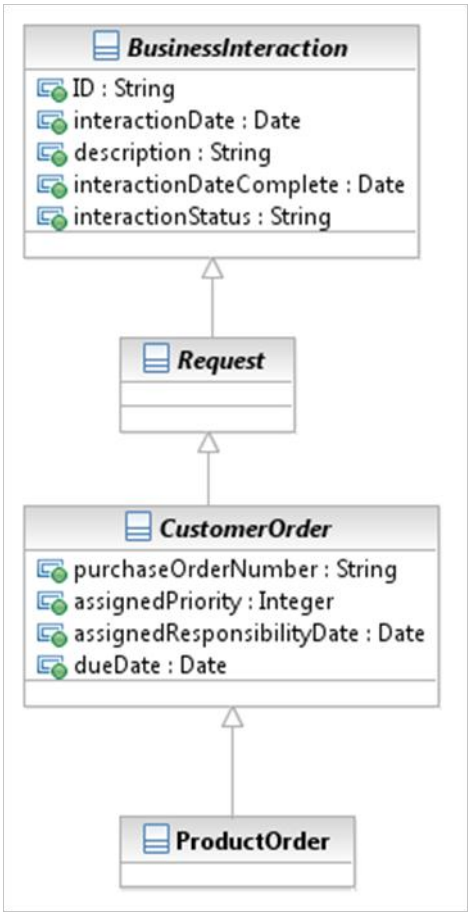

Figure 2.01 – Consolidating Sub-Classes – Top Down - Before

In this example, all attributes from the super-classes above the ProductOrder entity are moved to it as shown in Figure 2.02- Consolidating Sub-Classes – Top Down - After. This simplifies the implementation view without compromising the structure of the SID, as the super-classes of ProductOrder can be constructed from it if desired for exposure via an interface.

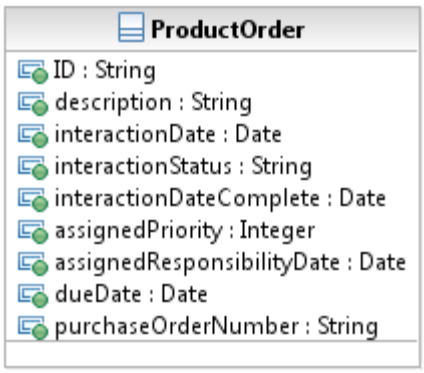

Figure 2.02 – Consolidating Sub-Classes – Top Down - After

The second technique is similar to the first, but the consolidation involves merging the subclasses into the super-class. Typical applications of this technique remove the composite/atomic sub-classes for a business entity or consolidate an extension to an entity into the entity being extended. Composite/atomic sub-classes are employed throughout the SID model to represent the fact that a single instance of an entity can be comprised of other instances of the same entity. For example, a bundled ProductSpecification is comprised of other instances of ProductSpecification. It is not unusual to apply this technique if entities can be related in a number of other ways in addition to a composite/atomic association, such as "mutually exclusive", "superseded by", and so forth. Figure 2.03 - Consolidating Sub-Classes – Bottom Up - Before shows a class hierarchy from the Product Specification ABE before consolidation.

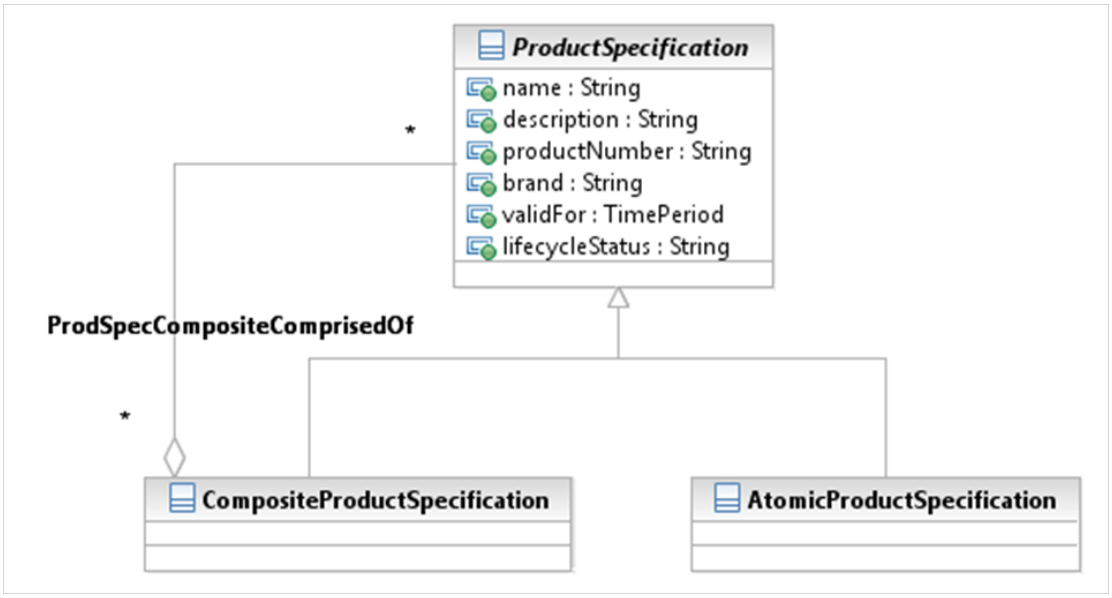

Figure 2.03 – Consolidating Sub-Classes – Bottom Up - Before

In this example, all attributes and relationships from the two ProductSpecification subclasses are moved to it as shown in Figure 2.04 - Consolidating Sub-Classes – Bottom Up - After. This simplifies the implementation view without compromising the structure of the SID, as the subclasses of ProductSpecification can be exposed via an interface if necessary and then later consolidated within an application. The manner in which this is accomplished is that the composite/atomic relationship between an instance of ProductSpecification and two or more other instances of ProductSpecification become another type of relationship maintained by the ProductSpecificationRelationship entity shown in Figure 2.04.

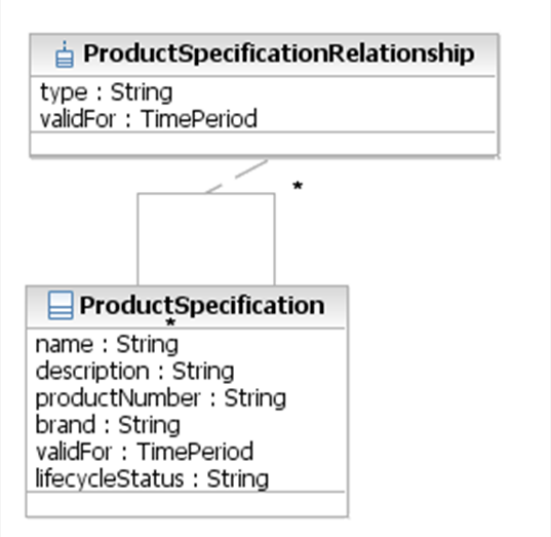

Figure 2.04 – Consolidating Sub-Classes – Bottom Up - After

The third technique, top-down bottom up, represents a combination of the first two techniques. This technique can be combined with the first technique when there is also a desire to consolidate a pattern such as the composite/atomic pattern. This technique involves moving attributes and relationships from one or more super-classes and from one or more subclasses to one or more intermediate sub-classes. Figure 2.05 - – Consolidating Sub-Classes – Top Down – Bottom Up - Before shows a class hierarchy from the Product Offering Price ABE before consolidation.

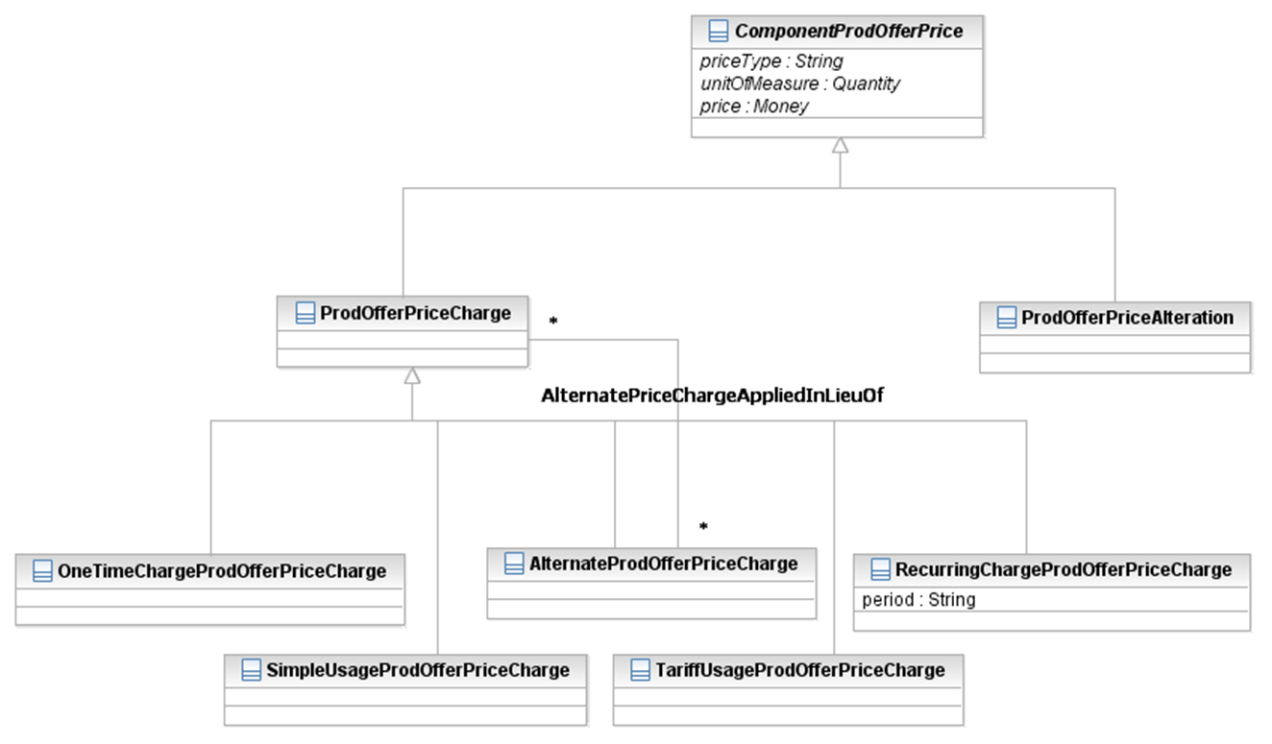

Figure 2.05 – Consolidating Sub-Classes – Top Down – Bottom Up - Before

In this example, a SID modeler wants to retain the visibility of two types of ComponentProdOfferPrice, while removing what are mainly illustrative examples (the set of concrete subclasses is not exhaustive) of the types of ProdOfferPriceCharge and ProdOfferPriceAlteration. This simplifies the implementation view without compromising the structure of the SID, as the ComponentProdOfferPrice and its subclasses can be exposed via an interface if necessary and then later consolidated within an application. This type of consolidation for the example is shown in Figure 2. 06 – Consolidating Sub-Classes – Top Down – Bottom Up - After.

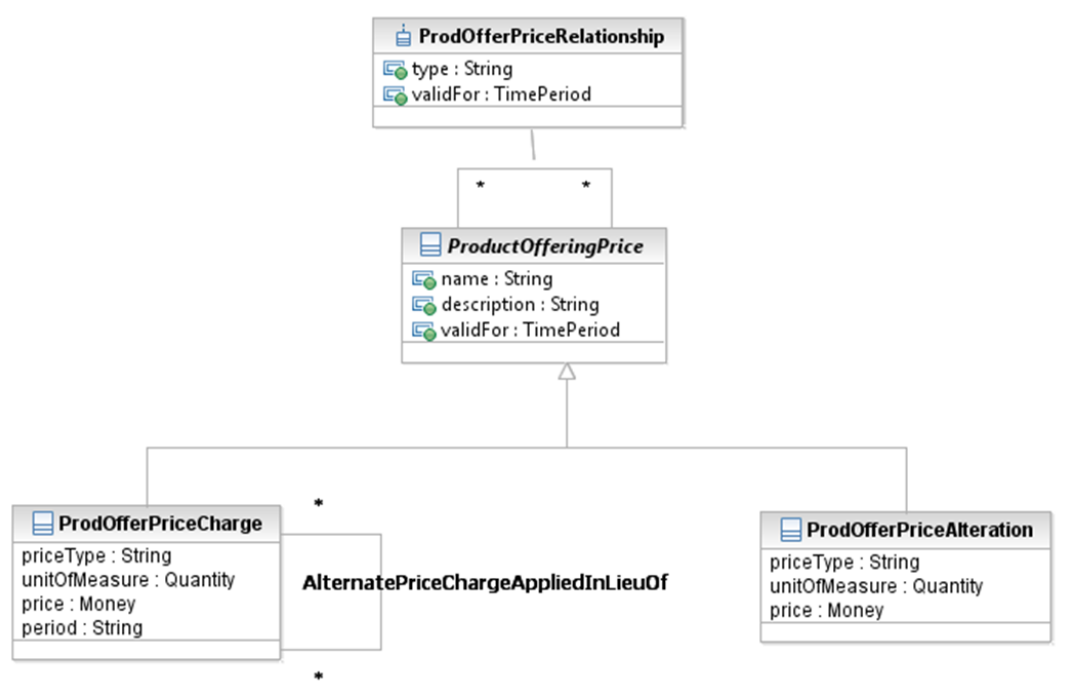

Figure 2.06 – Consolidating Sub-Classes – Top Down – Bottom Up - After

The fourth technique, typing, is a variant of the bottom up technique. This technique involves moving attributes and relationships from one or more subclasses and to their intermediate super-class and by adding an attribute that indicates what "type" of subclass is represented by an instance of the super-class. This technique can be applied when the subclasses do not represent the entire set of possible types of the super-class or there is a very small if any number of attributes in the subclasses. Figure 2.07 – Consolidating Sub-Classes – Typing - Before shows a class hierarchy from the Customer Order ABE before consolidation.

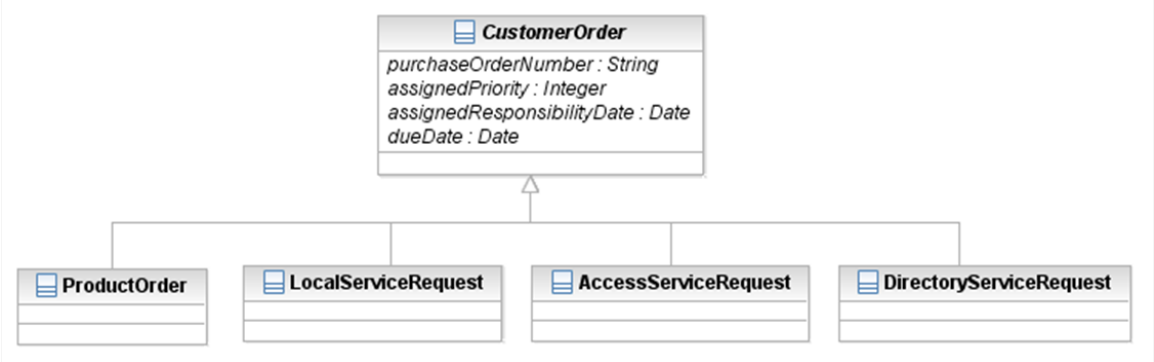

Figure 2.07 – Consolidating Sub-Classes – Typing - Before

In this example, a SID modeler is not interested in retaining explicit visibility into the four of the possible types of CustomerOrders. This technique simplifies the implementation view without compromising the structure of the SID, as the various types of CustomerOrders can be exposed via an interface if necessary and then later consolidated within an application. The resulting consolidation for the example is shown in Figure 2.08 - Consolidating Sub-Classes – Typing - After.

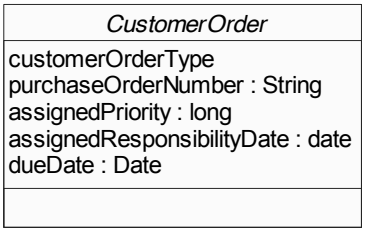

Figure 2.08– Consolidating Sub-Classes – Typing - After

The next chapter describes more techniques that can be used in transforming the SID logical perspective into the SID physical perspective.

Class Hierarchy Technique Guidance

When choosing the class hierarchy implementation technique, there are a number of considerations that can be taken into account to choose the technique.

Application Boundaries

Application boundaries can provide guidance when choosing the first or second option. For example, the first option (top-down) may be chosen if there are separate applications that deal with different types of performance, such as product, service, and resource performance. Figure 2.09 – Performance Management Class Hierarchy shows a generalized Performance entity and its subclasses. Note that the entities related to the specifications shown in the figure would also be transformed from the top down.

Figure 2.09 – Performance Management Class Hierarchy

The Figure 2.10 – Transformed Performance Management Class Hierarchy shows the resultant model after the transformation.

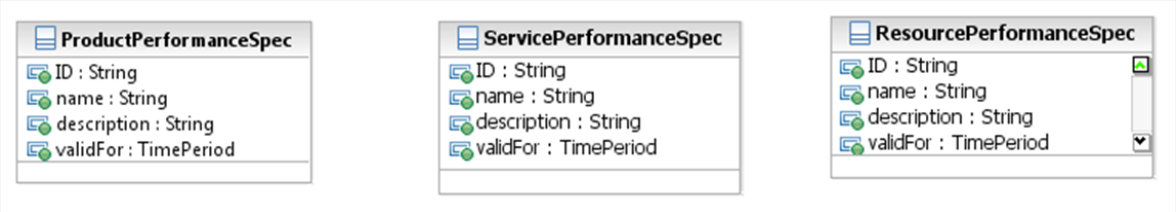

Figure 2.10 – Transformed Performance Management Class Hierarchy

Another example of application boundaries providing some guidance is where there are two performance applications. One application is part of a Customer Relationship Management (CRM) solution that supports Product performance. Another application is part of an Inventory, or Infrastructure, application that supports Service and Resource performance. In this case, the top-down option may be also chosen that results in the transformation shown in Figure 2.11 – Product and Inventory Performance. In this example, the top-down option results in two entities as shown.

Figure 2.11 – Product and Inventory Performance

In the figure, an inventoryType attribute has been added to indicate whether the specification is a Service or Resource performance specification. Alternatively, the association (not shown) to ServiceSpecification or ResourceSpecification could be used to determine the type of performance specification.

In another case where application boundaries can assist in determining the technique to use, consider a Mediation application. The various types of usage (product, service, and resource) may just represent states, or stages, in the life cycle of a usage record. In this case the second option (bottom-up) may be chosen, with the associations to product, service, and resource representing the states in the life of usage as it is guided to a product. Figure 2.12 – Usage Class Hierarchy depicts the usage model before transformation. It is also important to note that "raw" Resource usage data typically includes data about Product and Service usage, which may further influence choosing this transformation option.

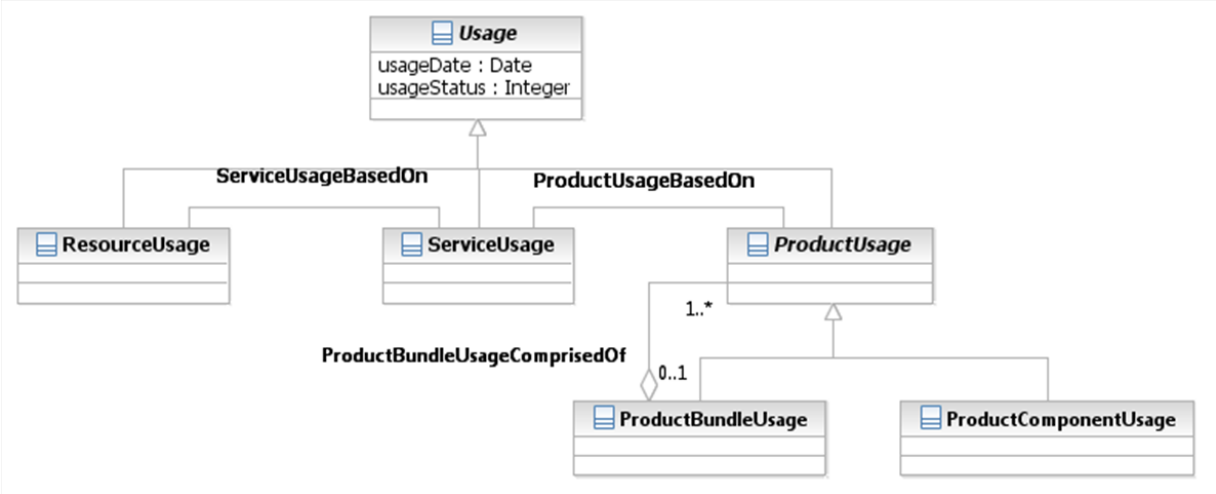

Figure 2.12– Usage Class Hierarchy

Figure 2.13 – Transformed Usage Class Hierarchy shows the hierarchy after transformation, including the associations with Product, Service, and Resource, which were also transformed from the bottom up. The Composite/Atomic ProductUsage entities were also transformed from the bottom up. In the transformed model it may be advantageous to either add a usageType attribute as discussed earlier.

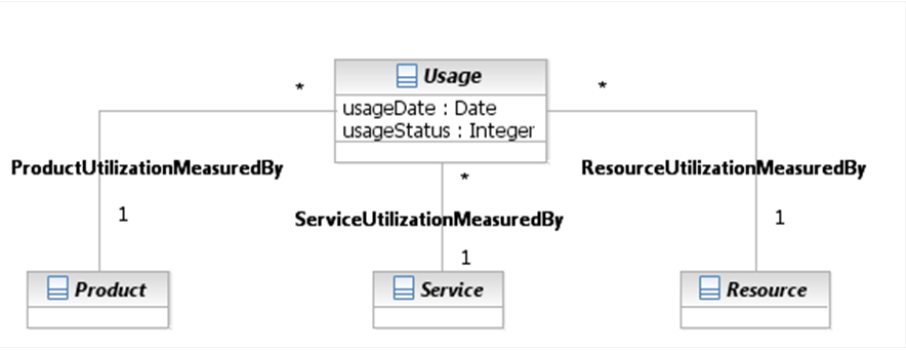

Figure 2.13 – Transformed Usage Class Hierarchy

Other Technique Guidance

It should not always be assumed that individuals are familiar with the SID. Often, in the case of B2B interfaces an interacting company is not familiar with the SID. For example the concepts of Business Interaction or other abstract classes, such as Party may not be known or may not be well-known. In these cases, option one, where classes are transformed from the top down may the implementation choice, as shown in Figure 2.08 – Consolidating Sub-Classes – Typing – After.

Often in this case, attribute names, such as interactionDate may also be changed to a more common term, such as customerOrderDate to facilitate understanding. A later chapter of this book discusses the implications of this type of transformation, where maintaining conformance should be considered.

Class Hierarchy Implementation Considerations

There are implementation considerations that should be taken into account for each option, some of which are included here. There are certainly others that may be taken into account based on a SID implementer's experience. These considerations do assume a relational database implementation.

When a new version of the SID is to be implemented a full or partial transformation may be required using the database design tool that supports automatic transformation. Alternatively, the changes may be manually introduced into the logical data model.

Considerations Applicable to All Techniques

There are integration cost considerations that must be taken into account. All of these techniques change the basic structure of the information model. Interfaces that expose these transformed structures often must be mapped back to the information model, if the information model is used as-is within an integration framework. Additionally, the changes made must be explained. Both of these add to the cost of integration.

All these options also present a small problem as any changes would have to be re-applied manually if the use of a new SID version containing updated entities involved in the transformation is desired. This consideration is not applicable if a data base design tool is used to automatically transform the SID to a logical data model.

An important consideration is that each option hides details that are shown by the subclasses. For example in Figure 2.04 – Consolidating Sub-Classes – Bottom Up - After the fact that "groups" (CompositeProductSpecification) can be defined is now hidden in a value that the attribute "type" (in ProductSpecificationRelationship), which is used to indicate that groups, or bundles, of ProductSpecifications can be defined. Just relying on this can result in an application not supporting the definition of groups. Figures that represent the entities before and after transformation are often kept and attached to application documentation to reduce the probability of this requirement not being supported. A database design tool could provide this before and after view with the unchanged SID UML information model representing the "before" and the logical data model representing the "after".

Considerations for Specific Techniques

Application of the top-down and middle-up-and-down techniques result in duplication of attributes. Therefore, if the properties of the attribute change, care must be taken to ensure they are change across all the tables. Also, the recursive association on business interaction is no longer usable to related different types of business interactions. They must be explicitly modeled. This also applies to any other entity that is transformed from the top-down or middle-down. A database design tool that supports automatic transformation negates all these considerations.

The bottom-up, middle-up-and-down, and "type"-attribute techniques result in many attributes and associations that were "required" in a subclass to become "optional" in the super-class. The conditions under which the attributes and associations are still required must be carefully documented. Some database design tools will add a type attribute or a type entity automatically for these types of transformations, if they are specified within the tool.

Considerations associated with the middle-up-and-down option include those of both the top-down and bottom-up techniques.

When applying the top-down technique, it should be noted that this does not have to be applied to all subclasses. For example, the PartyRole entity may be transformed top-down into Customer and Supplier because there is no interest in implementing an enterprise wide Party model. However, it may be desirable to consolidate some of the other roles that can be played by individuals or organizations that do not have a large number of attributes or related entities into a single set of entities that represent the Party and the PartyRoles played and their related entities. The next chapter provides an example of the top-down transformation of PartyRole into Customer.

There is an implementation consideration that should be taken into account when transforming any application of the Composite/Atomic pattern using the bottom-up technique. There is often a concern about changing a composite instance to an atomic instance or an atomic instance to a composite instance, if the pattern is implemented as-is. This should not be an issue from an implementation perspective, because there should be application functionality in place to accommodate these requirements.

For example, changing a BundledProductOffering to a SimpleProductOffering, must first remove all instances of SimpleProductOfferings from the BundledProductOffering, even if the bottom-up technique has been employed; and application functionality should be in place to copy the instance of the BundledProductOffering to an instance of a SimpleProductOffering.

Another consideration here is that the original entity's "type" should not be changed so that a historical record of its type is maintained, particularly if there are instances of Product associated to it. Changing the type can lead to problems, particularly if instances of SimpleProductOfferings are actually removed (deleted) from a BundledProductOffering as described in the example above. The Products related to the SimpleProductOfferings would be orphaned, possible creating referential integrity problems, or even deleted if the SimpleProductOffering to which they are related are deleted. Rather changing the type, the original entity should be inactivated using the endDateTime and a new instance of the entity reflecting the type change should be created.

The next two figures show an example of the bottom-up technique that can lead to multiple issues. The first figure shows part of the as-is Service class hierarchy. The second figure shows the resulting transformation.

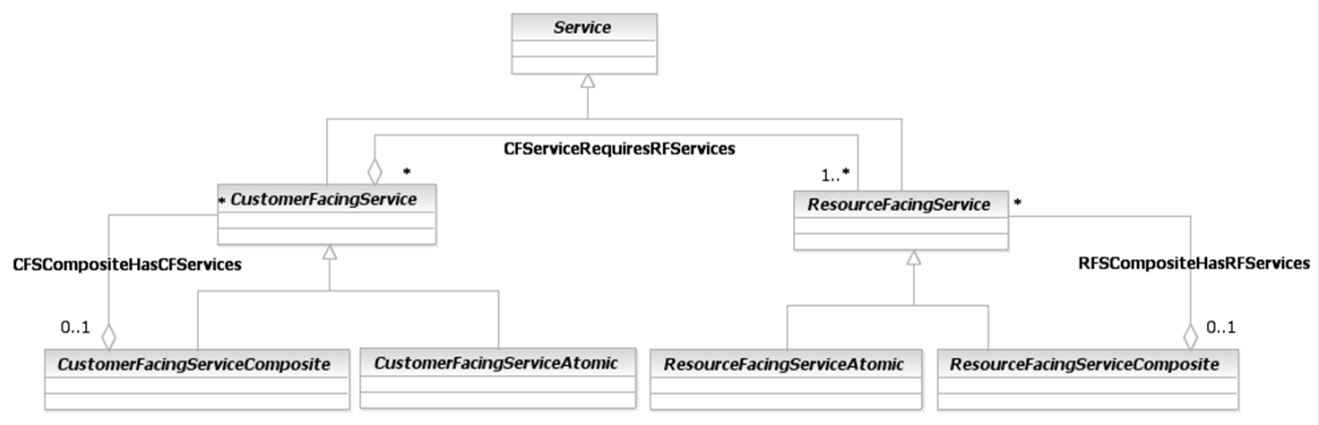

Figure 2.14 - Service Hierarchy - Bottom Up - Before

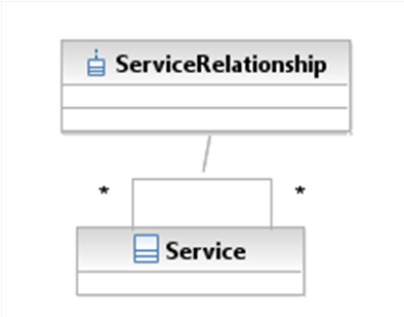

Figure 2.15 – Service Hierarchy – Bottom Up – After

This transformation example would allow a CustomerFacingService (CFS) to be required by an ResourceFacingService (RFS) and a CFS to be composed of RFS and vice versa. A possible solution to these problems would to stop bottom up transformation at CFS and RFS, which would include transforming the two aggregation associations to two recursive associations. If not some form of logic, and possibly a "rules" entity, would need to be included in an application that supported the full bottom-up transformation. Or a middle up and down transformation could be employed, leaving only CFS and RFS and the applicable associations.

Implementing Other Modeling Patterns

There are also techniques and considerations for implementing the EntitySpecification/Entity, Entity/EntityRole, and the CharacteristicSpecification/CharacteristicValue modeling patterns. Included are guidance on what not to do or what to do, as well as a discussion of application boundary considerations.

EntitySpecification/Entity

It sometimes may appear desirable to collapse an EntitySpecification entity, such as ProductSpecification into the related Entity, such as Product. But this also means collapsing any subclasses and associations, not just the attributes. Here the focus will be on the implications of duplicating attributes from the EntitySpecification in the related Entity.

These implications are called anomalies. There are three:

- Update anomaly
- Insert anomaly
- Delete anomaly

When an attempt is made to modify (update, add, or delete from) instances, undesired side-effects may follow. The examples will use the ProductSpecification and Product entities.

Each instance of Product would contain the ProductSpecification attributes, such as name, description, productNumber, and brand. A change to any of these attributes will need to be applied to multiple instances of Product. If the change is not successful, the brand is updated for some instances but not others. This results in inconsistencies, which means conflicting answers to the question of what this particular ProductSpecification's brand is. This phenomenon is known as an **update anomaly**.

> There are circumstances in which Product attribute values cannot be recorded at all. For example, each instance of a Product contains specification information. This means that attribute values that describe a new instance of ProductSpecification cannot be defined unless there is at least one instance of Product. This phenomenon is known as an **insertion anomaly**.
>
> If the last instance of Product that contains the attributes the describe the associated specification is deleted, then the instance of ProductSpecification is also deleted. All information about the ProductSpecification is lost. This phenomenon is known as a **deletion anomaly**.

These anomalies should be considered before one to many association is collapsed, not just the parts of the model where the EntitySpecification/Entity pattern has been applied.

Entity/EntityRole

There may be a desire to transform the EntityRole application of the Entity/EntityRole pattern to a subclass of the Entity.

Figure 2.16 - Party and PartyRole shows the Party and PartyRole application of this pattern.

Figure 2.16 – Party and PartyRole

Some of the subclasses of PartyRole, such as Customer, ServiceProvider , and Employee, are shown in Figure 2.17 – PartyRoles Transformed Into Party Subclasses.

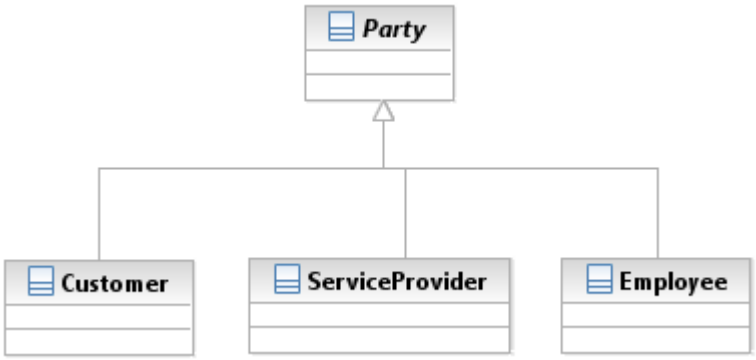

Figure 2.17 – PartyRoles Transformed Into Party Subclasses

With this transformation, a separate instance of Party would have to be created for each role, losing the capability of sharing instances of Party attributes and associations for each role a Party plays and the anomalies discussed earlier reappear. This also ignores the fact Party is already sub-classed. And, the same anomalies that occur when an entity's specification is collapsed into the associated entity await those who follow this path for the Entity/EntityRole patterrn!

If the two entities that make up this pattern are collapsed, it is difficult to support multiple roles and still maintain data integrity. For example, where would you store the name of someone who is both a customer and an employee?

CharacteristicSpecification/CharacteristicValue

This section presents a number of implementation considerations when using the CharacteristicSpecification/CharacteristicValue pattern.

Characteristics are Attributes, not Entities

Keep in mind that characteristics are attributes, not entities. While composite attributes can be defined using this pattern, attributes don't participate in complex associations, they can't be sub-classed, and can't have attributes that describe them.

For example, suppose there is a requirement to add new performance monitoring entity that is related to a party role. This could be added as a composite attribute called "PerformanceMonitoring" with the attributes define it as atomic attributes. But, how is the referential integrity with the associated PartyRole maintained? Atomic attributes could the name of the related entity, PartyRole in this case, and the name of the entity's identifier. However, this is a work-around for modeling the entities and associations and requires an application to do maintain referential integrity that is done by a database. If also does not account for any misspellings in the name of the related entity and the name of the attribute which identifies the entity.

Explicit Modeling Versus Characteristics

When using and implementing the CharacteristicSpecification/CharacteristicValue pattern a number of questions often arise, including

- When should the attributes (characteristics) be modeled explicitly versus using the pattern?
- If characteristics are used, is there anything else that needs to be considered from a modeling perspective?
- Are there any performance issues that should be considered?.

Explicit modeling should take the following considerations into account

- Adding/removing/modifying attributes changes the model
- Attributes are visible
- Attributes stable/well known
- Logic associated with attribute
- Can start with this technique to identify characteristics.

Characteristics should take the following considerations into account

- No changes required to the model when adding/modifying/removing attributes
- Hides attributes
- Dynamic attributes- ones that are not known at the time of constructing the model

- Informational attributes – ones that are not used in logic.

It is about choices. There is nothing wrong with explicitly modeling new types of entities (specifications and entities). The considerations above should help make the choice. Also, note that it is often convenient to start out using explicit modeling as this is a way to document the attributes and their properties. Information modelers often refer to this type of modeling as the construction of a business object model and keep a historical copy of it to be used when populating instances of characteristic specification entities. Note that the explicitly modeled entities would not be present in the information model if characteristics are chosen to support the entity and its attributes.

There are other considerations that need to be taken into account. The Information Framework does not contain entities that support the dynamic design of user interfaces, such as web pages. To more completely support the CharacteristicSpecification/CharacteristicValue pattern the framework should be extended to provide this support. No user wants characteristics to be randomly placed on a user interface! To help model these requirements, think of the properties that are specified when designing a user interface, such as position, label, prompt, length, and so forth.

Another consideration is modeling behavior-related entities so that code associated with a characteristic can easily be added to an application.

There may be other considerations that are specific to a given use of the pattern, but these are ones that are typical to any use of the pattern.

The next two chapters will discuss implementing a single characteristic model from a database perspective. Later chapters will discuss other implementation considerations.

Chapter 3 – SID and Database Design

This chapter describes techniques that can be used when employing a database design tool for implementing the five SID modeling patterns, as well as any part of the SID model that include class hierarchies or entities involved in one to many associations. Included in the chapter are many alternatives that can be considered along with lessons learned that should be taken into account before and after transforming the SID information model to a SID-based data model. The examples in this chapter start with transforming UML-specified entities into Entity-Relationship specified logical data model entities and then to relational physical data model entities. The Data Definition Language (DDL) generated from the physical data models are provided for some transformations.

A Note on SID and Database Design

Using the SID information model as the basis for the design of a database is not all that much different than data base design for any model that contains class hierarchies or modeling patterns similar to those used in the SID. Many, if not all, of the transformations could deferred to database design, if a tool that is used for database design provides functionality to roll up and roll down entities and other transformations described in the previous chapter. For example, the top-down technique can be supported by rolling-down the BusinessInteraction entity and its associations to entities in lower levels of its class hierarchy.

Database Design & Class Hierarchy Patterns

This section focuses on the SID patterns that represent class hierarchies, the Business Interaction pattern and the Composite/Atomic pattern.

General Considerations

As a technology-neutral information model, the "I" component of the SID has never been intended to be implemented as-is. Parts of the SID include multi-level class hierarchies that are not necessarily suitable for one-to-one SID entity to database table implementation.

There are two key implementation issues that are resolved by using the transformation techniques. The first is that if entities in a class hierarchy are implemented as-is, then multiple instances of entities are created. For example, in the BusinessInteraction class-hierarchy were implemented as-is to support the ProductOrder entity, then four instances of it must be created, one in each table, BusinessInteraction, Request, CustomerOrder, and ProductOrder. This may be viewed as creating too many tables to represent one concept in the database.

The second issue deals with the resolution of table joins that would be necessary if a class hierarchy is implemented as-is. In the BusinessInteraction example, four tables would have to be joined to retrieve a single instance of a ProductOrder. This may result in unacceptable database performance in high volume applications, such as those required by a Customer Order Management application.

These two issues also exist for applications of the Composite/Atomic pattern, although few tables would be generated if the applications of this pattern is implemented as-is.

Top Down – Business Interaction Pattern

Described here is a top-down technique that can be used to develop a database that implements the Business Interaction pattern. Since Product Order and the other subclasses of CustomerOrder can be considered examples, this is the use of the top-down technique, not the middle – up & down technique.

Also, keep in mind that this technique can be used to collapse any other class hierarchies in the SID, such the Service or Resource entity hierarchies.

Figure 3.01 – Customer Order Logical Data Model Fragment depicts the BusinessInteraction hierarchy, including CustomerOrder and CustomerOrderItem. Some of the associations that BusinessInteraction and BusinessInteractionItem have with other SID entities are also shown. Not all are shown because the transformation for these is similar to those shown in the diagram.

For those not familiar with the Entity-Relationship diagramming, in the figure and all similar figures shown in this chapter the "o" means optional, the "|" indicates a one multiplicity, the "<" indicates a many multiplicity, and the "  " represents a subtype (subclass). If the reader wants more information about Entity Relationship modeling many tutorials/explanations can be found on the Internet.

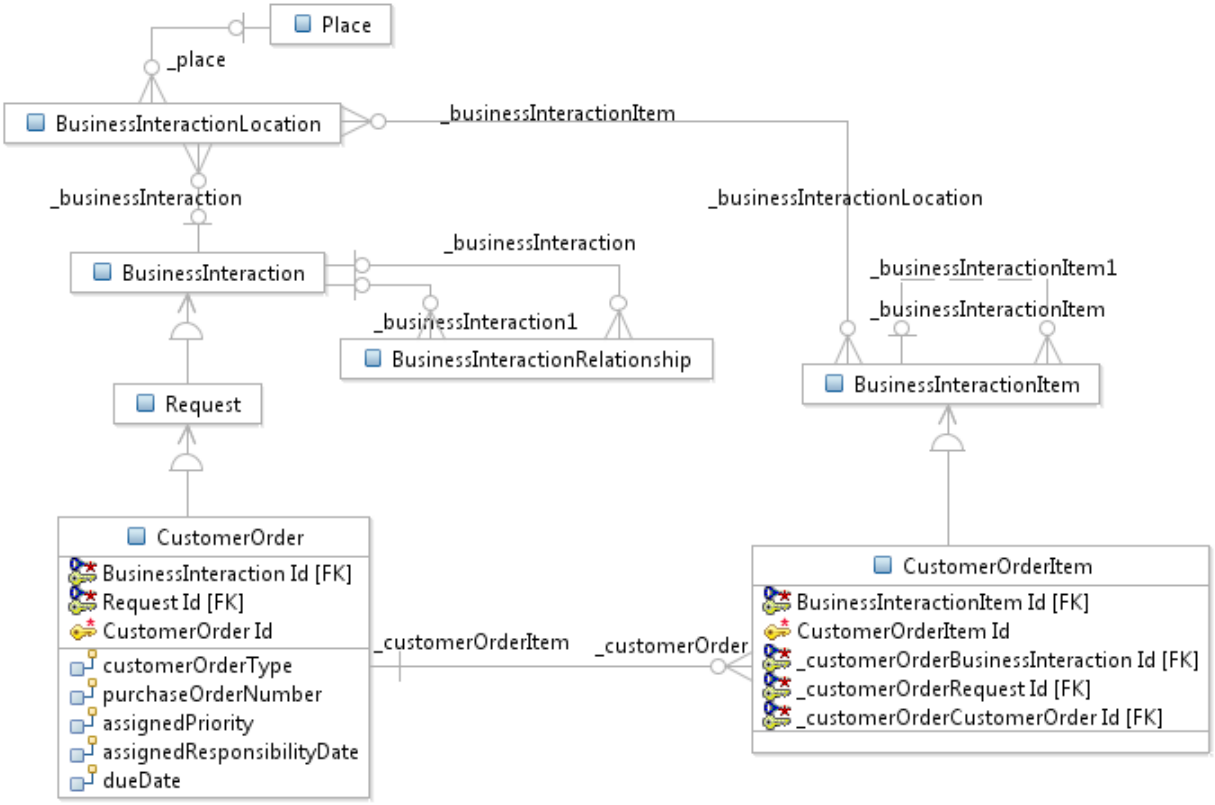

Figure 3.01 – Customer Order Logical Data Model Fragment

Figure 3.02 – Transformed Customer Order Model Fragment shows the results of transforming the logical data model to the physical model. The super-classes BusinessInteraction and Request were rolled down (collapsed) into CustomerOrder, and the superclass BusinessInteractionItem was rolled down into CustomerOrderItem. This was specified in the logical data model. Notice the inherited BusinessInteraction and BusinessInteractionItem attributes. The validFor (datatype is TimePeriod in the Base Types ABE) attribute is missing in BusinessInteractionItem. This will be discussed in a later section of this chapter. Also, an upcoming figure will show some of the inherited associations.

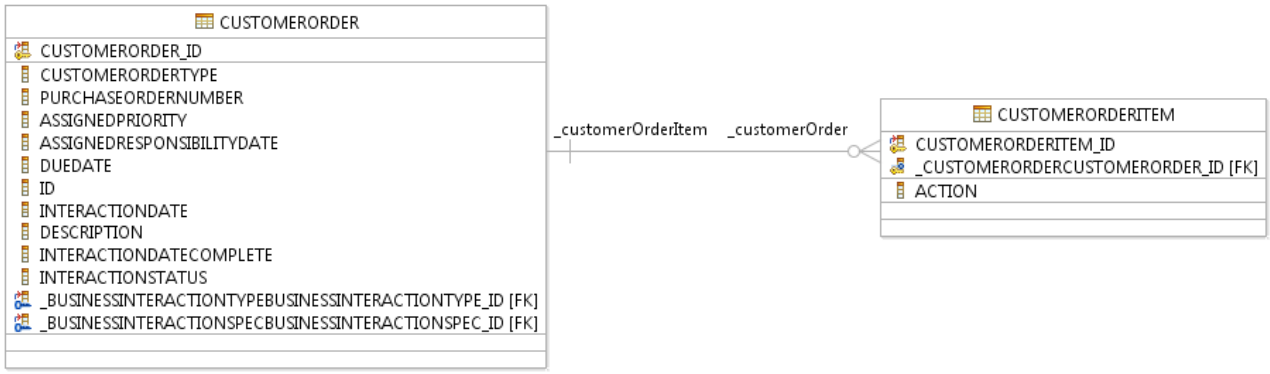

Figure 3.02 – Transformed Customer Order Model Fragment

There a number of changes that were made before and after the transformation to the physical data model. Non-needed keys have been removed. For example, requestID was made non-persistent in the logical model and businessInteractionID was removed in the physical model after the proper foreign keys were generated. Some tools allow this to be done in logical data model, so regeneration of physical data model does not require removal of them.

Keep in mind that this is not unique to the SID. However, it may be necessary to retain some inherited primary key attributes in the logical data model to correctly generate the physical data base foreign key attributes. As with any tool that transforms an information model to various stages of data models, practice with the tool is essential! Knowing the desired end result will impact how the transformations are defined and the number of manual changes that are required at each step in the transformation.

Also, it is up to the implementer to decide if the "interaction" prefix on the attributes inherited from BusinessInteraction should be changed to "customerOrder".

Figure 3.03 – Customer Order Inherited Associations shows some of the inherited associations.

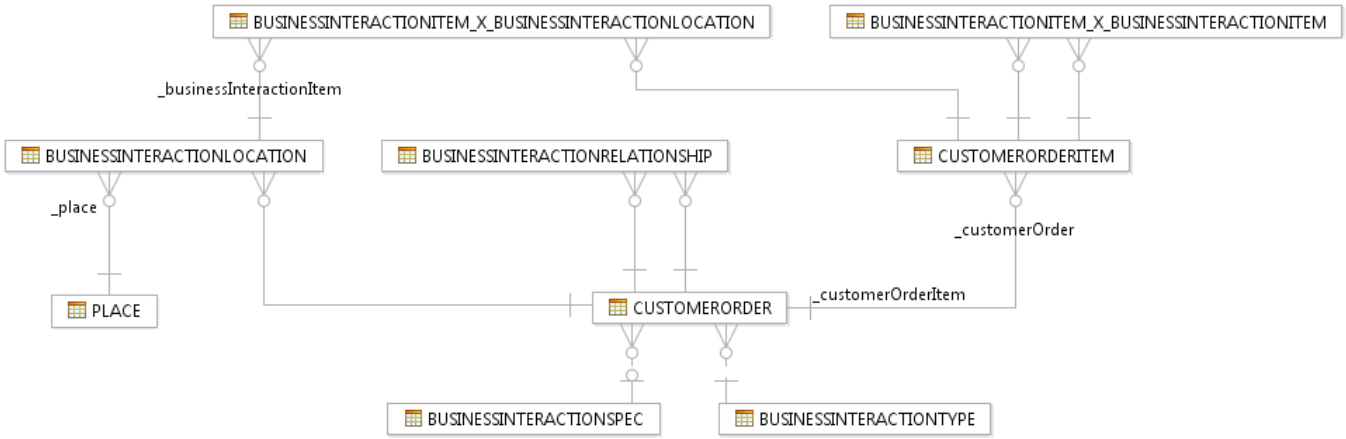

Figure 3.03 – Customer Order Inherited Associations

Notice that the two BusinessInteraction many to many (*-*) associations were transformed into separate entities as part of the transformation. The last section in this chapter describes an approach to handling these types of associations that should be considered before transforming the information model to the logical data model.

It is up to the SID implementer to decide whether to change the name of the BusinessInteraction-focused names. For example, it may be desirable to change BusinessInteraction prefixes to CustomerOrder and BusinessInteractionItem prefixes to CustomerOrderItem. Notice that the BusinessInteractionRelationship and BusinessInteractionItem_X_BusinessInteractionItem entities now only provide for inter-relating CustomerOrder entities. Support for associations across the subclasses of BusinessInteractions is lost but could be manually introduced by adding associations in the logical or physical data model. The last chapter in this book discusses maintaining cross-domain and cross-ABE associations that may cross application boundaries.

Shown next are two extracts from the Data Definition Language (DDL), also call Data Description Language, generated from the physical data model. DDL is the starting point for the generation of physical tables in a database. For those not familiar with Data Definition Language, the internet can be used as a starting point to gain an understanding.

```
CREATE TABLE "Schema"."CUSTOMERORDER" (
            "CUSTOMERORDER_ID" INTEGER NOT NULL GENERATED BY DEFAULT
AS IDENTITY ( START WITH 1 INCREMENT BY 1 MINVALUE 1 MAXVALUE
2147483647 NO CYCLE CACHE 20 NO ORDER ),
            "CUSTOMERORDERTYPE" VARCHAR(32672),
            "PURCHASEORDERNUMBER" VARCHAR(32672),
            "ASSIGNEDPRIORITY" INTEGER,
            "ASSIGNEDRESPONSIBILITYDATE" DATE,
            "DUEDATE" DATE,
            "ID" VARCHAR(32672),
            "INTERACTIONDATE" DATE,
            "DESCRIPTION" VARCHAR(32672),
            "INTERACTIONDATECOMPLETE" DATE,
            "INTERACTIONSTATUS" VARCHAR(32672),
```

```
            "_BUSINESSINTERACTIONTYPEBUSINESSINTERACTIONTYPE_ID"
INTEGER NOT NULL,
            "_BUSINESSINTERACTIONSPECBUSINESSINTERACTIONSPEC_ID"
INTEGER
      )
      DATA CAPTURE NONE;

CREATE TABLE "Schema"."CUSTOMERORDERITEM" (
            "CUSTOMERORDERITEM_ID" INTEGER NOT NULL GENERATED BY
DEFAULT AS IDENTITY ( START WITH 1 INCREMENT BY 1 MINVALUE 1
MAXVALUE 2147483647 NO CYCLE CACHE 20 NO ORDER ),
            "_CUSTOMERORDERCUSTOMERORDER_ID" INTEGER NOT NULL,
            "ACTION" VARCHAR(32672)
      )
      DATA CAPTURE NONE;
```

Note that the length of the VARCHAR (UML String data types) is the default for the tool used. It can be changed in the physical data model or the logical data model depending on which of these (or both) support incremental updates as part of the transformation to them. For example, some tools don't provide for incremental changes to the logical data model when transforming from UML; if this is the case, then the changes can be made in the physical data model, with the hope that transformation to it can be incremental!

The second fragment of the generated Data Definition Language follows:

```
ALTER TABLE "Schema"."CUSTOMERORDER" ADD CONSTRAINT
"CUSTOMERORDER_BUSINESSINTERACTIONSPEC_FK" FOREIGN KEY
      ("_BUSINESSINTERACTIONSPECBUSINESSINTERACTIONSPEC_ID")
      REFERENCES "Schema"."BUSINESSINTERACTIONSPEC"
      ("BUSINESSINTERACTIONSPEC_ID");

ALTER TABLE "Schema"."CUSTOMERORDER" ADD CONSTRAINT
"CUSTOMERORDER_BUSINESSINTERACTIONTYPE_FK" FOREIGN KEY
      ("_BUSINESSINTERACTIONTYPEBUSINESSINTERACTIONTYPE_ID")
      REFERENCES "Schema"."BUSINESSINTERACTIONTYPE"
      ("BUSINESSINTERACTIONTYPE_ID");

ALTER TABLE "Schema"."CUSTOMERORDERITEM" ADD CONSTRAINT
"CUSTOMERORDERITEM_CUSTOMERORDER_FK" FOREIGN KEY
      ("_CUSTOMERORDERCUSTOMERORDER_ID")
      REFERENCES "Schema"."CUSTOMERORDER"
      ("CUSTOMERORDER_ID");
```


Bottom Up – Composite/Atomic Pattern

Described here is a bottom up technique that can be used to develop a database that implements the Composite/Atomic pattern. Shown here is a typical example, using Product Offering ABE entities, but this technique can be used to transform other similar class hierarchies in the SID, such the Performance and Usage entity hierarchies, keeping in mind implementation guidance provided in the previous chapter of this book.

Figure 3.04 – Product Offering UML Model Fragment is typical example of the application of the Composite/Atomic pattern that will be used to demonstrate the bottom up technique.

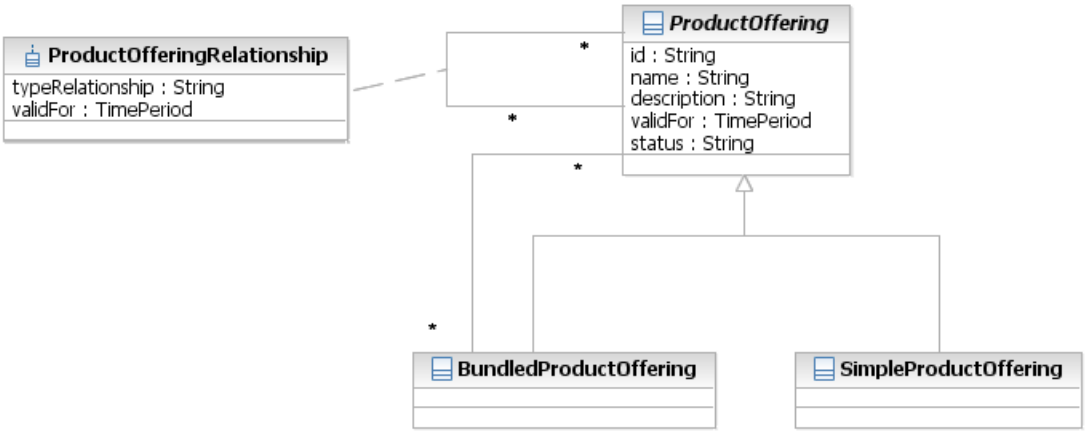

Figure 3.04 – Product Offering UML Model Fragment

Figure 3.05 – Product Offering Logical Data Model shows the UML model transformed to the logical data model. The primary difference is the introduction of foreign and primary keys. Also, note that the validFor attribute was not transformed as mentioned in the previous section that described the top down transformation technique.

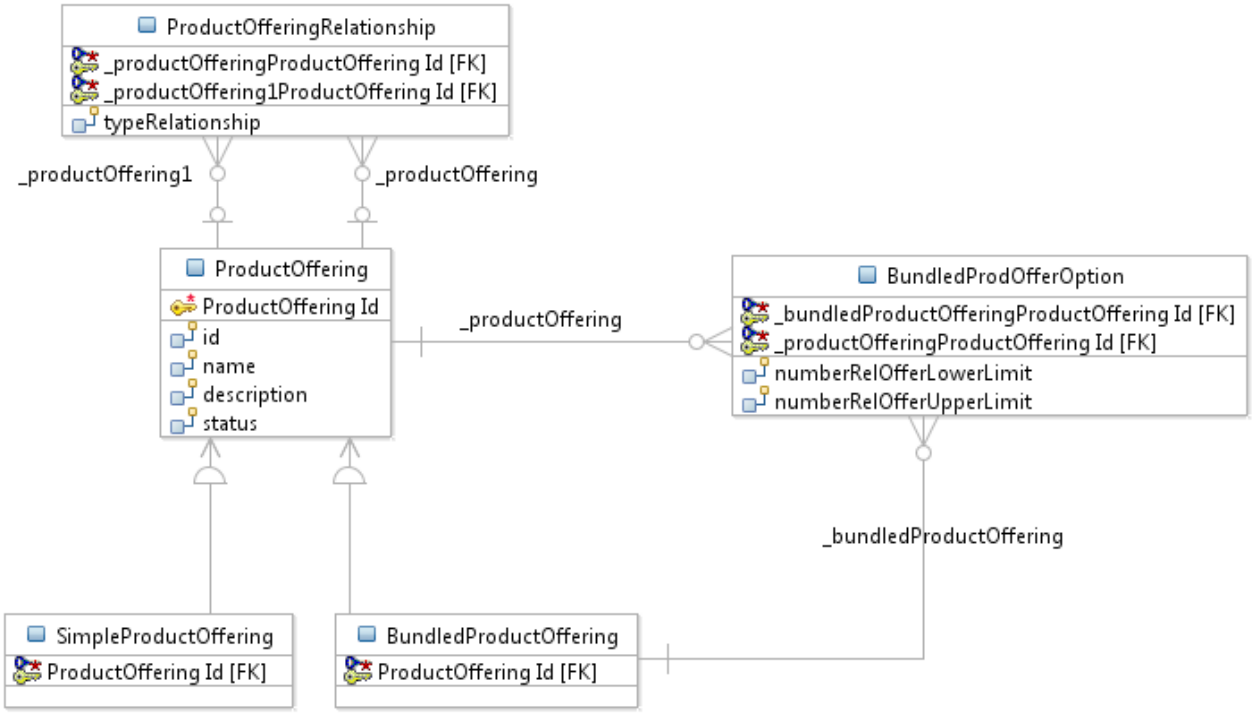

Figure 3.05 – Product Offering Logical Data Model

Figure 3.06 – Product Offering Physical Data Model shows the transformation from the logical data model to the physical data model. The BundledProductOffering and SimpleProductOffering subclasses were rolled up into the superclass ProductOffering.

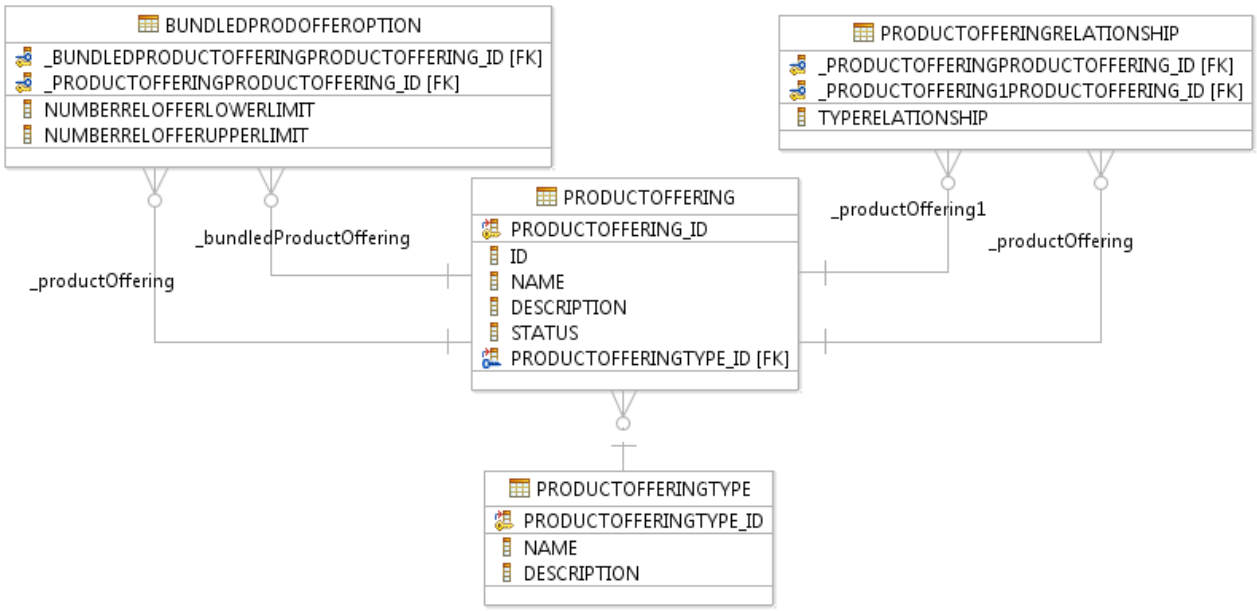

Figure 3.06 – Product Offering Physical Data Model

Notice that the BundledProdOfferOption entity is now related to the ProductOfferingEntity and has been transformed into an entity. Also, note that a new entity, ProductOfferingType, was added to the physical data model. This was done as part of the transformation by the tool used to perform the transformation. It was added to support the two different types of ProductOffering, bundled and simple. If the SID implementer prefers to use some other technique to make the distinction, such as a simple offering not having any lower level offerings, then this added entity can be removed.

Below is the full Data Definition Language (DDL) generated from the physical data model.

```
--<ScriptOptions statementTerminator=";"/>

CREATE SCHEMA "Schema";

CREATE TABLE "Schema"."BUNDLEDPRODOFFEROPTION" (
                "_BUNDLEDPRODUCTOFFERINGPRODUCTOFFERING_ID" INTEGER NOT
NULL,
                "_PRODUCTOFFERINGPRODUCTOFFERING_ID" INTEGER NOT NULL,
                "NUMBERRELOFFERLOWERLIMIT" INTEGER,
                "NUMBERRELOFFERUPPERLIMIT" INTEGER
        )
        DATA CAPTURE NONE;

CREATE TABLE "Schema"."DISTCHANNELPRODOFFER" (
                "_DISTRIBUTIONCHANNELDISTRIBUTIONCHANNEL_ID" INTEGER NOT
NULL
        )
```



```
      DATA CAPTURE NONE;

CREATE TABLE "Schema"."DISTRIBUTIONCHANNEL" (
            "DISTRIBUTIONCHANNEL_ID" INTEGER NOT NULL GENERATED BY
DEFAULT AS IDENTITY ( START WITH 1 INCREMENT BY 1 MINVALUE 1
MAXVALUE 2147483647 NO CYCLE CACHE 20 NO ORDER ),
            "ID" VARCHAR(32672),
            "NAME" VARCHAR(32672)
      )
      DATA CAPTURE NONE;

CREATE TABLE "Schema"."PRODUCTOFFERING" (
            "PRODUCTOFFERING_ID" INTEGER NOT NULL GENERATED BY
DEFAULT AS IDENTITY ( START WITH 1 INCREMENT BY 1 MINVALUE 1
MAXVALUE 2147483647 NO CYCLE CACHE 20 NO ORDER ),
            "ID" VARCHAR(32672),
            "NAME" VARCHAR(32672),
            "DESCRIPTION" VARCHAR(32672),
            "STATUS" VARCHAR(32672),
            "PRODUCTOFFERINGTYPE_ID" INTEGER NOT NULL
      )
      DATA CAPTURE NONE;

CREATE TABLE "Schema"."PRODUCTOFFERINGRELATIONSHIP" (
            "_PRODUCTOFFERINGPRODUCTOFFERING_ID" INTEGER NOT NULL,
            "_PRODUCTOFFERING1PRODUCTOFFERING_ID" INTEGER NOT NULL,
            "TYPERELATIONSHIP" VARCHAR(32672)
      )
      DATA CAPTURE NONE;

CREATE TABLE "Schema"."PRODUCTOFFERINGTERM" (
            "PRODUCTOFFERINGTERM_ID" INTEGER NOT NULL GENERATED BY
DEFAULT AS IDENTITY ( START WITH 1 INCREMENT BY 1 MINVALUE 1
MAXVALUE 2147483647 NO CYCLE CACHE 20 NO ORDER ),
            "_PRODUCTOFFERINGPRODUCTOFFERING_ID" INTEGER
      )
      DATA CAPTURE NONE;

CREATE TABLE "Schema"."PRODUCTOFFERINGTYPE" (
            "PRODUCTOFFERINGTYPE_ID" INTEGER NOT NULL,
            "NAME" CHAR(10),
            "DESCRIPTION" CHAR(40)
      )
      DATA CAPTURE NONE;

CREATE TABLE "Schema"."PRODUCTOFFERING_X_DISTRIBUTIONCHANNEL" (
            "PRODUCTOFFERING_ID" INTEGER NOT NULL,
            "DISTRIBUTIONCHANNEL_ID" INTEGER NOT NULL
      )
      DATA CAPTURE NONE;

ALTER TABLE "Schema"."BUNDLEDPRODOFFEROPTION" ADD CONSTRAINT
"BUNDLEDPRODOFFEROPTION_PK" PRIMARY KEY
      ("_BUNDLEDPRODUCTOFFERINGPRODUCTOFFERING_ID",
       "_PRODUCTOFFERINGPRODUCTOFFERING_ID");
```

```
ALTER TABLE "Schema"."DISTCHANNELPRODOFFER" ADD CONSTRAINT
"DISTCHANNELPRODOFFER_PK" PRIMARY KEY
      ("_DISTRIBUTIONCHANNELDISTRIBUTIONCHANNEL_ID");

ALTER TABLE "Schema"."DISTRIBUTIONCHANNEL" ADD CONSTRAINT
"DISTRIBUTIONCHANNEL_PK" PRIMARY KEY
      ("DISTRIBUTIONCHANNEL_ID");

ALTER TABLE "Schema"."PRODUCTOFFERING" ADD CONSTRAINT
"PRODUCTOFFERING_PK" PRIMARY KEY
      ("PRODUCTOFFERING_ID");

ALTER TABLE "Schema"."PRODUCTOFFERINGRELATIONSHIP" ADD CONSTRAINT
"PRODUCTOFFERINGRELATIONSHIP_PK" PRIMARY KEY
      ("_PRODUCTOFFERINGPRODUCTOFFERING_ID",
       "_PRODUCTOFFERING1PRODUCTOFFERING_ID");

ALTER TABLE "Schema"."PRODUCTOFFERINGTERM" ADD CONSTRAINT
"PRODUCTOFFERINGTERM_PK" PRIMARY KEY
      ("PRODUCTOFFERINGTERM_ID");

ALTER TABLE "Schema"."PRODUCTOFFERINGTYPE" ADD CONSTRAINT
"PRODUCTOFFERINGTYPE_PK" PRIMARY KEY
      ("PRODUCTOFFERINGTYPE_ID");

ALTER TABLE "Schema"."PRODUCTOFFERING_X_DISTRIBUTIONCHANNEL" ADD
CONSTRAINT "PRODUCTOFFERING_X_DISTRIBUTIONCHANNEL_PK" PRIMARY KEY
      ("PRODUCTOFFERING_ID",
       "DISTRIBUTIONCHANNEL_ID");

ALTER TABLE "Schema"."BUNDLEDPRODOFFEROPTION" ADD CONSTRAINT
"BUNDLEDPRODOFFEROPTION_BUNDLEDPRODUCTOFFERING_FK" FOREIGN KEY
      ("_BUNDLEDPRODUCTOFFERINGPRODUCTOFFERING_ID")
      REFERENCES "Schema"."PRODUCTOFFERING"
      ("PRODUCTOFFERING_ID");

ALTER TABLE "Schema"."BUNDLEDPRODOFFEROPTION" ADD CONSTRAINT
"BUNDLEDPRODOFFEROPTION_PRODUCTOFFERING_FK" FOREIGN KEY
      ("_PRODUCTOFFERINGPRODUCTOFFERING_ID")
      REFERENCES "Schema"."PRODUCTOFFERING"
      ("PRODUCTOFFERING_ID");

ALTER TABLE "Schema"."DISTCHANNELPRODOFFER" ADD CONSTRAINT
"DISTCHANNELPRODOFFER_DISTRIBUTIONCHANNEL_FK" FOREIGN KEY
      ("_DISTRIBUTIONCHANNELDISTRIBUTIONCHANNEL_ID")
      REFERENCES "Schema"."DISTRIBUTIONCHANNEL"
      ("DISTRIBUTIONCHANNEL_ID");

ALTER TABLE "Schema"."PRODUCTOFFERING" ADD CONSTRAINT
"PRODUCTOFFERING_PRODUCTOFFERINGTYPE_FK" FOREIGN KEY
      ("PRODUCTOFFERINGTYPE_ID")
      REFERENCES "Schema"."PRODUCTOFFERINGTYPE"
      ("PRODUCTOFFERINGTYPE_ID");

ALTER TABLE "Schema"."PRODUCTOFFERINGRELATIONSHIP" ADD CONSTRAINT
"PRODUCTOFFERINGRELATIONSHIP_PRODUCTOFFERING_FK" FOREIGN KEY
      ("_PRODUCTOFFERING1PRODUCTOFFERING_ID")
```

```
      REFERENCES "Schema"."PRODUCTOFFERING"
      ("PRODUCTOFFERING_ID");

ALTER TABLE "Schema"."PRODUCTOFFERINGRELATIONSHIP" ADD CONSTRAINT
"PRODUCTOFFERINGRELATIONSHIP_PRODUCTOFFERING_FK1" FOREIGN KEY
      ("_PRODUCTOFFERINGPRODUCTOFFERING_ID")
      REFERENCES "Schema"."PRODUCTOFFERING"
      ("PRODUCTOFFERING_ID");

ALTER TABLE "Schema"."PRODUCTOFFERINGTERM" ADD CONSTRAINT
"PRODUCTOFFERINGTERM_PRODUCTOFFERING_FK" FOREIGN KEY
      ("_PRODUCTOFFERINGPRODUCTOFFERING_ID")
      REFERENCES "Schema"."PRODUCTOFFERING"
      ("PRODUCTOFFERING_ID");

ALTER TABLE "Schema"."PRODUCTOFFERING_X_DISTRIBUTIONCHANNEL" ADD
CONSTRAINT
"PRODUCTOFFERING_X_DISTRIBUTIONCHANNEL_DISTRIBUTIONCHANNEL_FK"
FOREIGN KEY
      ("DISTRIBUTIONCHANNEL_ID")
      REFERENCES "Schema"."DISTRIBUTIONCHANNEL"
      ("DISTRIBUTIONCHANNEL_ID");

ALTER TABLE "Schema"."PRODUCTOFFERING_X_DISTRIBUTIONCHANNEL" ADD
CONSTRAINT
"PRODUCTOFFERING_X_DISTRIBUTIONCHANNEL_PRODUCTOFFERING_FK" FOREIGN
KEY
      ("PRODUCTOFFERING_ID")
      REFERENCES "Schema"."PRODUCTOFFERING"
      ("PRODUCTOFFERING_ID");

COMMENT ON COLUMN "Schema"."PRODUCTOFFERING"."DESCRIPTION" IS
'A narrative that explains what the offering is.';

COMMENT ON COLUMN "Schema"."PRODUCTOFFERING"."ID" IS
'A unique identifier for the ProductOffering.';

COMMENT ON COLUMN "Schema"."PRODUCTOFFERING"."NAME" IS
'A word, term, or phrase by which the ProductOffeirng is known and
distinguished from other ProductOfferings.';

COMMENT ON COLUMN "Schema"."PRODUCTOFFERING"."STATUS" IS
'The condition in which the offering exists, such as planned,
obsolete, active';

COMMENT ON TABLE "Schema"."DISTRIBUTIONCHANNEL" IS
'A distribution channel is the organization or entity by which a
product catalog is presented to a customer.';

COMMENT ON TABLE "Schema"."PRODUCTOFFERING" IS
'The presentation of one or more ProductSpecifications to the
marketplace for sale, rental, or lease for a ProductOfferingPrice.
A ProductOffering may target one or more MarketSegments, be included
in one or more ProductCatalog, presented in support of one or more
ProductStrategies, and made available in one or more Places.
ProductOffering may represent a simple offering of a single
```

```
ProductSpecification or could represent a bundling of one or more
other ProductOffering.';

COMMENT ON TABLE "Schema"."PRODUCTOFFERINGTERM" IS
'A condition under which a ProductOffering is made available to
Customers.  ProductOfferingTerm include
ProductOfferingFinancialTerm, which includes such things as
acceptable methods of payment, ShipmentTerm, and ServiceTerm.';
```

Middle Up & Down (Top Down, Bottom Up)

Designing a database that employs this technique is the combination of roll up and roll down techniques to a class somewhere in the "middle" of a class hierarchy. Therefore, there are examples that depict this technique have not been included.

Database Design & Other Modeling Patterns

This section describes database design techniques that can be used when implementing the EntitySpecification/Entity, Entity/EntityRole, and CharacteristicSpecification/CharacteristicValue patterns.

EntitySpecification/Entity

The previous chapter contained recommendations about collapsing specifications into entities, specifically that it introduces insert, update, and delete anomalies. However, if either or both of the entities in the application of this pattern involve class hierarchies, such as the Composite/Atomic pattern applied in the Product Specification ABE, then the techniques described in the previous section could be applied.

Entity/EntityRole

The previous chapter contained recommendations about collapsing specifications into entities, specifically that it also, like collapsing applications of the EntitySpecification/Entity pattern, introduces insert, update, and delete anomalies However, if either or both of the entities in the application of this pattern involve class hierarchies, such as the Composite/Atomic pattern, such as its application in the Product Specification ABE, then the techniques described in the previous section could be applied.

Figure 3.07 – Customer without Party Role shows a very basic implementation of the Customer entity.

Figure 3.07 – Customer Without Party Role

This physical data model was generated without generating the accompanying Party, PartyRole, and related entities, such as PartyName and ContactMedium. The next set of figures and their associated figures will demonstrate how a complete customer database can be generated.

A general guideline to remember is when developing a database for an entity that is a subclass of a role entity, and the role is in another ABE, don't forget to transform both to the logical data model. Or, the resulting physical data model will look like the one in the figure above.

Figure 3.08 – Partial PartyRole and Customer Logical Data Model shows a possible database design if the choice is made to develop a separate Party database.

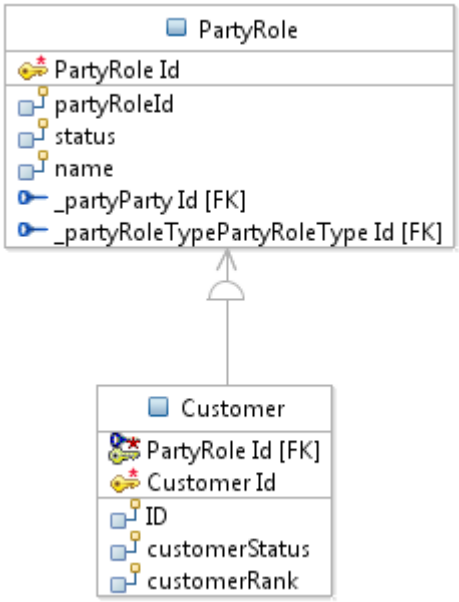

Figure 3.08 – Partial PartyRole and Customer Logical Data Model

Figure 3.09 – Partial Party, PartyRole, and Customer Physical Database shows the addition of the Party entity and PartyRoleType entity for the physical database. An upcoming figure will a possible implementation where a separate Party database has not been developed.

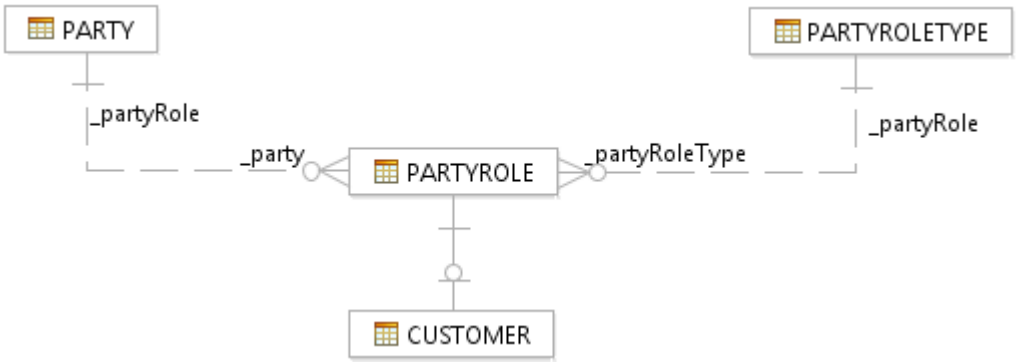

Figure 3.09 – Partial Party, PartyRole, and Customer Physical Database

The dashed association line in this and all figures means the association is non-identifying.

The next two figures show the development of a Customer physical data model in which Party and PartyRole do not appear, but in which the associated entities appear as directly related to Customer. This will be accomplished by collapsing the PartyRole hierarchy and applying other implementation techniques, all of which will be described following Figure 3.10 – Customer Logical Data Model.

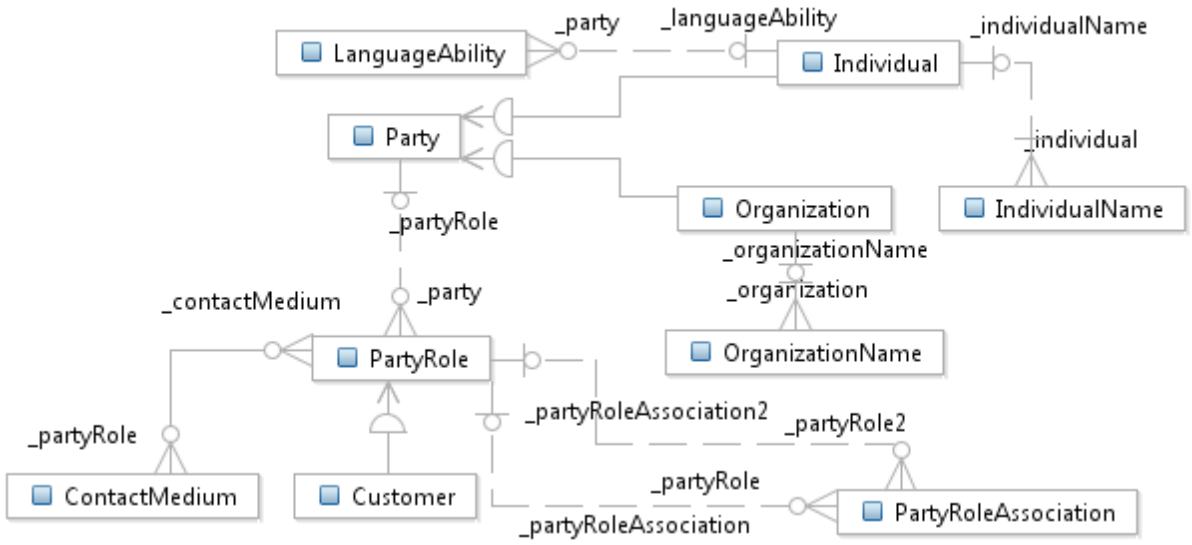

Figure 3.10 – Customer Logical Data Model

Two transformation techniques were employed to transform the logical data model to the physical data model:

- Rolling down Party into Individual and Organization, which creates associations between PartyRole and these entities
- Rolling down PartyRole into Customer so that "inherits" the associations to entities associated with PartyRole, including Individual and Organization.

Note that these could vary based on the database design tool used. However, no matter what tool is used, the end result should be the same or close to what is shown in Figure 3.11 – Customer Physical Data Model. The figure shows the implementation of the Customer entity, not all the other entities within the Customer ABE.

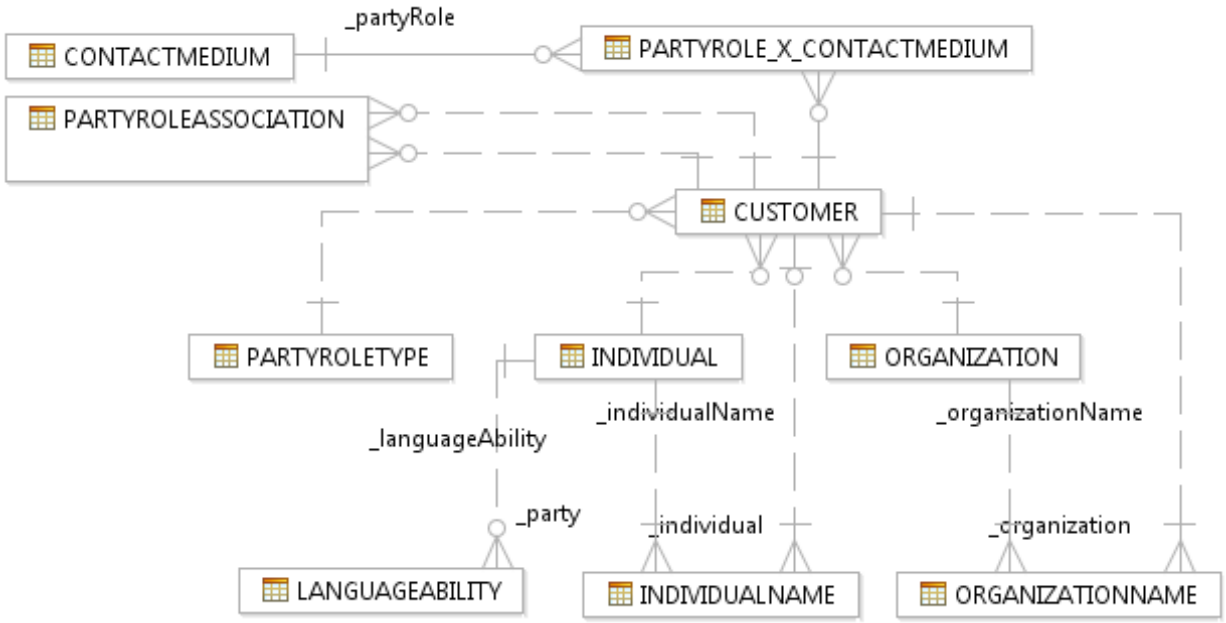

Figure 3.11 – Customer Physical Data Model

The subclasses of ContactMedium are not shown in this implementation example. They could be rolled up so that there is a single ContactMedium entity in the physical database, or they could be rolled down so that single tables are created for each way in which a party can be contacted. One subclass to keep in mind is the PostalContact subclass of Contact Medium and ContactMedium itself. Both are related to address entities, that would provide the address of the customer as well as the address to which instances of ContactMedium apply, such as a phone number applying to an address that represents the home address of the Customer.

The Address (Place) part of the model is not shown, but could be implemented using a combination of techniques that have been described here and in the previous chapter of this book.

Further transformations can be made in either the logical data model (if the final physical data model has already been envisioned). For example, the names of the entities Individual and Organization could be renamed to IndividualCustomer and OrganizationCustomer.

The names of the Party related entities could also be changed to use Customer instead. Also, not that PartyRoleType could represent the EntitySpecification portion of the EnittySpecification/Entity pattern. This would enable dynamic attributes to be defined for each "type of customer by incorporating the Characteristic pattern into the Customer ABE model . This could be easily done by adding an association to the EntitySpeCharUse entity in the UML model before any transformations are performed. This does illustrate the point that it is often of benefit to know the intended design of the database. If not, then several iterations of transformations will be required, which is realistically what often occurs.

The implementation of the Customer ABE shown here represents just one implementation option. There can be many others, such as rolling Individual and Organization up into Party, collapsing Party into Role, and Role into Customer. These decisions often are dependent on the use cases that will act on entities within an application and with any other applications that require access to customer information (if a single customer database is maintained).

If this alternate approach is taken then all the individual/organization attributes reside in Customer. This means that from an application perspective it would be more of a challenge to determine if an individual or organization exists as more than one instance of a customer.

When designing any database use cases specific to the entities contained in the database can be used as one input into determining the design of the database. In this example, use cases associated with Individual and Organization types of Customers, such as when setting them up, changing information, preparing for billing, including the associated user interfaces should be taken into consideration. If the majority of use cases vary significantly based on the type of Party, then retaining separate Individual and Organization entities may be desirable.

There may be additional tasks to perform after the initial generation of the physical data model. Figure 3.12 – Details of Customer Physical Data Base Entity shows the Customer entity after its transformation to its physical implementation.

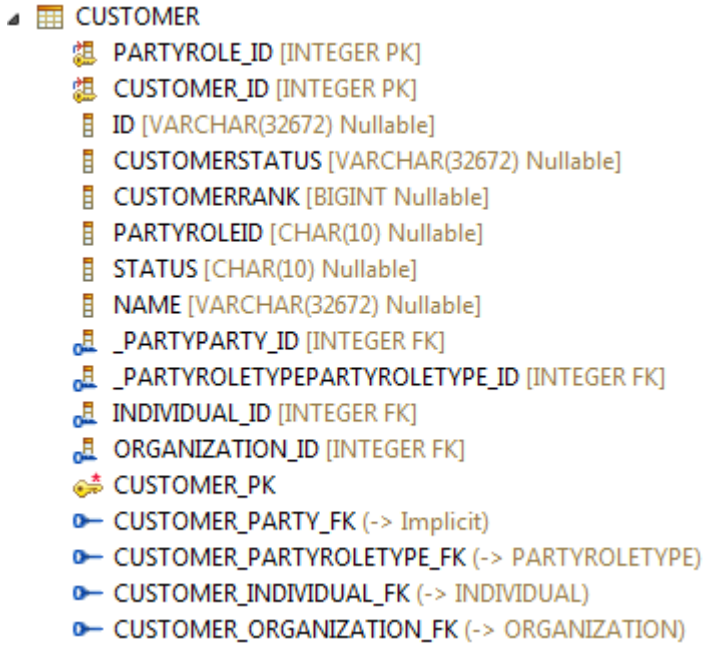

Figure 3.12 – Details of Customer Physical Data Base Entity

In the figure there are both primary and foreign keys, such as PartyRoleID and PartyPartyID, that may not be of interest from a database perspective. These can be deleted. Note that there is also an implicit Customer_Party foreign key, which resulted from the roll downs of entities. An implicit key represents a reference to some other entity that is not necessarily of interest, which means it is not enforced in any way. Various forms of foreign keys are discussed further in the **Error! Reference source not found.** section of Chapter 6.

CharacteristicSpecification/CharacteristicValue

Although Characteristics are duplicated across a number of domains (this practice is no longer continued in the SID), a single set of shareable Characteristic tables can be implemented. The Characteristic model that is contained in the Root Business Entity ABE in the Common Business Entity domain can be used as a "template" for this. If there is a concern about performance, an implementation could include a set of tables for each domain, shareable by entities where Characteristics are used by entities within a domain.

Figure 3.13 – Basic Characteristic and "Use" Entities shows a basic/simplified view of a database design that has implemented the Characteristic model as-is. Only a subset of the implemented entities are shown.

The complete model of characteristic specifications UML model is shown in the figure following Figure 3.13 – Basic Characteristic and "Use" Entities.

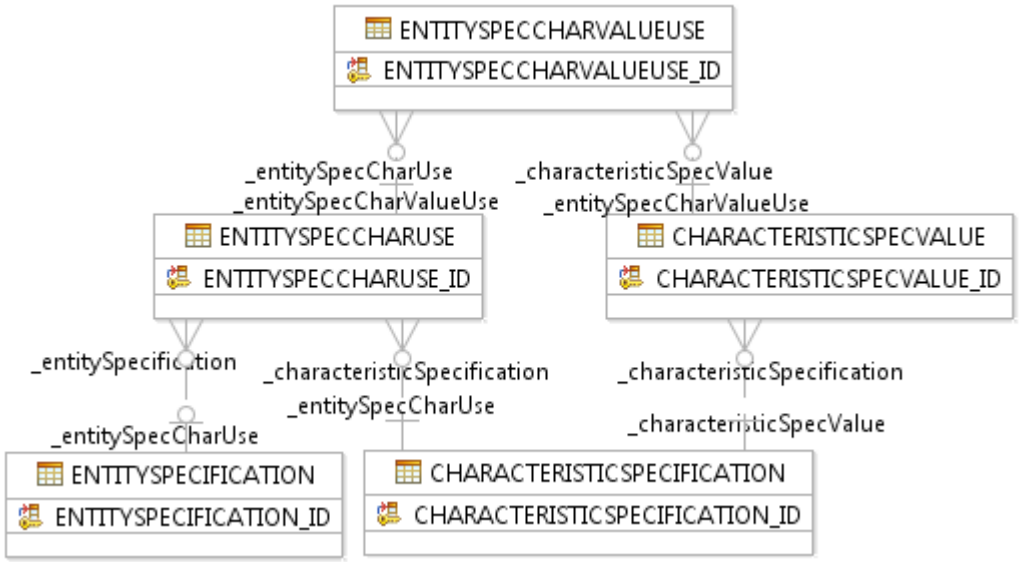

Figure 3.13 – Basic Characteristic and "Use" Entities

Figure 3.14 – Full Characteristic Specification Model

The model shown could be transformed as-is to a logical data model. Since this is the case, a full example of the transformation is not show here.

Figure 3.15 – Product Specification "Use" of Characteristics shows the use of the Characteristic model to support the dynamic sub-classing of ProductSpecification entities. This is accomplished by adding an association between the ProductSpecifcation entity and the EntitySpecCharUse entity.

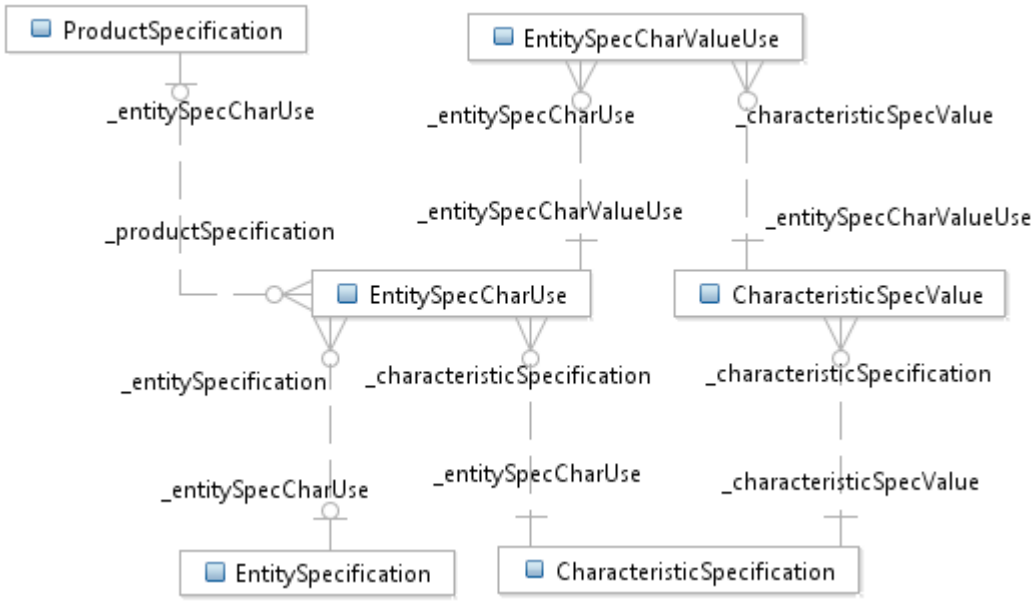

Figure 3.15 – Product Specification "Use" of Characteristics

Figure 3.16 – Product Specification and Characteristic Keys shows how the ProductSpecfication entity "uses" the generalized Characteristic model in a physical data model. This is an alternative to repeating the application/implementation of the Characteristic model within the Product Specification ABE. Note that the existing associations in the Product domain that relate ProdSpecCharValueUse to and ComponentProdOfferPrice and to ProductOffering need to be "moved" to EntitySpecCharValueUse. This may also mean that a subclass EntitySpecCharValueUse should be created, possibly named ProdDomainCharValueUse.

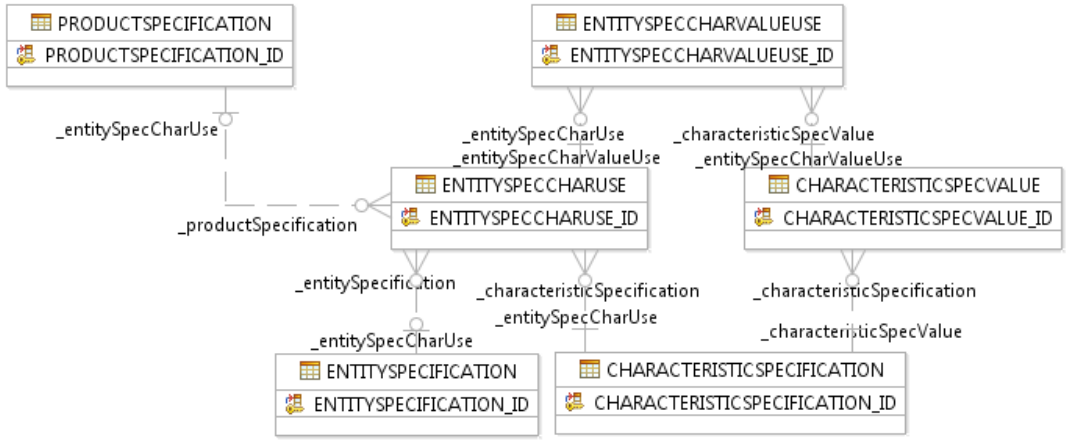

Figure 3.16 – Product Specification and Characteristic Keys

The figure shows primary keys but not foreign keys. Some examples of foreign keys are shown in the next figure.

Figure 3.17 – Example Physical Entity with Columns and All Keys shows the EntitySpecCharUse entity with all columns (attributes), key columns (primary and foreign), as well as primary and foreign keys.

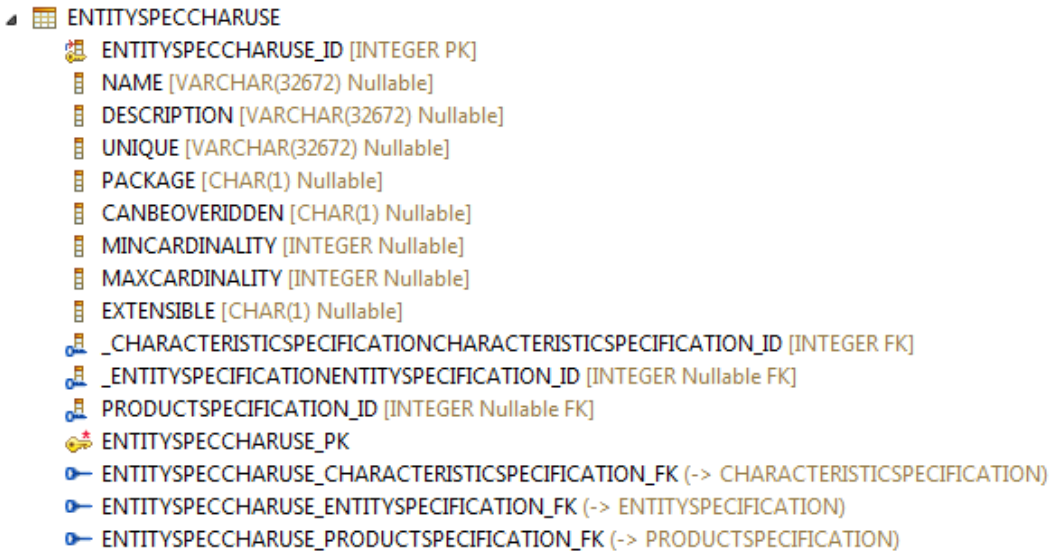

Figure 3.17 – Example Physical Entity with Columns and All Keys

There are other ways to support this, rather than associating each entity to EntitySpecCharUse entity. One is to have a single entity (table) that contains instances of all specifications, similar to the EntitySpecification entity shown in Figure 3.14 – Full Characteristic Specification Model. To support the extension of just entities, such as Customer and ProductSpec attributes, an entity similar to RootEntityType could be used. This will be discussed further in a later chapter of this book.

Characteristic Performance Improvement Opportunities

There are a number of performance improvement opportunities associated with implementing Characteristics. First, the specification part of the model should be cached in some way to ensure acceptable performance. This avoids having to navigate a database when dynamically constructing a user interface or using the CharacteristicSpecification entities to support any other functionality.

Another opportunity is to implement the relationship between CharacteristicSpecification and CharacteristicValue "by reference", as opposed to "by value". In this case the reference (foreign key) is the name, but a foreign key can be created for any attribute. The "name" attribute could also be defined as the primary key of CharacteristicSpecification, if its values will be unique.

Essential to a successful implementation of this pattern are considerations to take into account for the CharacteristicValue entity. This entity only has two attributes, value and validFor (from and to effective dates). Conceptually, the value attribute is only populated when a value is entered for the Characteristic that does not have enumerated values specified. For example, a userId associated with an email account. From an implementation perspective, queries are often made on CharacteristicValues.

For performance and simplicity reasons, de-normalizing key attributes from CharacteristicSpecification entities should be considered before the UML model is transformed. Typical candidates for de-normalization are name, value (from enumerations in CharacteristicSpecValue), and unitOfMeasure. This enables queries to be made on the name attribute and eliminate the need to navigate to CharacteristicSpecification implementation of the SID.

There may be volume performance concerns with implementing the CharacteristicValue table in general or for a given application of it. If this is the case then the planned implementation should prototyped to ensure there are no performance issues. If there are then refer to the **Error! Reference source not found.** section in Chapter 5.

Working with Other SID ABEs

This section describes implementation considerations when transforming other SID ABEs, including:

- Base Types
- Contact Medium
- Location

Working with Base Type ABE Entities

Base Type ABE entities may be treated in various ways by database design tools when transforming the UML model to a logical data model.

Base Type entities represent composite attributes, such as a TimePeriod entity that contains a startDateTime attribute and an endDateTime attribute. The data type of an attribute, such as validFor, is set to this entity, currently modeled as a class, but may be changed to a UML datatype in a future version of the SID. This avoids having to define two attributes in an entity and does provide a consistent specification for attributes that represent this type of composite attribute.

Some tools may replace the attribute in an entity that references the Base Type entity with the attributes that make up the composite. Others may transform the Base Type entity as-is and show a one-to-one association to the entity that contains an attribute that references the Base Type entity. This type of transformation is shown in Figure 3.18 – TimePeriod as a Related Entity. Note that the validFor attribute is shown as the name of the association between PartyRole and TimePeriod.

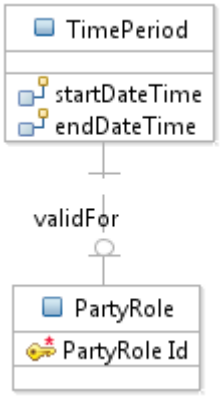

Figure 3.18 – TimePeriod as a Related Entity

The TimePeriod entity can be rolled up into the entities that reference it. If the TimePeriod entity in the logical data model contains foreign key columns these will be rolled up into the referencing entity also. These can be deleted after the logical data model is transformed to the physical data model. Keep in mind that if they are deleted in the logical data model then the roll up may not happen. Depending on the database design tool used, if the roll up is done for one entity it is done all entities that reference the Base Type entity.

Some other technique, such as doing an initial transformation to the physical data model, then removing the foreign key columns from the logical data model, and performing the transformation to the physical data model again. Some tools provide "compare" functionality when a physical data model is re-generated. Coupled with merge or replace options they provide a means to automate the removal of the foreign key columns in the referencing entity.

The more preferred technique would be to develop a plug-in that automatically perform the transformation if the database design tool supports plug-in development.

Figure 3.19 – Time Period Attributes Transformed As Attributes shows an example of the final physical data model after the foreign key columns have been removed from the PartyRole entity.

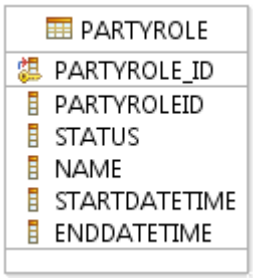

Figure 3.19 – Time Period Attributes Transformed As Attributes

Notice that validFor has disappeared from the physical data model. The names of the two validFor attributes could be changed to include a "validFor" prefix.

Implementing the Contact Medium ABE

The section in this chapter that showed how the Party ABEs and the Customer ABE could be implemented. However, it did not show details of how the Contact ABE (within the Party ABE) or the Location ABEs could be implemented to support a complete Customer database. An possible implementation of the Contact ABE is shown here followed by a possible implementation of Location ABEs.

Figure 3.20 – Contact Medium Logical Data Base Model shows the results of transforming the UML Contact ABE to its logical data model equivalent. Note that the subclasses of ContactMedium, EmailContact, PostalContact, FaxNumber, and TelephoneNumber are not shown but are present in the logical data model, but are present in it. They will be rolled up into the ContactMedium entity during the transformation to the physical data model.

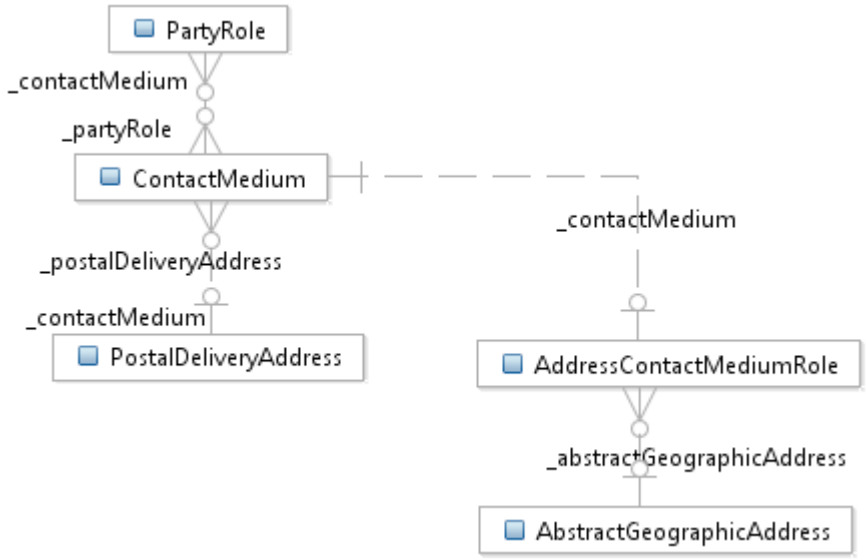

Figure 3.20 – Contact Medium Logical Data Base Model

The transformation of this model to the physical data model is shown in Figure 3.21 – Contact Medium Physical Data Model.

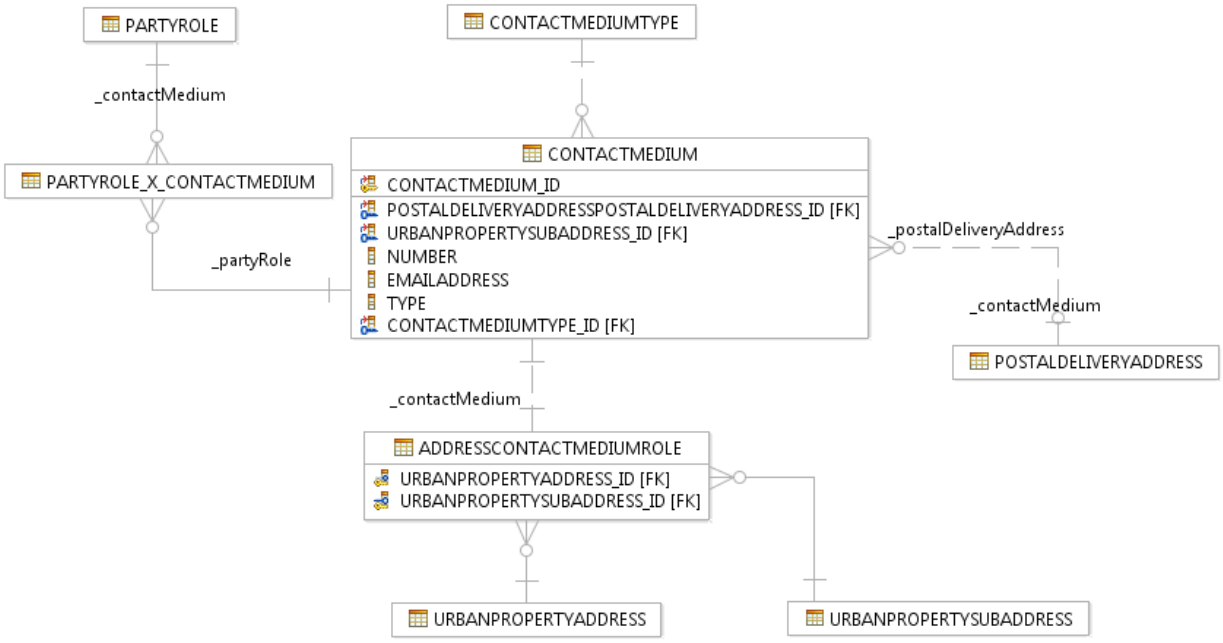

Figure 3.21 – Contact Medium Physical Data Model

Note that a new entity, ContactMediumType has been automatically added to the model during the transformation process. This was also shown in an earlier section, when the subclasses of ProductOffering were also rolled up. The ContactMedium "type" attribute that was rolled up from the TelephoneNumber subclass could be used instead of this generated entity.

An intersection/link table has also been automatically added to resolve the many-to-many association between PartyRole and ContactMedium. This entity should most likely have been added to the UML model. An attribute in the entity could indicate the type of contact medium, such as office, home, primary, secondary.

Also, the multiplicity of the UML association between ContactMedium and UrbanPropertyAddress should possibly be changed to many-to-many in the UML model to support multiple addresses that have the same contact information.

The next section will provide an example transformation of the Location ABE, which together with implementation of the Contact ABE, and what was shown for the Customer ABE, demonstrates a full implementation of all entities that support the Customer ABE.

Implementing the Location ABE

The first two figures mirror the current UML model with its deep class hierarchy, which will be simplified by the transformation. Roll down roll up that have been demonstrated in other sections of this chapter will be used to simplify the physical data model.

Figure 3.22 – Partial Address Logical Data Model shows the class hierarchy associated with UrbanPropertyAddress and UrbanPropertySubAddress. The figure only shows the keys associated with each entity. The physical data model will show the keys as well as the columns.

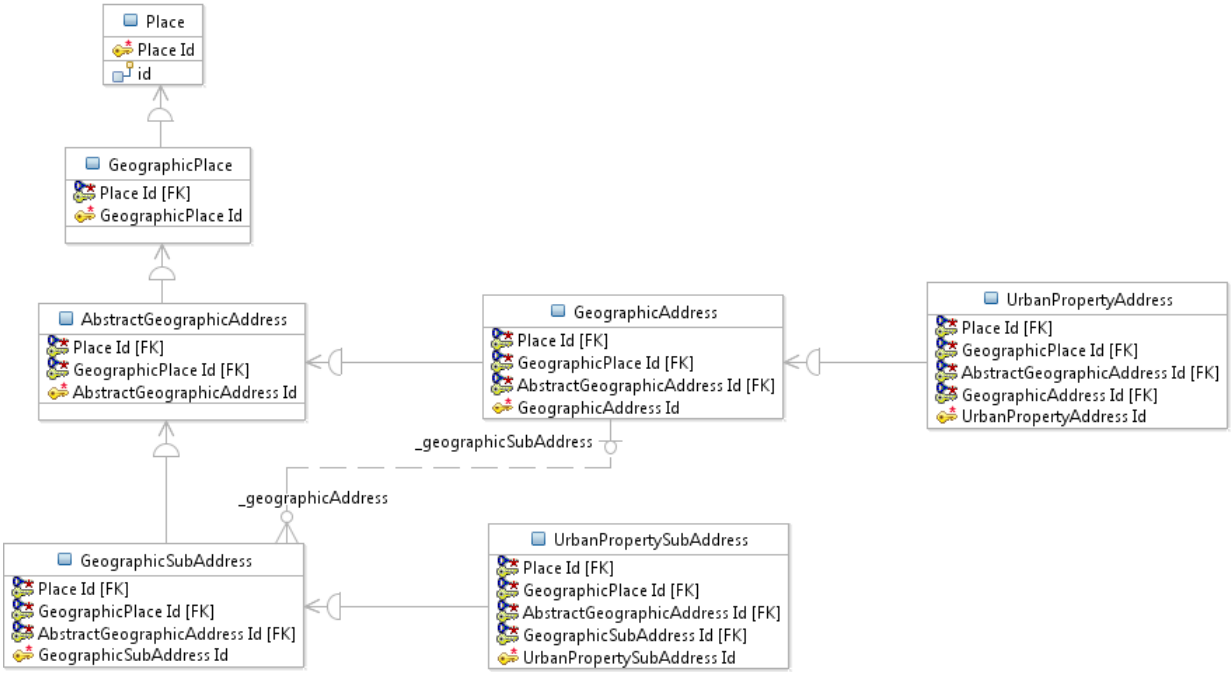

Figure 3.22 – Urban Property Addresses Logical Data Model

When transforming this part of the logical data model Place, GeographicPlace, AbstractGeographicAddress, GeographicAddress and GeographicSubAddress entities will be rolled down.

Figure 3.23 – Postal Delivery Address Logical Data Model Model shows the class hierarchy associated with PostalDeliveryAddress and PoBoxAddress entities. The figure only shows the keys associated with each entity. The physical data model will show the keys as well as the columns.

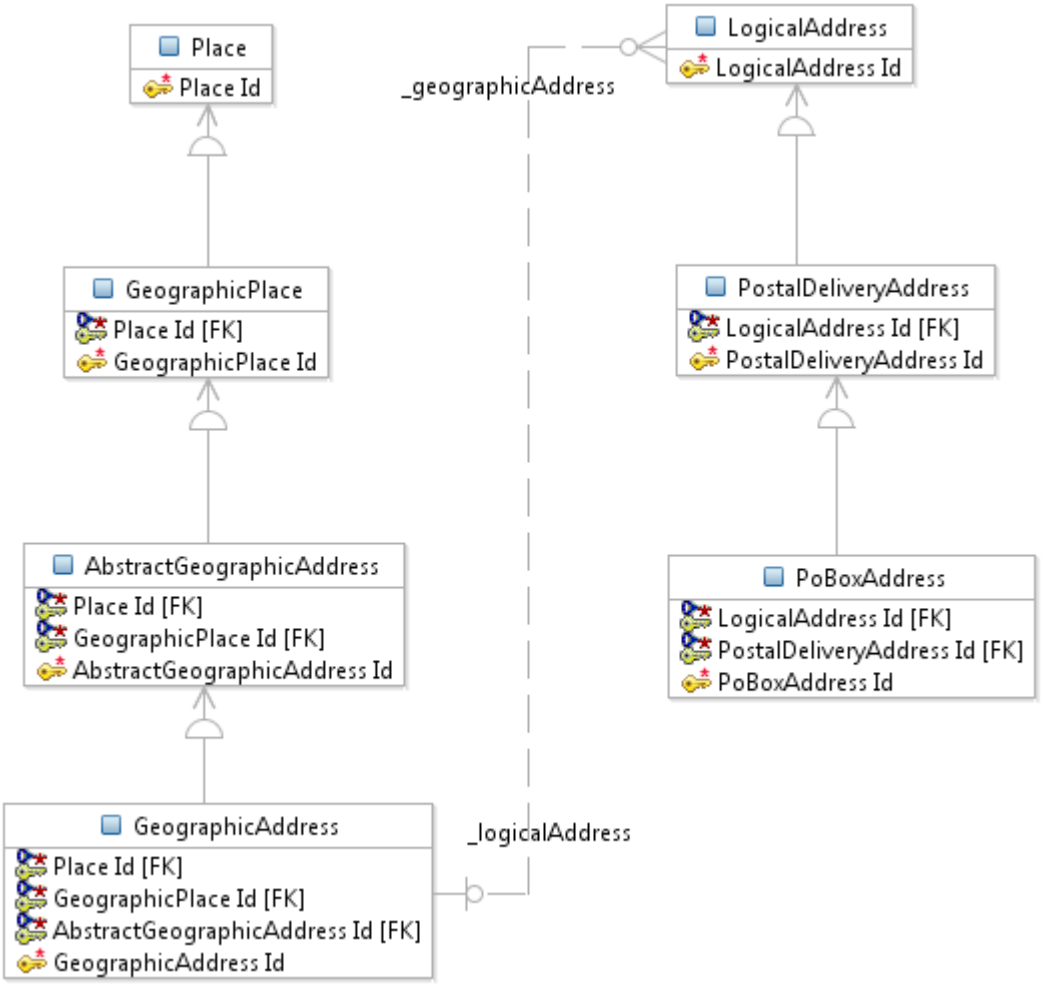

Figure 3.23 – Postal Delivery Address Logical Data Model

When transforming this part of the logical data model, in addition to the entities rolled down in the previous figure, LogicalAddress will be rolled down, and PoBoxAddress will be rolled up.

Figure 3.24 – Address Physical Data Model shows a much simplified physical data model that contains only three entities.

Figure 3.24 – Address Physical Data Model

A number of foreign and primary keys can be deleted from the entities, such as PlaceID, GeographicAddressID from UrbanPropertyAddress. Removing keys, such as these, was described in earlier sections of this chapter. This is the first time that it can be seen that many of them inherited from super-classes that were rolled down can be removed.

The next chapter explains how conformance to the SID can still be maintained from a model perspective and if interfaces expose this physical representation of the SID information model.

Some SID implementers choose to use the entity spec and char patterns as an alternate for the current Location model. This includes further normalizing some attributes in the current model, such as Country, PostCode, StateorProvince, by creating entities for them. Conformance to the current model can also be maintained as will be described in the next chapter.

Lessons Learned

This chapter's final section presents some lessons that have been learned over time when transforming the SID into its physical representation. It describes both information model considerations and database design tool considerations that should be taken into account when implementing the SID in its physical form.

Information Model Considerations

Within each Level 1 SID ABE there are often implied Level 2 ABEs that represent the Level 1 concept. For example, within the Level 1 Product Offering ABE there are a number of Level 2 ABEs, such as Product Catalog and Product Offering Price. Along with these are a set of entities that represent the entities that define a Product Offering, which are not included in a Level 2 ABE called Product Offering. Figure 3.25 – SID Product Offering ABE shows the Product Offering Level 1 ABE and its decomposition.

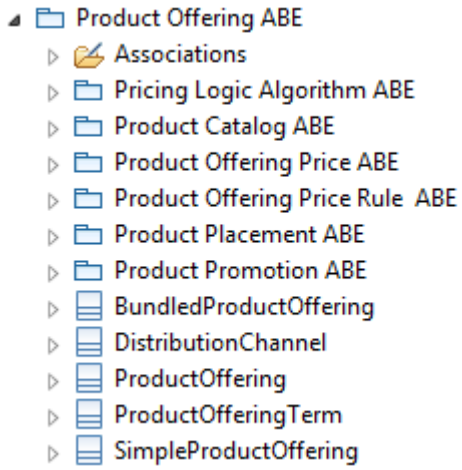

Figure 3.25 – SID Product Offering ABE

Often a data base that supports only a subset of the Product Offering Level 1 ABE will be designed. Most database design tools support a partial transformation of a Level 1 ABE by allowing the selection of Level 2 or lower level ABEs. However, because Product Offering entities are included directly under the Product Offering Level 1 ABE, the Product Offering L1 ABE must be selected in order to transform these entities. That means that all the ABEs may have to be included. In order to enable a subset of the ABEs to be selected, it may be necessary to add a Level 2 ABE called Product Offering and move the Product Offering entities to it.

Database design tools should/will resolve many-to-many (*-*) associations by adding an entity to the logical or physical data mode. This entity represents the intersection between the two entities involved in the association. However, there is typically information (attributes/associations/subclasses) that describes association. Therefore, it may be advantageous to add the entity before any transformation to the logical data model is made.

Make certain all the associations are included in the correct ABE. Sometimes, an assumption is made that an association that is to be included in the transformation is contained within the ABE(s) being transformed. This assumption may not always be correct! For example, the association between ContactMedium and PostalDeliveryAddress is currently in the Association folder directly under the Party Level 1 ABE. Therefore, when transforming the Contact ABE, this association will not be transformed, losing the association in the logical data model. Making certain about the location of associations can prevent a lot of re-transformations!

Database Design Tool Considerations

Database design tools may transform aggregation associations in a manner which is not expected. For example, even if an aggregation association is many-to-many (*-*), the "aggregating role" of the association may be transformed into a 0,1. Therefore, it is a good practice when first working with a tool to investigate how this type of association is transformed before beginning any "real" transformations.

As mentioned in the Information Model Considerations section, a tool should support partial transformations of the SID. Also, it is not recommended that the entire SID be transformed at one time; some tools may not support this volume of transformations. Additionally, some associations that are actually abstract should not be transformed, such as the association between BusinessInteraction and BusinessInteractionItem. This association is explicitly modeled for each type of interaction, such as in the Customer Order ABE.

Some database design tools may not support the transformation of all UML objects, such as data types. Any that are needed, but not supported may require the development of a plug in that can become part of the transformation from the UML (information model) to the logical/physical data model.

The following is copied from an earlier section of this chapter because it also applies here:

> There a number of changes that were made before and after the transformation to the physical data model. Non-needed keys, such as those from rolled down super-classes, may been removed. For example, requestID can be made non-persistent in the logical model and businessInteractionID can be removed in the physical model after the proper foreign keys were generated. Some tools allow this to be done in logical data model, so regeneration of physical data model does not require removal of them. Keep in mind that this is not unique to the SID.
>
> However, it may be necessary to retain some inherited primary key attributes in the logical data model to correctly generate the physical data base foreign key attributes. As with any tool that transforms an information model to various stages of data models, practice with the tool is essential! Knowing the desired end result will impact how the transformations are defined and the number of manual changes that will be required at each step in the transformation.

Copy/merge (incremental updates) on transformations should be a requirement, particularly when going from UML to the logical data model and the logical data model to the physical data model. Figure 3.26 – Structure Compare of Two Physical Data Models shows a compare of a new transformation to a physical data model (on the left) with the existing physical data model (on the right). The arrow icons at the bottom right of the figure allow for copy or merge of the new transformation to the existing physical data model.

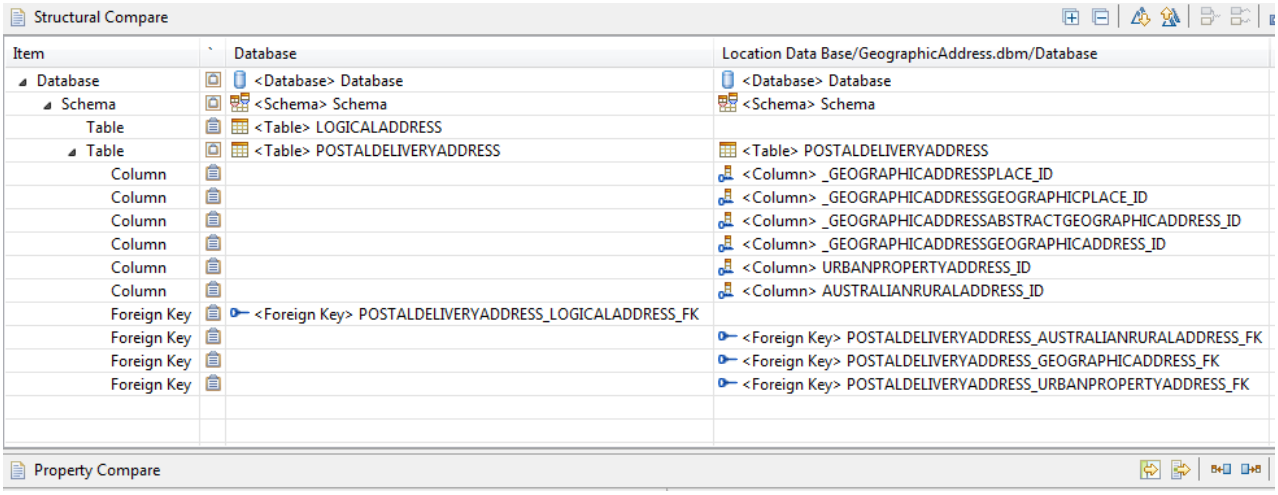

Figure 3.26 – Structure Compare of Two Physical Data Models

