Frameworx How-To Guide

**Information Framework (SID)**

*User’s Guide*

**Information Framework Suite**

**GB922 User’s Guide**

**Release 18.0.1**

**October 2018**

|  |  |
| --- | --- |
| **Latest Update: TM Forum Release 18.0.1** | **TM Forum Approved**  **Suitable for Conformance** |
| **Version 18.0.2** | **IPR Mode: RAND** |

# **Notice**

Copyright © TM Forum 2018. All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to TM FORUM, except as needed for the purpose of developing any document or deliverable produced by a TM FORUM Collaboration Project Team (in which case the rules applicable to copyrights, as set forth in the [TM FORUM IPR Policy](http://www.tmforum.org/IPRPolicy/11525/home.html), must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by TM FORUM or its successors or assigns.

This document and the information contained herein is provided on an “AS IS” basis and TM FORUM DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

TM FORUM invites any TM FORUM Member or any other party that believes it has patent claims that would necessarily be infringed by implementations of this TM Forum Standards Final Deliverable, to notify the TM FORUM Team Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the TM FORUM Collaboration Project Team that produced this deliverable.

The TM FORUM invites any party to contact the TM FORUM Team Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this TM FORUM Standards Final Deliverable by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the TM FORUM Collaboration Project Team that produced this TM FORUM Standards Final Deliverable. TM FORUM may include such claims on its website but disclaims any obligation to do so.

TM FORUM takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this TM FORUM Standards Final Deliverable or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on TM FORUM's procedures with respect to rights in any document or deliverable produced by a TM FORUM Collaboration Project Team can be found on the TM FORUM website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this TM FORUM Standards Final Deliverable, can be obtained from the TM FORUM Team Administrator. TM FORUM makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

Direct inquiries to the TM Forum office:

4 Century Drive, Suite 100

Parsippany, NJ  07054 USA

Tel No.  +1 973 944 5100

Fax No.  +1 973 944 5110

TM Forum Web Page: [www.tmforum.org](http://www.tmforum.org/)

Table of Contents

Notice 2

Table of Contents 4

List of Figures 6

1. Introduction 8

2. Principles used in the Information Framework 9

2.1. Stereotypes and keywords 9

2.1.1. Core Framework Component Status definition 9

2.1.2. Additional Stereotypes and Keywords 11

2.2. Business Identifiers 12

2.3. Attributes’ multiplicity 13

2.4. Naming Conventions for Association, Attribute, and Package 13

2.4.1. Association Naming Conventions 13

2.4.2. Attribute Naming Conventions 17

2.4.3. Attribute Capitalisation Conventions 18

2.4.4. Package Naming Conventions 18

2.4.5. Guidelines for Naming Entities 19

2.4.6. Guidelines for Defining Associations 19

2.5. Patterns For Defining New Aggregate Business Entities 20

2.5.1. Business Entity Patterns 21

2.5.2. Entity Specification/Entity Pattern 21

2.5.3. Entity/Entity Role 23

2.5.4. Composite/Atomic 24

2.5.5. CharacteristicSpecification/CharacteristicValue 24

2.5.6. Entity Classification Group 25

2.6. General Modeling Guidelines 27

2.6.1. Best Practices 27

2.6.2. General Note about Model Diagrams 27

2.6.3. Installing RSA 28

3. Extending the Information Framework 28

3.1. Creating Packages to Hold Extensions 28

3.1.1. Adding attributes 29

3.1.2. Adding entities 31

3.1.3. Adding associations 33

4. Implementing the Information Framework 33

4.1. Implementing SID Patterns 33

4.1.1. Class Hierarchy Implementation Techniques 34

4.1.2. Implementing Other Modeling Patterns 44

4.2. SID and Database Design 47

4.2.1. A Note on SID and Database Design 47

4.2.2. Database Design & Class Hierarchy Patterns 47

4.2.3. Database Design & Other Modeling Patterns 57

4.2.4. Working with Other SID ABEs 66

4.2.5. Lessons Learned 72

5. Administrative Appendix 75

5.1. About this document 75

5.2. Document Life Cycle 75

5.3. Document History 75

5.3.1. Version History 75

5.3.2. Release History 77

5.4. Acknowledgments 78

List of Figures

[Figure U.1 – Simple Association Naming 13](#_Toc519012543)

[Figure U.2 – Using One Business Entity in the Name of an Association 15](#_Toc519012544)

[Figure U.3 – Using Both Related Business Entities in the Name of an Association 16](#_Toc519012545)

[Figure U.5 – Qualifying Attribute Names 18](#_Toc519012546)

[Figure U.6 – Using “ABE” in the Name of an ABE Package 19](#_Toc519012547)

[Figure U.24 – Derived Association Example 20](#_Toc519012548)

[Figure U.7 – Entity Specification/Entity Pattern 21](#_Toc519012549)

[Figure U.8 – Service Specification/Service Use Of the Entity Spec/Entity Pattern 22](#_Toc519012550)

[Figure U.9 – Service Specification ABE 22](#_Toc519012551)

[Figure U.10 – Entity/Entity Role Pattern 23](#_Toc519012552)

[Figure U.11 – Party/Party Role 23](#_Toc519012553)

[Figure U.12 – Composite/Atomic Pattern 24](#_Toc519012554)

[Figure U.13 – Product Specification Composite/Atomic 24](#_Toc519012555)

[Figure U.14 – Entity Specification Characteristic/Entity Characteristic Pattern – 1 25](#_Toc519012556)

[Figure U.15 – Entity Specification Characteristic/Entity Characteristic Pattern – 2 25](#_Toc519012557)

[Figure U.16 – Classification Group Pattern 26](#_Toc519012558)

[Figure U.17 – Market Statistic Use of Classification Group Pattern 27](#_Toc519012559)

[Figure U.18 – Package Structure For Information Framework Extensions 29](#_Toc519012560)

[Figure U.19 – Adding Attributes Using Sub-Classing 30](#_Toc519012561)

[Figure U.20 – Adding Attributes Using Composition 30](#_Toc519012562)

[Figure U.21 – Adding Attributes Using Stereotypes 31](#_Toc519012563)

[Figure U.22 – Adding an Entity Related to an Existing Entity – Technique 1 32](#_Toc519012564)

[Figure U.23 – Adding a Related Business Entity 33](#_Toc519012565)

[Figure U.25 – Consolidating Sub-Classes – Top Down – Before 35](#_Toc519012566)

[Figure U.26 – Consolidating Sub-Classes – Top Down – After 35](#_Toc519012567)

[Figure U.27 – Consolidating Sub-Classes – Bottom Up – Before 36](#_Toc519012568)

[Figure U.28 – Consolidating Sub-Classes – Bottom Up – After 37](#_Toc519012569)

[Figure U.29 – Consolidating Sub-Classes – Top Down – Bottom Up – Before 37](#_Toc519012570)

[Figure U.30 – Consolidating Sub-Classes – Top Down – Bottom Up – After 38](#_Toc519012571)

[Figure U.31 – Consolidating Sub-Classes – Typing – Before 38](#_Toc519012572)

[Figure U.32 – Consolidating Sub-Classes – Typing – After 39](#_Toc519012573)

[Figure U.33 – Performance Management Class Hierarchy 39](#_Toc519012574)

[Figure U.34 – Transformed Performance Management Class Hierarchy 40](#_Toc519012575)

[Figure U.35 – Product and Inventory Performance 40](#_Toc519012576)

[Figure U.36 – Usage Class Hierarchy 41](#_Toc519012577)

[Figure U.37 – Transformed Usage Class Hierarchy 41](#_Toc519012578)

[Figure U.38 – Service Hierarchy – Bottom Up – Before 43](#_Toc519012579)

[Figure U.39 – Service Hierarchy – Bottom Up – After 43](#_Toc519012580)

[Figure U.40 – Party and PartyRole 45](#_Toc519012581)

[Figure U.41 – PartyRoles Transformed Into Party Subclasses 45](#_Toc519012582)

[Figure U.42 – Customer Order Logical Data Model Fragment 49](#_Toc519012583)

[Figure U.43 – Transformed Customer Order Model Fragment 49](#_Toc519012584)

[Figure U.44 – Customer Order Inherited Associations 50](#_Toc519012585)

[Figure U.45 – Product Offering UML Model Fragment 52](#_Toc519012586)

[Figure U.46 – Product Offering Logical Data Model 53](#_Toc519012587)

[Figure U.47 – Product Offering Physical Data Model 53](#_Toc519012588)

[Figure U.48 – Customer without Party Role 58](#_Toc519012589)

[Figure U.49 – Partial PartyRole and Customer Logical Data Model 58](#_Toc519012590)

[Figure U.50 – Partial Party, PartyRole, and Customer Physical Database 59](#_Toc519012591)

[Figure U.51 – Customer Logical Data Model 59](#_Toc519012592)

[Figure U.52 – Customer Physical Data Model 60](#_Toc519012593)

[Figure U.53 – Details of Customer Physical Database Entity 61](#_Toc519012594)

[Figure U.54 – Basic Characteristic and “Use” Entities 62](#_Toc519012595)

[Figure U.55 – Full Characteristic Specification Model 62](#_Toc519012596)

[Figure U.56 – Product Specification “Use” of Characteristics 63](#_Toc519012597)

[Figure U.57 – Product Specification and Characteristic Keys 63](#_Toc519012598)

[Figure U.58 – Example Physical Entity with Columns and All Keys 64](#_Toc519012599)

[Figure U.59 – Product Specification Use of Characteristics 65](#_Toc519012600)

[Figure U.60 – Inter-Domain Characteristic Associations 65](#_Toc519012601)

[Figure U.61 – Relationships Between Characteristic Use Entities 66](#_Toc519012602)

[Figure U.62 – TimePeriod as a Related Entity 67](#_Toc519012603)

[Figure U.63 – Time Period Attributes Transformed As Attributes 68](#_Toc519012604)

[Figure U.64 – Contact Medium Logical Database Model 68](#_Toc519012605)

[Figure U.65 – Contact Medium Physical Data Model 69](#_Toc519012606)

[Figure U.66 – Urban Property Addresses Logical Data Model 70](#_Toc519012607)

[Figure U.67 – Postal Delivery Address Logical Data Model 71](#_Toc519012608)

[Figure U.68 – Address Physical Data Model 72](#_Toc519012609)

[Figure U.69 – SID Product Offering ABE 73](#_Toc519012610)

[Figure U.70 – Structure Compare of Two Physical Data Models 74](#_Toc519012611)

# Introduction

The Information Framework (SID) is a framework of frameworks. The framework of frameworks approach provides inherent extensibility and flexibility. The Information Framework program will be describing and documenting new functionality during each phase, so it is important to enable this new functionality to be easily added and extended without adversely affecting the overall Information Framework model.

This Addendum provides guidelines and examples of principles used in the Information Framework and how to define extensions to the Information Framework.

As a by-product, it also provides guidelines for developing ABEs[[1]](#footnote-1) that have not yet been modelled by the Information Framework team. Use of these guidelines will enable extensions made by different people and organizations to have the same structure, thus enabling these extensions to be compatible with each other as well as with the Information Framework itself. It also enables such extensions to be proposed back to the Information Framework team for official incorporation as part of the Information Framework.

The guidelines presented here include:

* Naming convention for association, attribute, and package
* Patterns and rules for extending existing business entities
* Patterns for new ABE development (which are used to design model packages and elements)
* General modeling guidelines.

Details about SID conformance can be found on the TM Forum Conformance Certification Assessment web pages [HERE](https://www.tmforum.org/conformance-certification/).

The remainder of this document will use “Information Framework model” or “Information Framework” to refer to GB922 and its Addenda, which collectively define the Frameworx Business view of the Information Framework.

# Principles used in the Information Framework

## Stereotypes and keywords

In the Information Framework, some stereotypes are used to highlight considerations of maturity from ABEs, Business Entities or Attributes. Where the profile is not applied there is no specific statement to make on the maturity.

Like all the Frameworx, the Information Framework follows the Core Framework Component Status. In addition, some statuses have been added.

### Core Framework Component Status definition

Al the Frameworx (eTOM, SID and TAM) components share the same basic status. Refer to GB991 Core Frameworks Concepts and Principles to find more information.

#### Status definitions

As core Frameworks Components evolve, they go through the following evolutionary stages or statuses:

* Released
* Preliminary
* Draft
* Not Fully Developed

##### Released

A core Frameworks Component with a Released Status is:

* Complete, i.e. no additions required
* Frameworx Team (maturity level 3) and Forum approved (level 4)

A core Frameworks Component with a Released Status is included as the basis for TM Forum Frameworx Conformance Certification.

##### Preliminary

Preliminary is typical work that is delivered over a series of iterations.

A core Frameworks Component with a Preliminary Status is:

* Incomplete, i.e. a deeper study is in-progress to complete the content of the item and it is usable as it is. Some modification might be done.
* Frameworx Team (maturity level 3) and Forum approved (level 4)

Note that even if a Preliminary component is approved by the Frameworx Team and the Forum, it won’t be included in TM Forum Conformance Certification as it has to be completed.

##### Draft

A Core Frameworks Component with a Draft Status is:

* Incomplete, i.e. a deeper study is in-progress to complete the content of the item and present it to approval
* Not yet Frameworx Team and Forum approved (maturity level 1 or 2)

***Note: A Draft work can’t remove or move any existing item and all new items have to be gathered in a new area with a Draft maturity level***

##### Not Fully Developed

A core Frameworks Component with a not fully developed status is

* Empty or almost empty, i.e. no work has started to specify it but it has been identified as required

In case of almost empty the Frameworx Team has to review the item and decide if it has to be removed.

#### How to specify the Status

The items concerned by the Status are:

* Domain: applies to everything inside the Domain
* ABE: applies to everything inside the ABE
* BE: applies to all attributes that describe the BE

In the model, the Status is specified as the following:

* Released: nothing specific. As a consequence, by default no stereotype means “Released” Status.
* Preliminary: with a “preliminary” stereotype on the package and entities
* Draft: with a “draft” stereotype on the package and entities
* Not Fully Developed: with a “not fully developed” stereotype on the package and with at least a description
* All the generated models (browse model, HTML, XSD…) contain only the items with a maturity level Released, Preliminary and Not Fully developed
* The RSA model contains all items of all Status

In Guide Books:

* Released: nothing specific
* Preliminary: with “preliminary” included in the title or section
* Draft: does not appear in the guide book
* Not Fully Developed: Only appears in the Concepts and Principles document and are marked as “not fully developed”

In the Information Framework Map:

* Released: black font in a white rectangle
* Preliminary: black font in a medium-grey rectangle
* Draft: not presented
* Not Fully Developed: black font in a dark-grey rectangle

### Additional Stereotypes and Keywords

In addition to standard Frameworx stereotypes, we use some specific stereotypes and keywords in the Information Framework.

#### Stereotypes and Keywords definitions

##### doNotImplement

This stereotype indicates that the marked element should NOT be used in an implementation (i.e. is not applicable in a TM Forum interface implementation).

This stereotype is used for packages containing Class Diagrams or examples of sub-classing a Business Entity that aren’t required in an implementation for a conformance perspective.

##### likelyToChange

The entity marked is identified as a likely candidate for future adjustments and refinements. This entity is still recommended for use in existing and new developments and it is also recommended that the developers monitor the appropriate TMF activities to track any proposed adjustments/refinements. There is no guarantee that changes will be made to the entity as whilst in this state the entity is still under study.

##### likelyToBeDeprecated

The entity marked is identified as a likely candidate for future removal.

It is suggested that this entity is not used in new developments. Where it is used in existing developments efforts should be made to consider future migrate to an alternative form. It is recommended that the developers monitor the appropriate TMF activities to track work on the identified entity.

If no member specified an issue with removing this entity, it is removed at the next release.

##### Required

This stereotype specifies when the attribute is required in an implementation.

In this case, the implementation of such an attribute will be checked for conformance compliance.

#### How to specify the Status

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Domain | Package | BE | Attribute |
| doNotImplement | NA | Applicable | Applicable | Applicable |
| likelyToChange | Applicable | Applicable | Applicable | Applicable |
| likelyToBeDeprecated | Applicable | Applicable | Applicable | Applicable |
| Required | NA | NA | NA | Applicable |

## Business Identifiers

The SID Information Model uses the concept of core business entities representing entities within an ABE that are not dependent upon any other entity within the ABE. Examples are: Customer, Product, and Service.

Generally, there is one single core business entity per ABE, with the same name as the name of the ABE. There are exceptions where a ABE contains more than one core entity; example: Customer and CustomerAccount are both core entities of the Customer ABE.

With respect to Business Identifiers, the SID is an information model, which only has business identifiers and not database unique keys.   
Typically, but not always, the presence of a business identifier attribute (ID) for a given entity aligns with the core nature of this entity. In other words, only core business entities have an ID and non-core business entities do not have (this ID attribute may be inherited from a superior entity).

Examples:

ID attributes in Customer and Customer Account, the two core Customer ABE entities

ID attribute in CharacteristicSpecification in the Characteristic ABE.

It is considered that other business entities (non-core) in an ABE are dependent on the core business entities and don’t need to have a business identifier, as they can be reached by navigation from the core business entity.

Example:

CharacteristicSpecValue do not need an ID attribute as it can be reached by navigation from CharacteristicSpecification via the CharacteristicSpecificationEnumatedBy association.

This example illustrates a “life cycle” dependency between CharacteristicSpecValue and CharacteristicSpecification through the presence of the 1..\* cardinalities. More precisely, if an instance of CharacteristicSpecification is deleted then the dependent instances of CharacteristicSpecValue are deleted as well.

In other words, from a business perspective, non-core entities will be reachable by navigating from a core entity using the appropriate associations; there is no need/intention to refer to these non-core entities directly.

## Attributes’ multiplicity

In the Information Framework multiplicities for attributes aren’t specified because they can at most take on a single value.

The reason is that if an attribute can take multiple values it is typically “promoted” to an entity that has a \* association to it from the entity that would have contained the multi-valued attribute.

## Naming Conventions for Association, Attribute, and Package

### Association Naming Conventions

One method of naming associations is to simply use a verb.

To interpret the meaning of the name, the two related business entities are used as subject and object.

For example, in the figure below the association between ProductSpecification and BusinessInteractionItem is interpreted as ProductSpecification Involvedin BusinessInteractionItem; the meaning is interpreted by using the upper business entity as the subject.

Similarly, the association between Service and BusinessInteractionItem is interpreted as Service InvolvedIn BusinessInteractionItem; the meaning is interpreted from left to right.

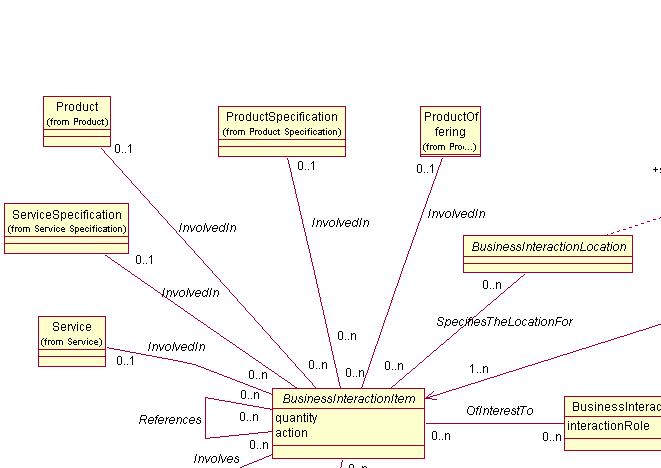


Figure U.1 – Simple Association Naming

This method of interpretation works well when the business entities retain their relative positions in every diagram in which they appear.

However, if in another diagram the positions of business entities are reversed, then the interpretation of the associations is incorrect.

For example, if the positions of BusinessInteractionItem and ProductSpecification are reversed, the interpretation of the association is BusinessInteractionItem InvolvedIn ProductSpecification, which is an incorrect interpretation.

This problem with interpreting the meaning of an association can be eliminated by introducing the name of either or both of the related business entities into the name of the association.

If one business entity name is used, the name should include the subject as shown in Figure U.2 – Using One Business Entity in the Name of an Association below.

For example, BusinessInteractionTypeCategorizes (BusinessInteraction, the implied object). This naming convention is always used unless the association name results in a duplicate name.

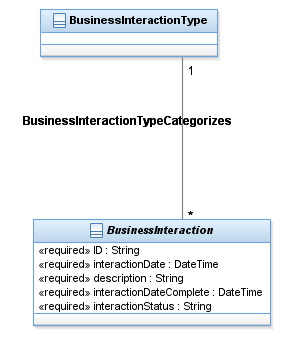


Figure U.2 – Using One Business Entity in the Name of an Association

To resolve duplicate association names both related business entities are used to form the name of the association as shown in Figure U.3 below.

For example, BusinessInteractionItemInvolvesProductSpecification.

The subject of the association name should be the ABE that owns the association. This means that in the model the association appears in the package containing the business entity that appears as the subject in the association name.

In this example, the association appears in the BusinessInteraction package, in which BusinessInteractionItem resides and the relationship’s name contains both Entities names.

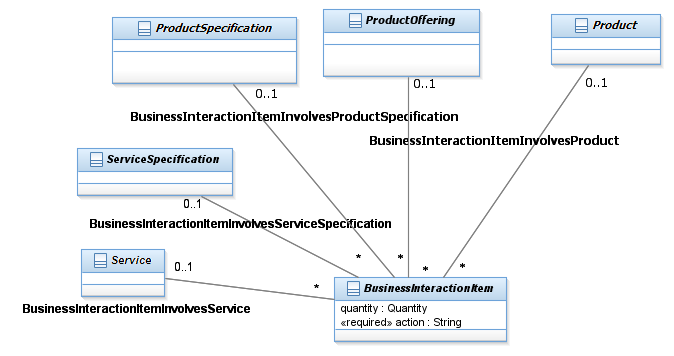


Figure U.3 – Using Both Related Business Entities in the Name of an Association

It may appear that always using both related business entities in the name of the association would be more prudent. One convention is much simpler to use and understand than two. However, there are a number of reasons for employing two naming conventions. They are:

1. Using both related business entities results in long association names that tend to clutter up diagrams and are difficult to use, and
2. Using the subject/verb/object naming convention will most likely only be needed on a minimum number of occasions.

### Attribute Naming Conventions

Case 1:

Attributes having the same meaning in several business entities of the model should share the same name.

Vice versa, two attributes from different entities in the model having the same name should share the same meaning (indeed relatively to their respective business entity).

It does not necessarily mean that the corresponding value/values of these different “identical” attributes should be exactly the same.

Attribute names of the model in this category (non-exhaustive list):

      description

      validFor

      status

Using a common set of attribute names for common business meanings imparts the model with a sense of unity and consistency. A casual reviewer will gain a clear understanding of the model, while someone looking for a more rigorous definition will go to the documentation of the attribute in context of the corresponding entity.   
A variation in the name (when correctly given) will raise a flag for the model reviewer that this attribute has a meaning that does not align with the common definition, and alert the reviewer to look at the specific documentation.

Case 2:

Attributes of the same entity must have unique names.

It may happen that several attributes of the same entity relate to similar kind of information, but indeed with specific meaning.   
A typical example is “date”. The same entity may contain several attributes representing “date” information.

In such a case it makes sense to reflect the kind of information in the name of the attribute (e.g. “date”), but indeed, the name of these attributes must be qualified to distinguish them.

In the figure below, PartyBillingCycle carries different dates with different meanings, that’s why they are prefixed by a qualifier.

The choice of the appropriate qualifiers is left to the designer of the model.

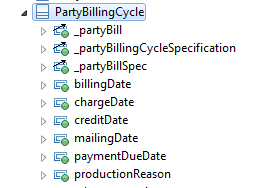


Figure U.5 – Qualifying Attribute Names

In the case of Boolean attributes, the recommended convention is to use “is” or “has” as a prefix. For example, the attribute “active” which can take on the values of “yes” or “no”, should be named isActive.

### Attribute Capitalization Conventions

* Attributes should not start with a capital letter.
* If the attribute is made of several words concatenated, camel case should be used (e.g. isOperational, serviceState, servicePolicyValidFor).
* Acronyms should be avoided as much as possible; exception are acceptable in the case the acronym is well known in the context where it is used.   
  In this case, acronyms should be represented only with capital letters even when it starts the name of the attribute (e.g. UTCOffset, GMTSessionStartDateTime, clientMACAddress).
* Attribute values representing constant values should use only capital letters.

### Package Naming Conventions

Package names in the Information Framework model correspond to the names of ABEs in the Information Framework.

Each package corresponding to an ABE must be suffixed by “ABE” and most of the time the main Business Entity’s name is used for the ABE name putting a space between each word.

Package names employing the ABE suffix are shown in the figure below.

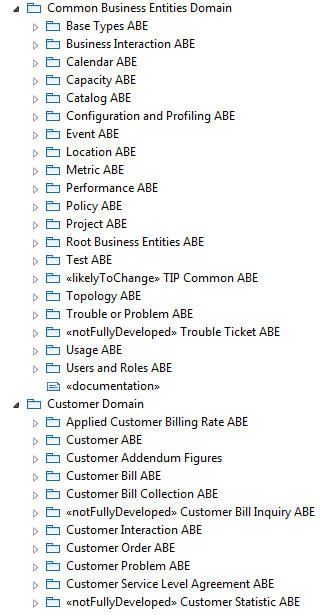


Figure U.6 – Using “ABE” in the Name of an ABE Package

### Guidelines for Naming Entities

Business entities should take on a name that will be familiar to a business individual. The name should not be expressed using technical jargon and must be singular.

### Guidelines for Defining Associations

In general, inter-ABE associations should have a multiplicity of \* – \*. This is done in order to minimize coupling (dependencies) between ABEs that reside in different Domains.

In the information model the use of derived relationships provides a way to present alternate views in a diagram that:

* Reduce the number of business entities on the diagram
* To show how to navigate from one entity to another via the transitivity of the associations.

Figure U.24 – Derived Association Example first shows an example of a transitive association between PaymentPlan and PaymentMethod via their association to PaymentPlanPaymentMethod. The second example shows a derived association directly between the two entities. The use of a “/” prefix in the association name indicates that the association is derived.



Figure U.24 – Derived Association Example

Note that the figure only shows the single entity PaymentPlanPaymentMethod between the two indirectly related entities. In other cases there can be more than one entity and two associations that indirectly relate two entities that can be used to show a derived association between the two entities.

## Patterns For Defining New Aggregate Business Entities

These guidelines should be used when developing a new ABE. Occasionally they may be used when adding more detail to an existing ABE. The guidelines include:

* Business entity patterns
* Entity Specification/Entity Pattern
* Entity/Entity Role
* Composite/Atomic
* CharacteristicSpec/CharacteristicValue
* Entity Classification Group

### Business Entity Patterns

There are established sets of business entity patterns that are used in the Information Framework. These include:

* Entity Specification/Entity
* Entity/Entity Role
* Composite/Atomic
* Characteristic Spec/ Characteristic Value.

Guidelines for implementing the patterns can be found in Chapter - Implementing the Information Framework

### Entity Specification/Entity Pattern

The Entity Specification/Entity pattern is used throughout the Information Framework model.

Typically, most core business entities (that is, an entity within an ABE that is not dependent upon any other entity within the ABE, such as Customer, Product, and Service) have their invariant attributes, methods, relationships and constraints defined by a specification, such as Product Specification and Service Specification. Customer does not have a related specification entity at this time.

This pattern should not be applied to existing ABEs, but should receive high consideration when adding a new ABE or detailing an existing ABE that has not been developed. The figure below shows the use of this pattern in the Root ABE.

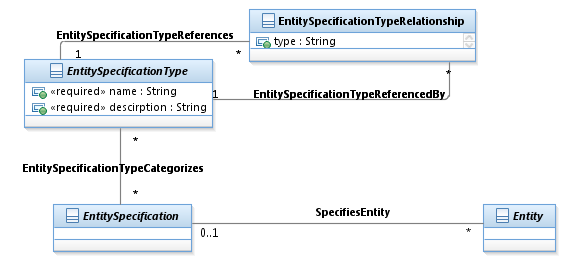


Figure U.7 – Entity Specification/Entity Pattern

The figure below shows the use of the Entity Specification/Entity pattern in the Service domain.

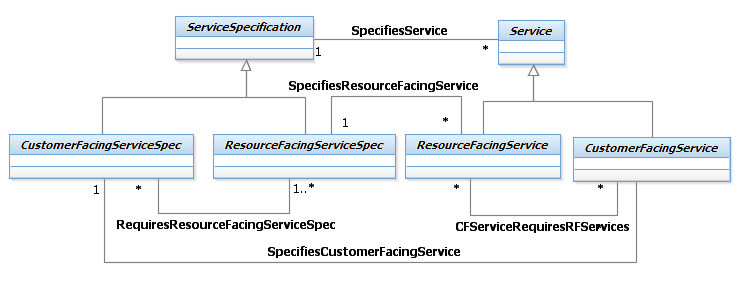


Figure U.8 – Service Specification/Service Use Of the Entity Spec/Entity Pattern

Typically, there will be an ABE for the specification and an ABE for the entity characterized by the specification within each Information Framework domain or complex ABE.

A complex ABE is one that decomposes into a number of other ABEs or into a number of nested ABEs. The reason for this is that each of these business concepts is complex enough to contain a number of related and dependent business entities.

The figure below shows the Service Specification ABE to illustrate this point.

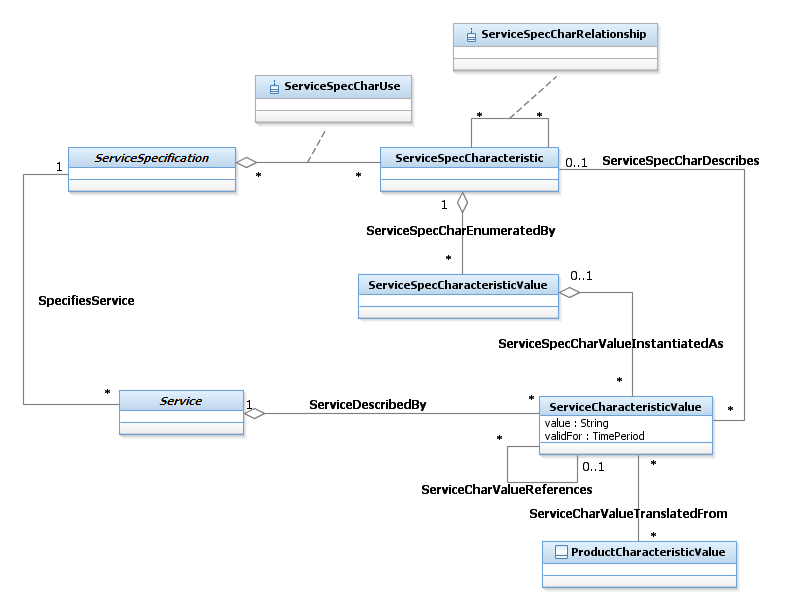


Figure U.9 – Service Specification ABE

### Entity/Entity Role

Many business entities take on a variety of roles during their life of interest to a business.

For example, an individual may be a customer and an employee of a service provider.

The Entity/EntityRole pattern, shown in the figure below, is used throughout the Information Framework model to represent this concept.

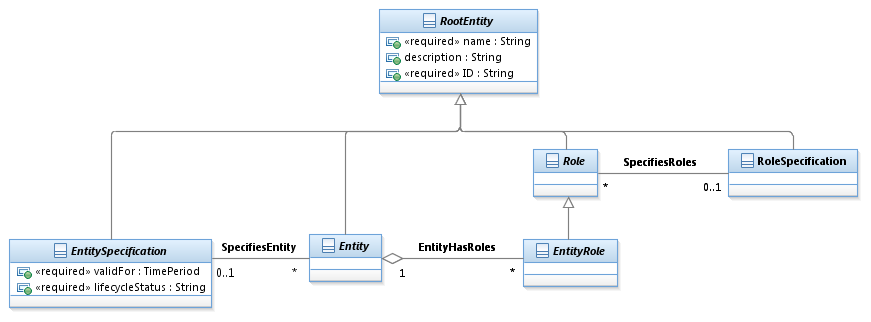


Figure U.10 – Entity/Entity Role Pattern

The figure below shows the use of this pattern in the Party ABE. Typically, each role that the entity plays is modelled as a subclass of the role entity. Many of these roles are also modelled as ABEs in their own right because of their importance to the business.

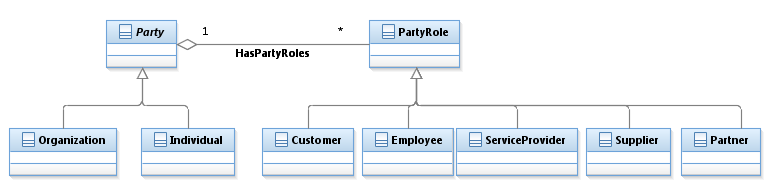


Figure U.11 – Party/Party Role

### Composite/Atomic

Often, instances of a business entity are composed of other instances of the same business entity. For example, the price for cellular phone service may include a fixed monthly charge and a charge for excess minutes used.

To model this, the Information Framework model employs the Composite/Atomic pattern, shown in the figure below.

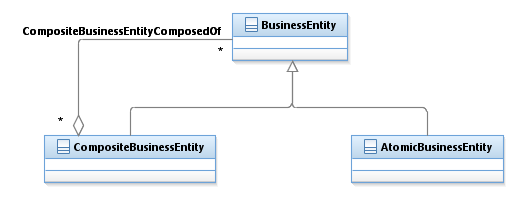


Figure U.12 – Composite/Atomic Pattern

The figure below shows the use of this pattern within the Product Specification ABE.

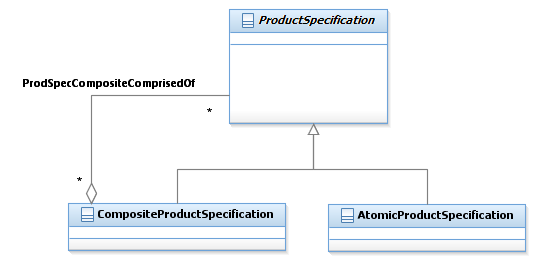


Figure U.13 – Product Specification Composite/Atomic

### CharacteristicSpecification/CharacteristicValue

When constructing any model, it is almost impossible to discover all the possible attributes that characterize a business entity. Even if all the attributes can be found when the model is constructed, additional attributes will be found as a model is extended. Additionally, certain attributes characterize different types (represented by entity specifications) of business entities. For example, one product specification may be characterized by color and size, while another is characterized by bandwidth. The CharacteristicSpecification/CharacteristicValue pattern provides for this type of extensibility and characterization of different entities. The application of the generalized pattern is shown for the Product domain in the figures below.

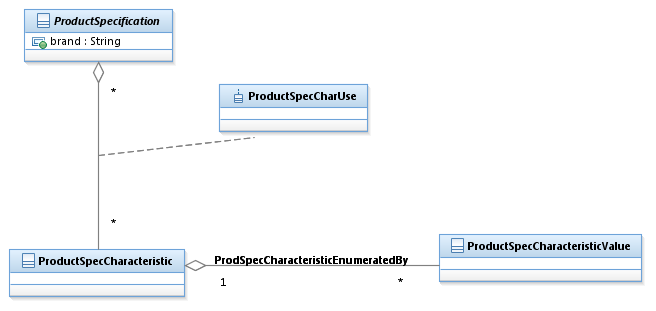


Figure U.14 – Entity Specification Characteristic/Entity Characteristic Pattern – 1

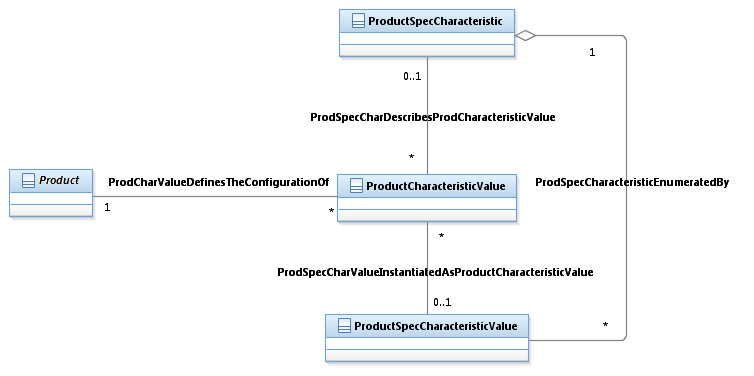


Figure U.15 – Entity Specification Characteristic/Entity Characteristic Pattern – 2

### Entity Classification Group

A Classification Group supports the grouping, organization and analysis of related business management and operational activities according to broad categories, subject areas, and so forth.

The use of Classification groups, which are standardized and agreed across the organization allows for the identification and comparison of like activities, as well as the reduction of uncertainty, ambiguity and duplication.

Classification of information into agreed classification groups provides organizations a mechanism to aggregate, interrogate and manage business information on broad categories rather than on an instance or individual basis.

An additional rationale for grouping apart from the above is that it also allows identification of patterns and common behaviors across groups. It allows for “management on the large”.

Examples of ClassificationGroups used in the telecommunications industry include: Market Segments, Product Portfolios, Distribution Channels, Consumer Revenue, Technology Platform, and so forth.

The figure below shows the ClassificationGroup pattern.



Figure U.16 – Classification Group Pattern

The figure below shows the application of this pattern for the Market Statistic ABE.

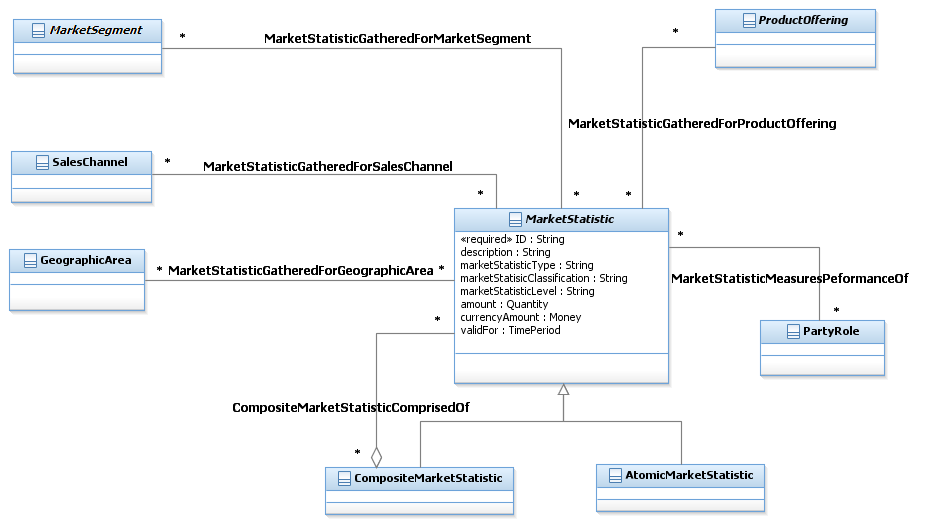


Figure U.17 – Market Statistic Use of Classification Group Pattern

## General Modeling Guidelines

### Best Practices

Any work performed on the Information Framework model should follow best practice modeling guidelines, such as:

* Avoiding the use of multiple inheritance
* Minimal use of association
* When considering the use of an association class to model an entity, a class should be used instead when
  + The entity will inherit from another entity
  + The entity will be sub-classed
  + The entity will be related to other entities not involved in the association
* Describing each artifact is required
* Following the GB922 series of documents with regard to format and content

### General Note about Model Diagrams

Model diagrams used in RSA or reproduced in the various Guide Books do not always display all associations existing in the model for the entities present in the diagrams. It is done this way to keep the diagrams readable.

### Installing RSA

RSA is the modeling tool used by TMForum to specify the Information Model in UML format.

Instructions to install RSA are available in a document which available as part of the normal Information Framework releases. This document is called: “Installing RSA and Using the Confluence SID Model Repository.docx.”

These instructions are intended to be followed by any member working on updates to the Information Framework (SID) model.

Included are instructions for:

* Installing Rational Software Architect (RSA)
* Using the SID Model Confluence repository
* Governance of the modeling work

# Extending the Information Framework

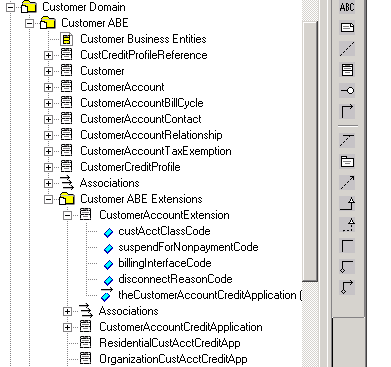
These guidelines should be used when extending an existing ABE. By following them, the existing structure and content of an ABE will not be compromised.

Additionally, if the Information Framework team makes changes to an ABE, adherence to these guidelines should minimize the impact to the extensions. The guidelines include:

* creating packages to hold Information Framework extensions
* adding attributes
* adding new entities
* adding associations

## Creating Packages to Hold Extensions

UML packages should be used to hold extensions to the Information Framework model. There are a number of reasons for this. One reason is so that future versions of the Information Framework can be imported into a model without impacting extensions. Another reason is that it is easy to show the extensions made to the Information Framework model. Packages added should also be defined as control units so that they can be easily moved from model to model. The figure below shows an example of a packaged added to hold extensions to the Information Framework model Customer ABE.



Package to hold new extensions

Figure U.18 – Package Structure For Information Framework Extensions

The package contains all new entities, attributes, and associations that are added to extend the Information Framework model.

### Adding attributes

There are four techniques that can be used to add attributes to an existing entity.

If an entity is not sub-classed, then create a subclass of the Information Framework model business entity to which the attributes will be added. The subclass will inherit all of the attributes and associations from the Information Framework business entity thus maintaining the integrity of the Information Framework model. The new subclass holds the attributes as shown in the figure below.

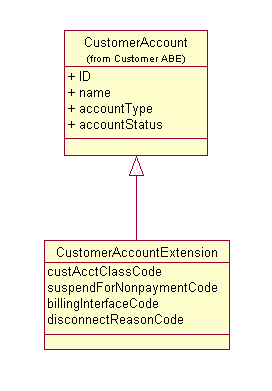


Figure U.19 – Adding Attributes Using Sub-Classing

The name of the business entity that holds attribute extensions is an example. The actual name is at the discretion of the individual extending the Information Framework model. At a minimum, a consistent naming convention should be used.

Another technique can be used when the entity to be extended is already sub-classed. Here the attributes are “inherited” via the association, often referred to as an “IsA” association, implying the extension is a type of ServiceSpecification in the next figure.

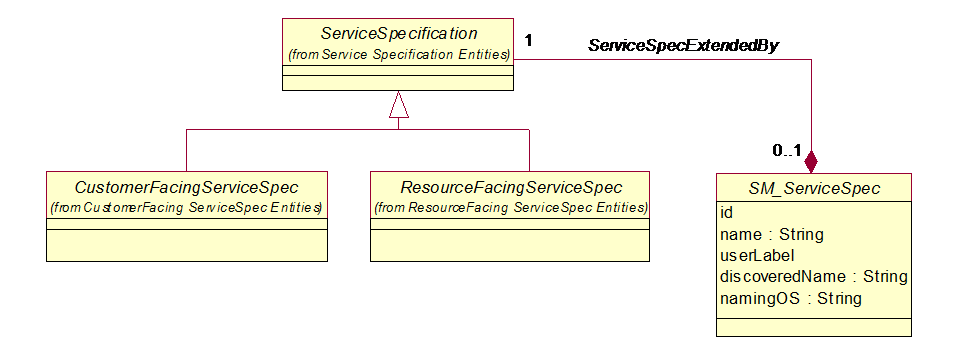


Figure U.20 – Adding Attributes Using Composition

The composition type of association is used to indicate that the life cycle of the framework entity and the entity containing the enterprise specific attributes share the same lifecycle. Where the composition appears in the association is up to the modeler. Here the composition is from the extension to the framework entity implying the key entity is the extension. If the modeler wishes to keep the framework entity as the key entity, then the composition would be from the framework entity. Some modelers prefer to use an aggregation type of association, which is also acceptable.

This is the most stable technique for adding attributes to an existing entity.

The third technique can be employed when there are a large number of extensions to make. For example, one member added extensions to half the framework’s entities, which would have increased the number of entities in the extended framework from about 1200 to 1800, increasing the apparent complexity of the model and making it more of a challenge to manage.

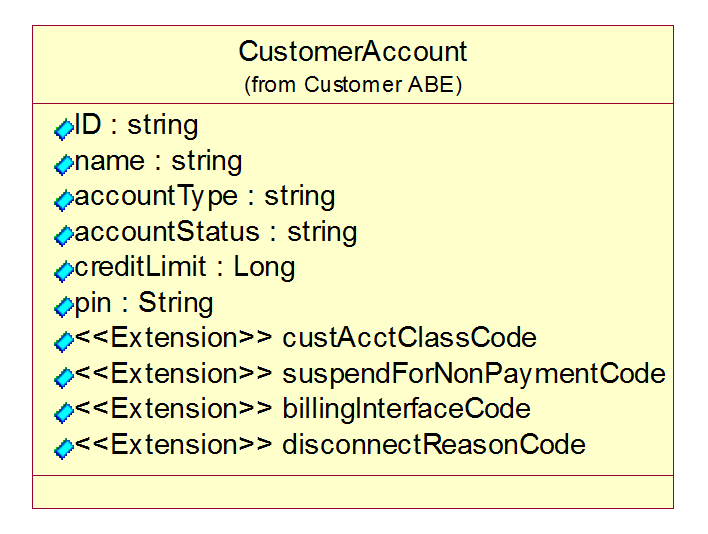


Figure U.21 – Adding Attributes Using Stereotypes

The disadvantage of this technique is that version control would have to be manually performed. For example, if a new version of CustomerAccount is introduced in the framework, either the extensions would have to be re-added or the new framework attributes would have to be manually added. Most framework entities are stable from their initial introduction, so the disadvantage of this technique rarely comes into play.

The fourth technique that can be employed is using the generalized CharacteristicSpecification/CharacteristicValue pattern that enables the dynamic addition of attributes.

### Adding entities

As in adding attributes, new entities should not be added directly to an existing Information Framework ABE package, but rather added to the package that holds the Information Framework model ABE extensions.

Two techniques can be used to add entities to an Information Framework model ABE.

The first technique should be used when attributes as well as a new class need to be added to the Information Framework to extend an existing Information Framework class. In the example below, the new entity (CustomerAccountCreditApplication) needs to be related to an extension of the existing Information Framework CustomerAccount entity (CustomerAccountExtension). This is done by first subclassing the existing Information Framework entity (creating CustomerAccountExtension) and then defining an association (CreditApplicationSubmittedBy) between the subclass of the existing Information Framework entity and the new entity. The figure below shows an example of using this technique.



Figure U.22 – Adding an Entity Related to an Existing Entity – Technique 1

The second technique can be used to add a related entity, such as CustomerAccountCreditApplication, when no attributes have been added to the existing related Information Framework model entity, such as CustomerAccount (there is no CustomerAccountExtension class). The figure below shows an example of using this technique.



Figure U.23 – Adding a Related Business Entity

### Adding associations

The associations that exist among Information Framework model entities should not be changed or deleted. If a new association is required between existing entities, then they can be added.

The association naming guidelines described in this addendum should be used.

# Implementing the Information Framework

This chapter provides guidance for implementing the Information Framework (SID).

## Implementing SID Patterns

This section reviews the current techniques for implementing class hierarchy 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 section that are automatically performed when transforming the logical data model to the physical data model. Many of the considerations described in this section that must be taken into account when transforming the SID UML are then not applicable. These will be pointed out when they are discussed in this section. This chapter’s section on SID and Database Design 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 a starting point for a physical database, 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 section 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 the impact of new releases of the SID with the 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 from 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 U.25 – 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.

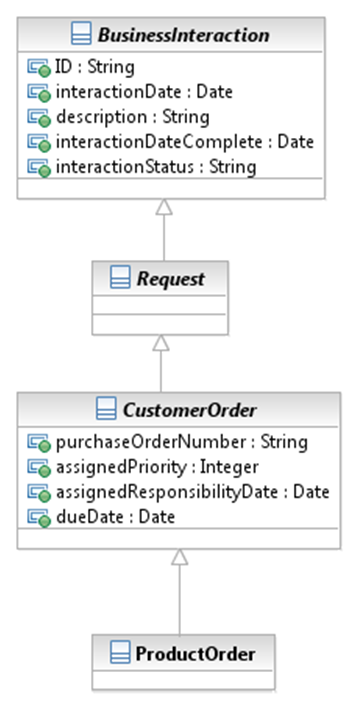


Figure U.25 – Consolidating Sub-Classes – Top Down – Before

In this example, all attributes from the super-classes above the ProductOrder entity are moved to it as shown in Figure U.26 – 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.

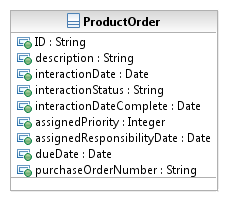


Figure U.26 – Consolidating Sub-Classes – Top Down – After

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 U.27 – Consolidating Sub-Classes – Bottom Up – Before shows a class hierarchy from the Product Specification ABE before consolidation.

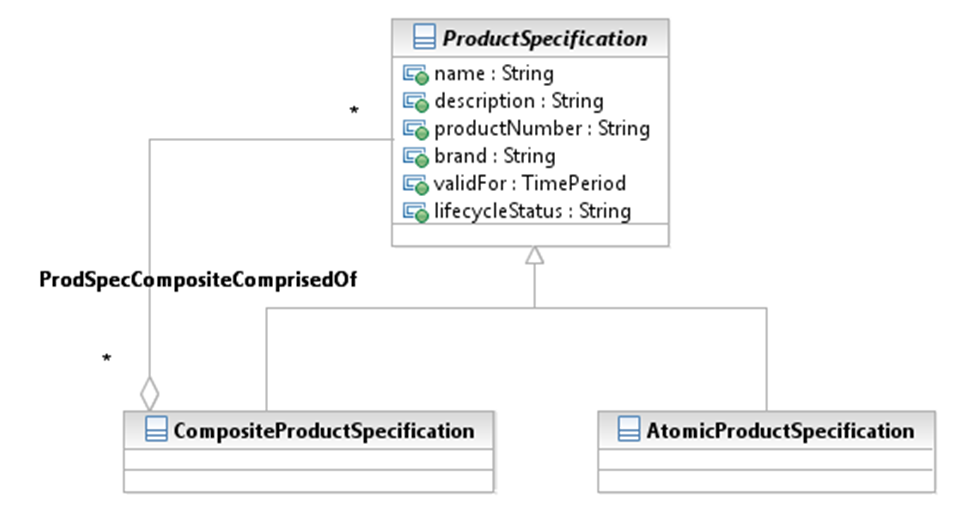
****

Figure U.27 – 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 U.28 – 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 U.28.

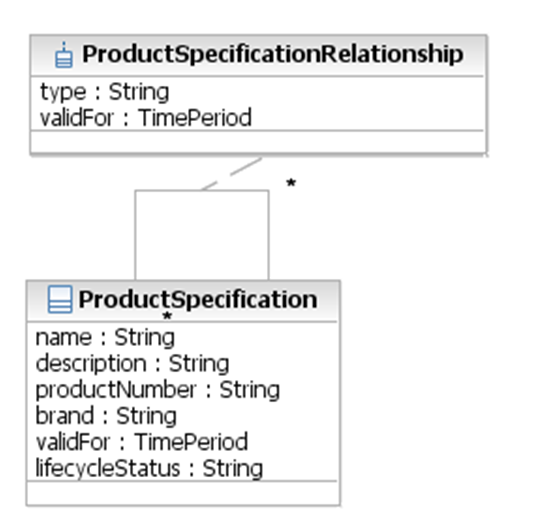


Figure U.28 – Consolidating Sub-Classes – Bottom Up – After

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 U.29 – Consolidating Sub-Classes – Top Down – Bottom Up – Before shows a class hierarchy from the Product Offering Price ABE before consolidation.

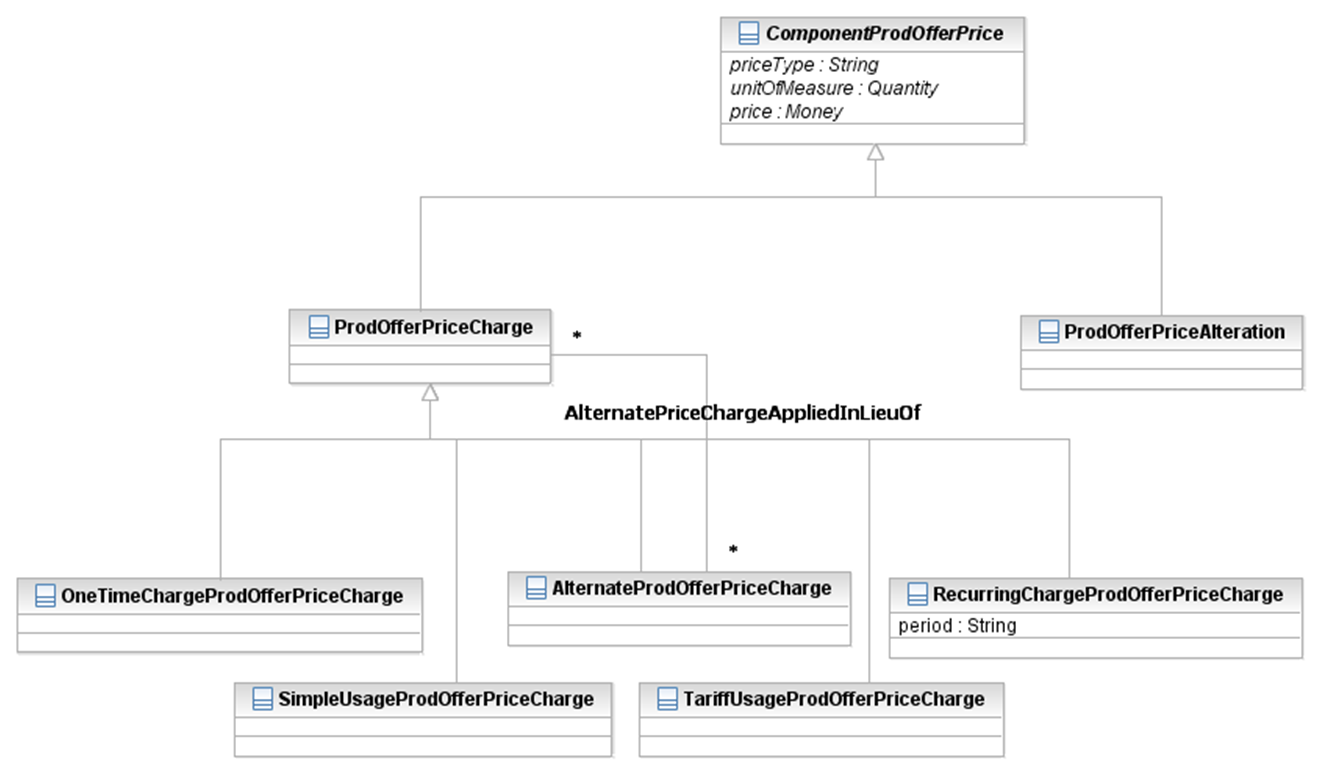


Figure U.29 – Consolidating Sub-Classes – Top Down – Bottom Up – Before

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 U.30 – Consolidating Sub-Classes – Top Down – Bottom Up – After.

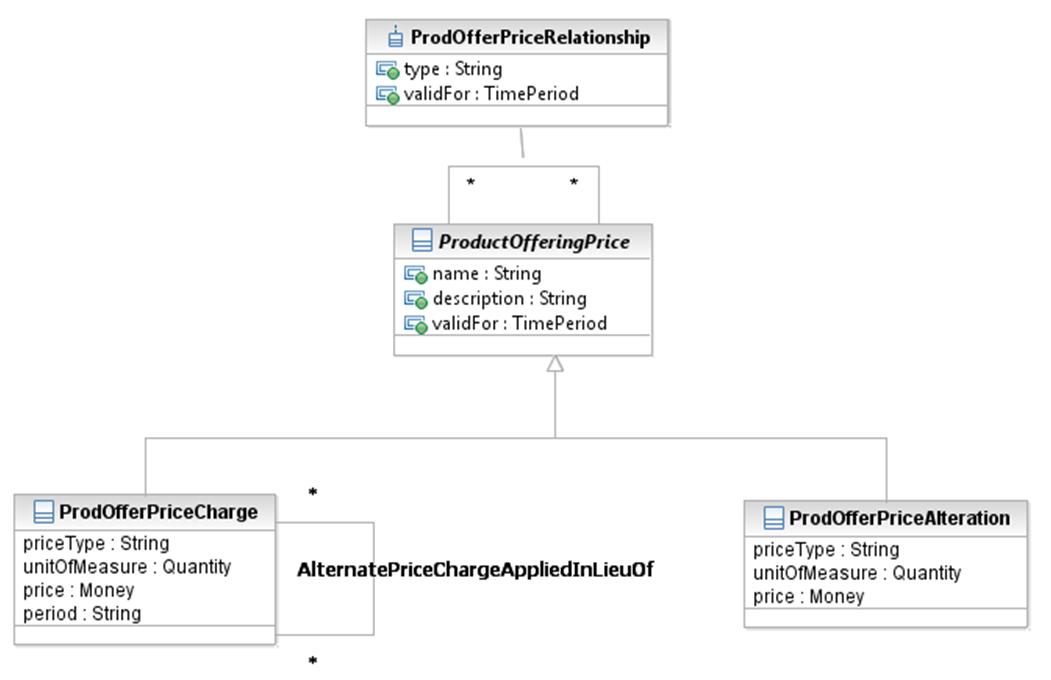


Figure U.30 – Consolidating Sub-Classes – Top Down – Bottom Up – After

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 U.31 – Consolidating Sub-Classes – Typing – Before shows a class hierarchy from the Customer Order ABE before consolidation.

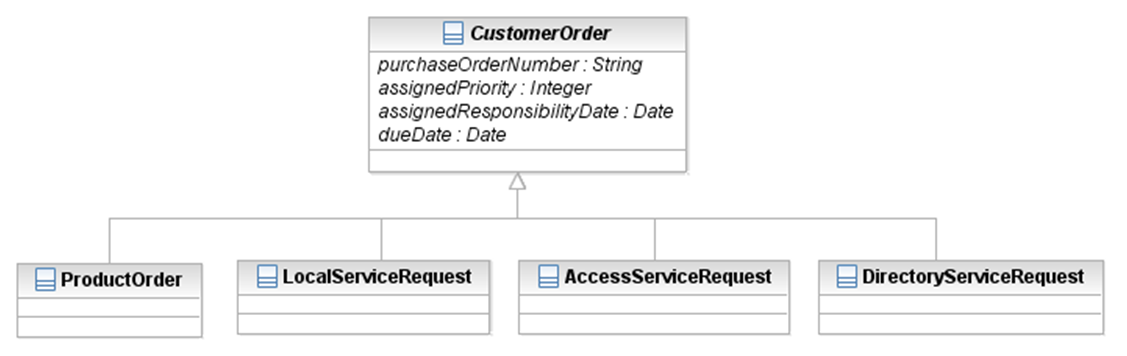


Figure U.31 – Consolidating Sub-Classes – Typing – Before

In this example, a SID modeler is not interested in retaining explicit visibility into the four 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 U.32 – Consolidating Sub-Classes – Typing – After.



Figure U.32 – Consolidating Sub-Classes – Typing – After

The SID and Database Design section in this 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 U.33 – 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 U.33 – Performance Management Class Hierarchy

Figure U.34 – Transformed Performance Management Class Hierarchy shows the resultant model after the transformation.

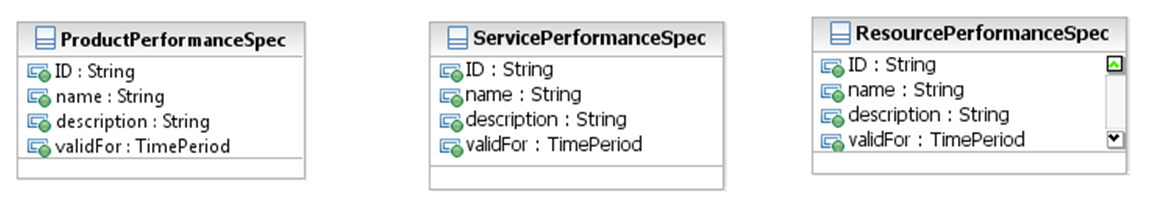


Figure U.34 – Transformed Performance Management Class Hierarchy

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 also be chosen that results in the transformation shown in Figure U.35 – Product and Inventory Performance. In this example, the top-down option results in two entities as shown.

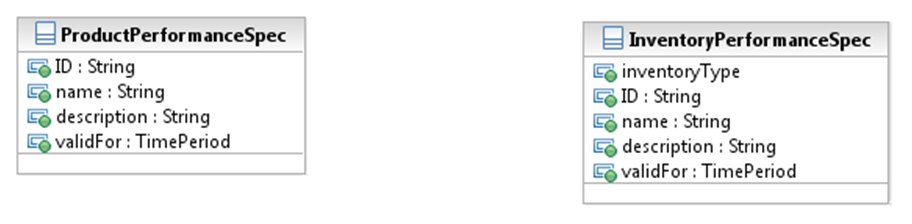


Figure U.35 – Product and Inventory Performance

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 U.36 – 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.

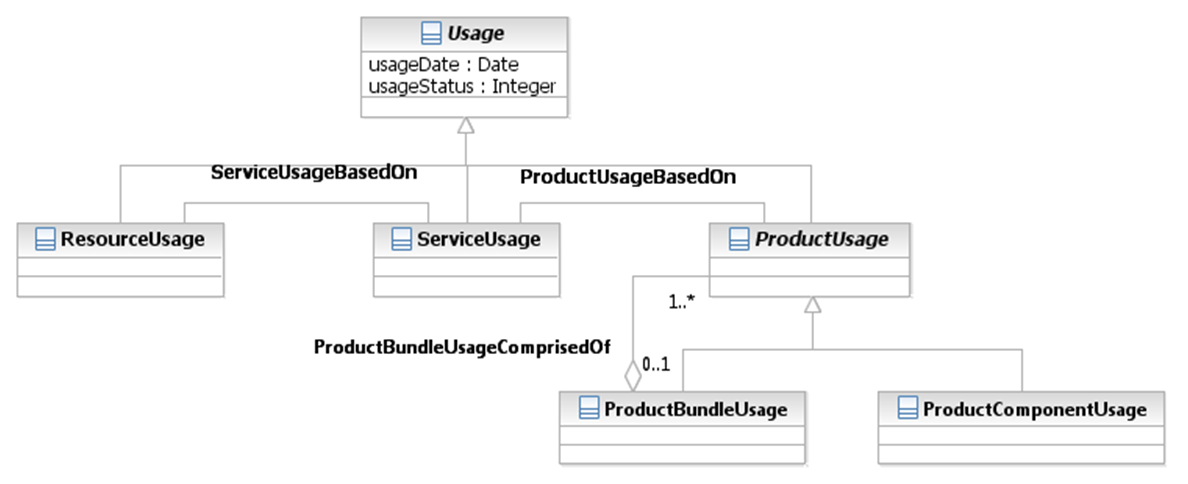


Figure U.36 – Usage Class Hierarchy

Figure U.37 – 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.

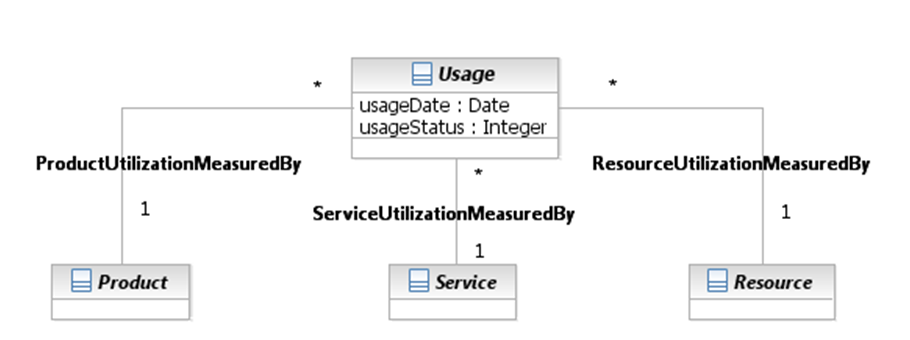


Figure U.37 – Transformed Usage Class Hierarchy

##### 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 be the implementation choice, as shown in Figure U.32 – 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.

#### 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. An object-oriented database implementation is provided in an appendix to this book.

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 database 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 U.28 – 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 changed 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 modelled. 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 SID and Database Design section of this 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 with 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 than 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.

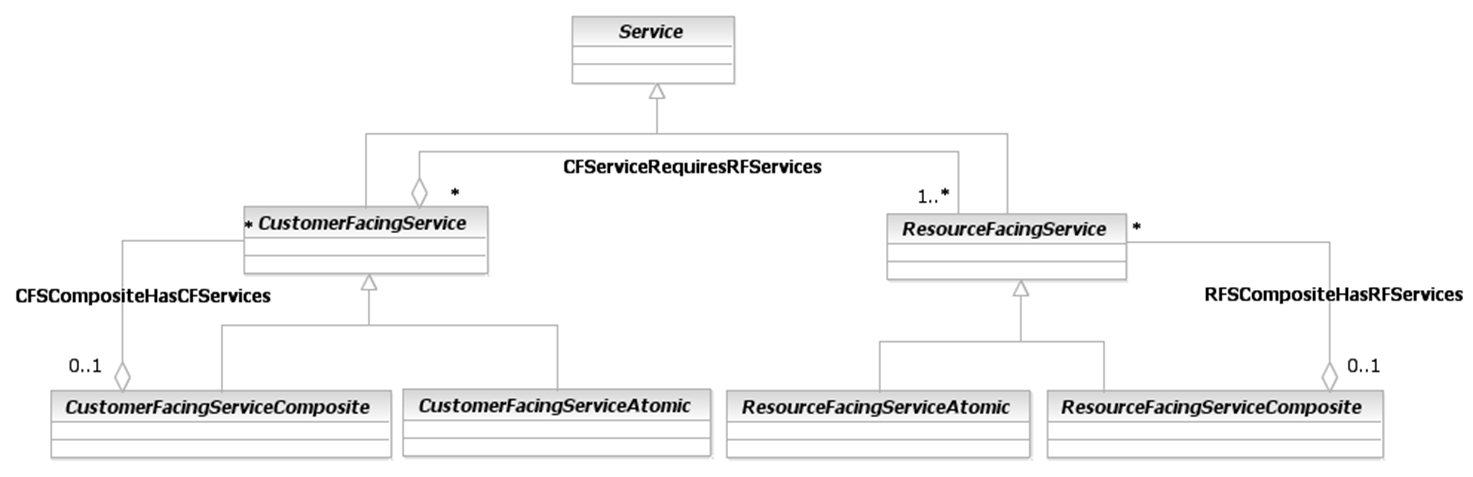


Figure U.38 – Service Hierarchy – Bottom Up – Before

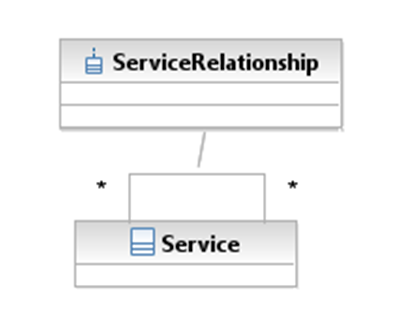


Figure U.39 – Service Hierarchy – Bottom Up – After

This transformation example would allow a CustomerFacingService (CFS) to be required by a ResourceFacingService (RFS) and a CFS to be composed of RFS and vice versa. A possible solution to these problems would be 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 described, 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 U.40 – Party and PartyRole shows the Party and PartyRole application of this pattern.

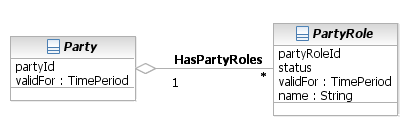


Figure U.40 – Party and PartyRole

Some of the subclasses of PartyRole, such as Customer, ServiceProvider, and Employee, are shown in Figure U.41 – PartyRoles Transformed Into Party Subclasses.

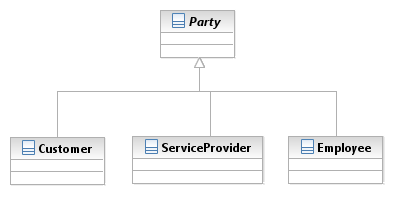


Figure U.41 – PartyRoles Transformed Into Party Subclasses

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 pattern!

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 a new performance monitoring entity that is related to a party role. This could be added as a composite attribute called “PerformanceMonitoring” with the attributes defining it as atomic attributes. But, how is the referential integrity with the associated PartyRole maintained? Atomic attributes could be 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 maintain referential integrity that is done by a database. It 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 modelled 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 modelled 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 section will discuss implementing a single characteristic model from a database perspective.

## SID and Database Design

This section 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 includes class hierarchies or entities involved in one-to-many associations. Included in the section 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 section 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 database 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 be 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 section Implementing SID Patterns. 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, if 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 are 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 U.42 – 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 section 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.

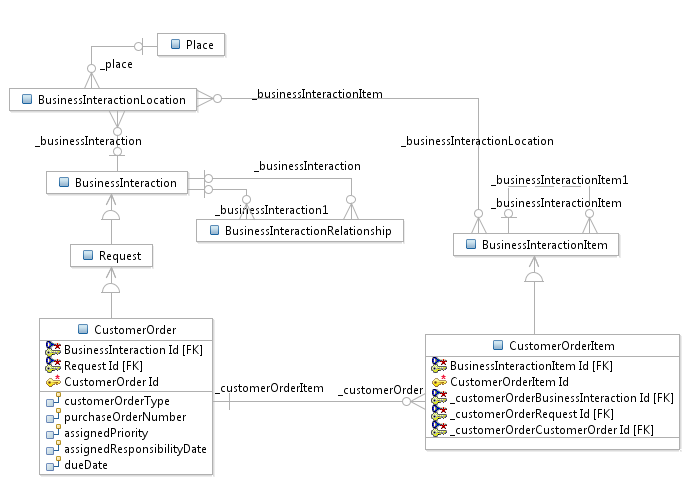


Figure U.42 – Customer Order Logical Data Model Fragment

Figure U.43 – 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. Also, an upcoming figure will show some of the inherited associations.

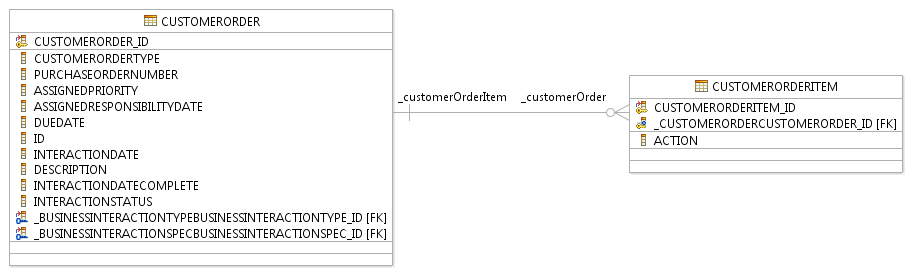


Figure U.43 – Transformed Customer Order Model Fragment

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.

Some tools generate a primary key attribute, shown in the figure as CUSOMERORDER\_ID. It may be desirable to use the ID attribute that was rolled down from BusinessInteraction as the primary key. In this case some tools allow the primary key to be changed to the desired attribute. The generated primary key then becomes a column that can be deleted.

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 database 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 U.44 – Customer Order Inherited Associations shows some of the inherited associations.

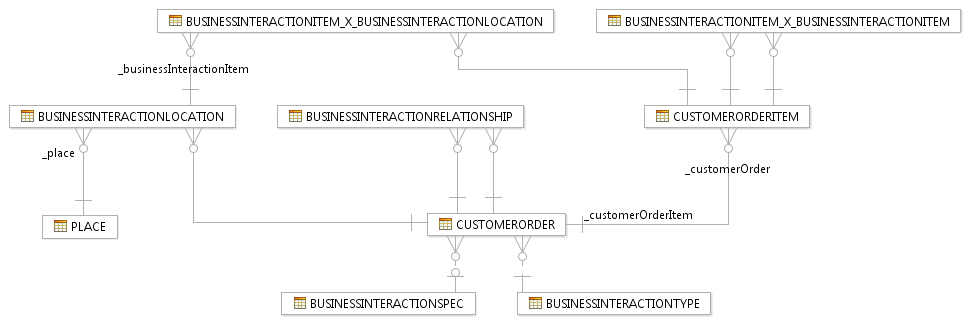


Figure U.44 – Customer Order Inherited Associations

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.

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 Implementing SID Patterns section of this chapter.

Figure U.45 – Product Offering UML Model Fragmentis a typical example of the application of the Composite/Atomic pattern that will be used to demonstrate the bottom-up technique.

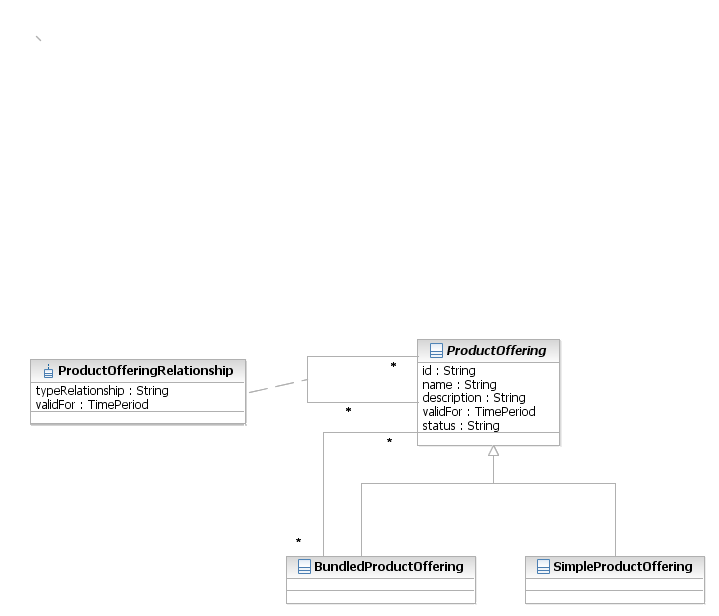


Figure U.45 – Product Offering UML Model Fragment

Figure U.46 – 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.

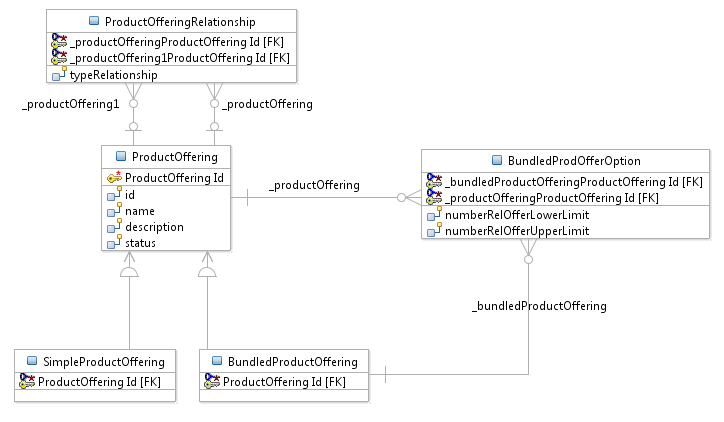


Figure U.46 – Product Offering Logical Data Model

Figure U.47 – 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.

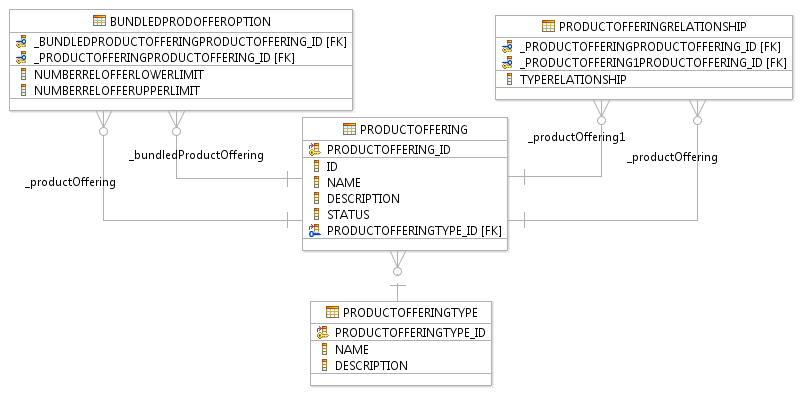


Figure U.47 – Product Offering Physical Data Model

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 ProductOffering 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, examples that depict this technique have not been included in this book.

### 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 Implementing SID Patterns section of this 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

Implementing SID Patterns 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 U.48 – Customer without Party Role shows a very basic implementation of the Customer entity.

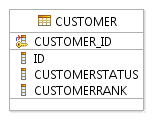


Figure U.48 – Customer without Party Role

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 U.49 – Partial PartyRole and Customer Logical Data Model shows a possible database design if the choice is made to develop a separate Party database.

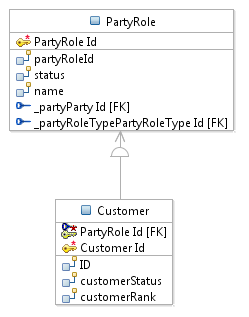


Figure U.49 – Partial PartyRole and Customer Logical Data Model

Figure U.50 – 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 show a possible implementation where a separate Party database has not been developed.

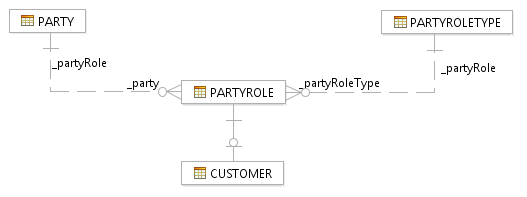


Figure U.50 – Partial Party, PartyRole, and Customer Physical Database

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 U.51 – Customer Logical Data Model.

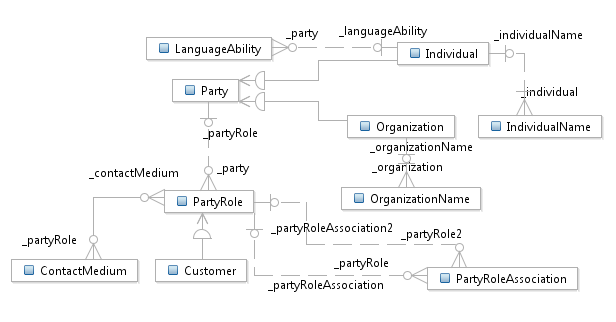


Figure U.51 – Customer Logical Data Model

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 U.52 – Customer Physical Data Model. The figure shows the implementation of the Customer entity, not all the other entities within the Customer ABE.

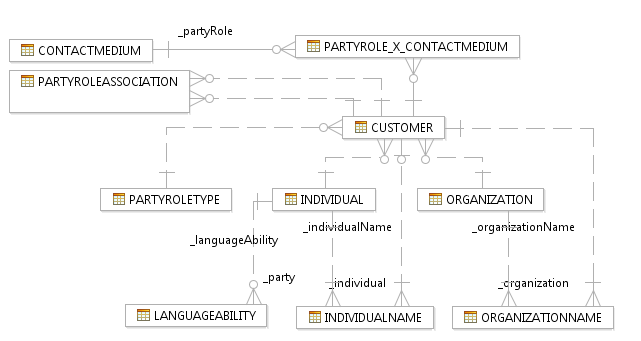


Figure U.52 – Customer Physical Data Model

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, which 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 section Implementing SID Patterns.

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, note that PartyRoleType could represent the EntitySpecification portion of the EntitySpecification/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 EntitySpecCharUse entity in the UML model before any transformations are performed. This does illustrate the point that it is often beneficial 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 U.53– Details of Customer Physical Database Entity shows the Customer entity after its transformation to its physical implementation.

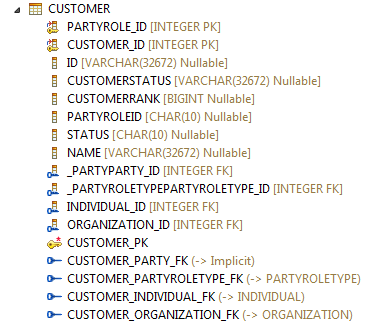


Figure U.53 – Details of Customer Physical Database Entity

In the figure, there are both primary and foreign keys, such as PartyRoleID and PartyPartyID, which 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.

#### 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 U.54 – 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 is shown.

The complete model of characteristic specifications UML model is shown in Figure U.55 – Full Characteristic Specification Model.

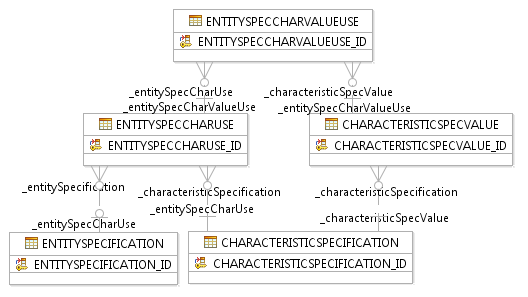


Figure U.54 – Basic Characteristic and “Use” Entities

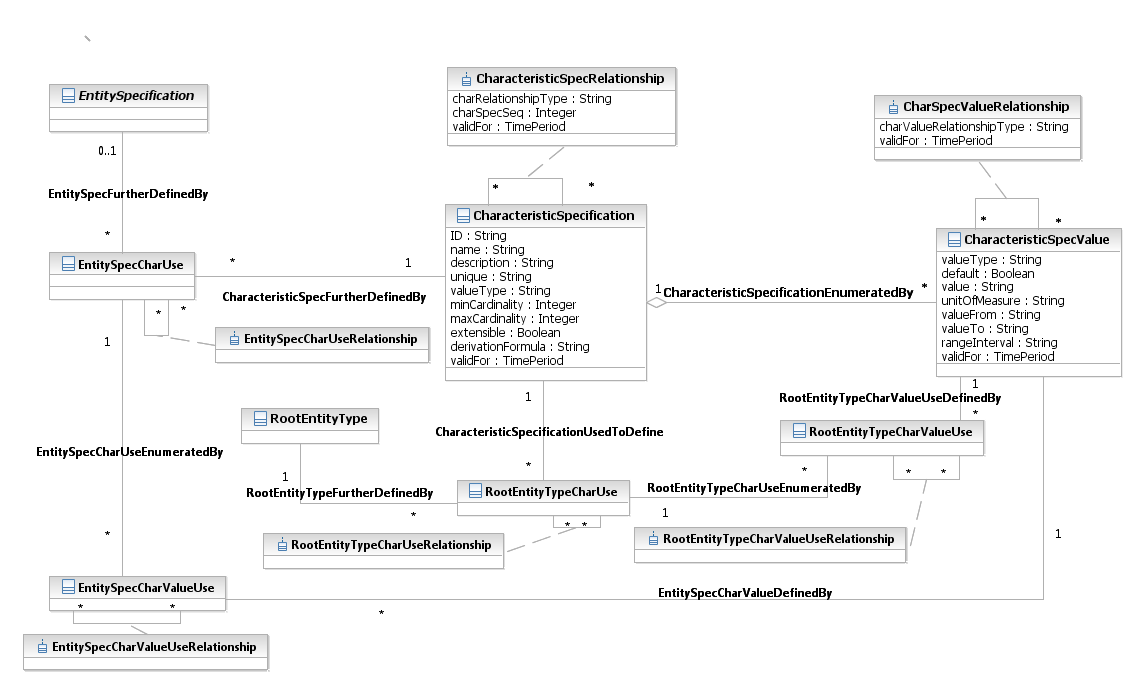


Figure U.55 – Full Characteristic Specification Model

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 shown here.

Figure U.56 – 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.

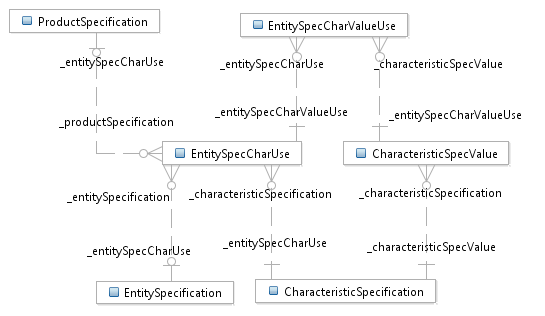


Figure U.56 – Product Specification “Use” of Characteristics

Figure U.57 – Product Specification and Characteristic Keys shows how the ProductSpecification 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 ComponentProdOfferPrice and to ProductOffering need to be “moved” to EntitySpecCharValueUse. This may also mean that a subclass EntitySpecCharValueUse should be created, possibly named ProdDomainCharValueUse.

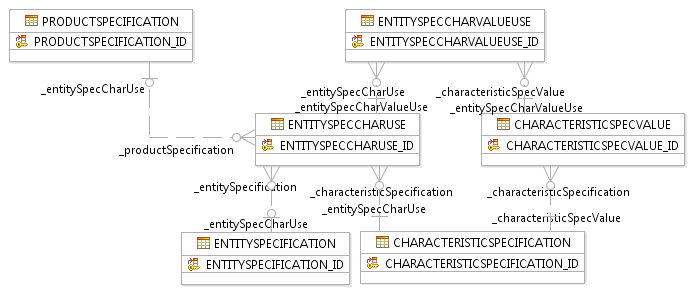


Figure U.57 – Product Specification and Characteristic Keys

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

Figure U.58 – 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.

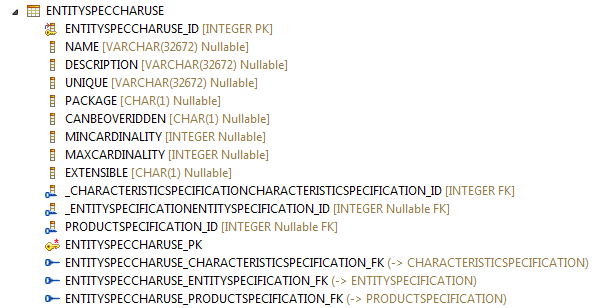


Figure U.58 – Example Physical Entity with Columns and All Keys

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 U.55 – 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 in greater detail later in 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 be prototyped to ensure there are no performance issues. If there are, then refer to the Characteristic Instantiation Option Considerations section below.

##### Characteristic Instantiation Option Considerations

The CharacteristicSpecification/CharacteristicValue section described how a single characteristic model could be used instead of implementing each of the applications of the pattern that appear in the SID information model. Note that the practice of duplicating the pattern for each application has been discontinued since SID release 8.0. Figure U.59 – Product Specification Use of Characteristics repeats the figure that shows part of the generalized characteristic physical data model.

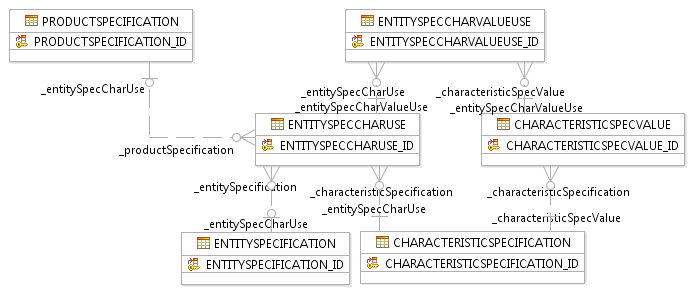


Figure U.59 – Product Specification Use of Characteristics

Implementing the generalized characteristic model would allow instances of Characteristic Specification entities to be shared across domains. For example, a bandwidth characteristic may apply to Products, Services that are used to realize the Products, and the Resources that are provisioned to configure the Services. This would reduce the need to duplicate instances of characteristics across domains.

One consideration that needs to be taken into account is that the SID information model contains cross-domain Characteristic associations. Each domain’s application of the Characteristic pattern includes associations among Product, Service, and Resource domain Characteristic entities. An example of this type of association is shown in Figure U.60 – Inter-Domain Characteristic Associations.

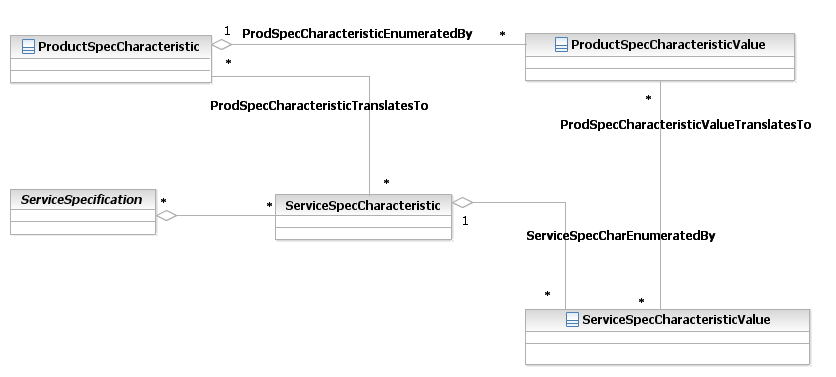


Figure U.60 – Inter-Domain Characteristic Associations

These cross-domain associations provided the ability to “translate” Characteristics in one domain to the equivalent Characteristic in another domain. For example, a premium value for a broadband ProductSpecification speed may translate into the highest bandwidth value for the associated ServiceSpecification.

A generalized implementation of the pattern can still support these associations using the “relationship” entities as shown in Figure U.61 – Relationships Between Characteristic Use Entities.

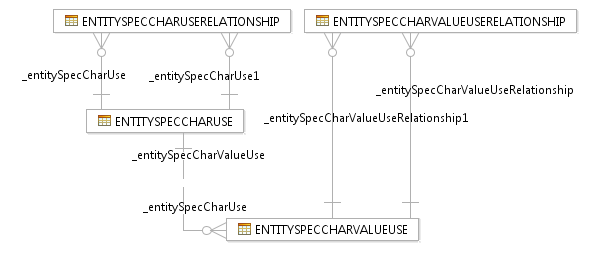


Figure U.61 – Relationships Between Characteristic Use Entities

There may be performance issues that indicate a single implementation should be split in some way. This does not mean that implementation of the entire generalized pattern has to be abandoned. Performance issues may arise for the single CharacteristicValue entity. In this case, the Characteristic specification entities may be kept in a single set of tables, while the CharacteristicValue entity is split into multiple tables, typically by domain. Also, some implementers are concerned about accessing the EntitySpecCharUse entity associated with a specification entity to see if it has any associated characteristics. Adding an attribute to the specification entity that indicates if it has any associated characteristics eliminates this need.

Or, there may be a need based on application boundaries to duplicate this pattern’s implementation across boundaries. For example, there could be separate implementations for domains such as Product, Service, and Resource, particularly if applications are provided by different vendors. In this case each vendor may have its own implementation of the pattern, and the instances of characteristics have to be duplicated. Maintaining cross domain associations is described later in this chapter.

The next section in this chapter will further discuss Characteristic implementation/instantiation options in combination with those for the Product Specification, Product Offering, Service Specification, and Resource Specification ABEs.

### 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 modelled 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 U.62 – TimePeriod as a Related Entity. Note that the validFor attribute is shown as the name of the association between PartyRole and TimePeriod.

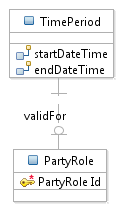


Figure U.62 – TimePeriod as a Related Entity

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 for all entities that reference the Base Type entity.

Another technique involves 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 performs the transformation if the database design tool supports plug-in development.

Figure U.63 – 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.

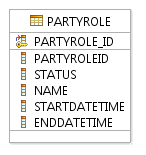


Figure U.63 – Time Period Attributes Transformed As Attributes

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

A section in this chapter 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. A possible implementation of the Contact ABE is shown here followed by a possible implementation of Location ABEs.

Figure U.64 – Contact Medium Logical Database Modelshows 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. They will be rolled up into the ContactMedium entity during the transformation to the physical data model.

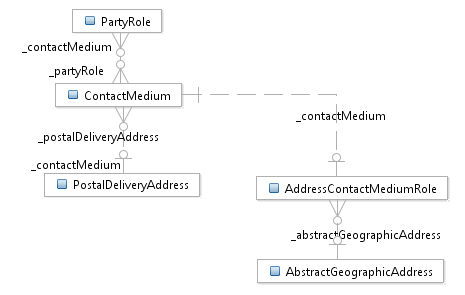


Figure U.64 – Contact Medium Logical Database Model

The transformation of this model to the physical data model is shown in Figure U.65 – Contact Medium Physical Data Model.

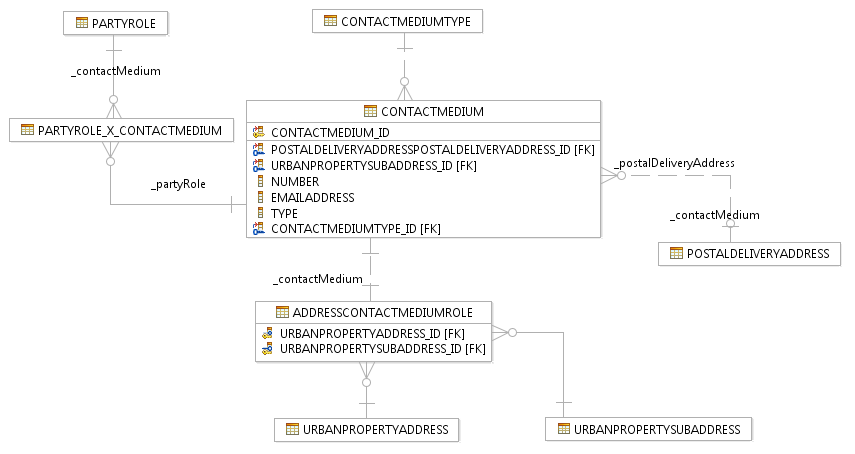


Figure U.65 – Contact Medium Physical Data Model

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, or 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 U.66 – 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.

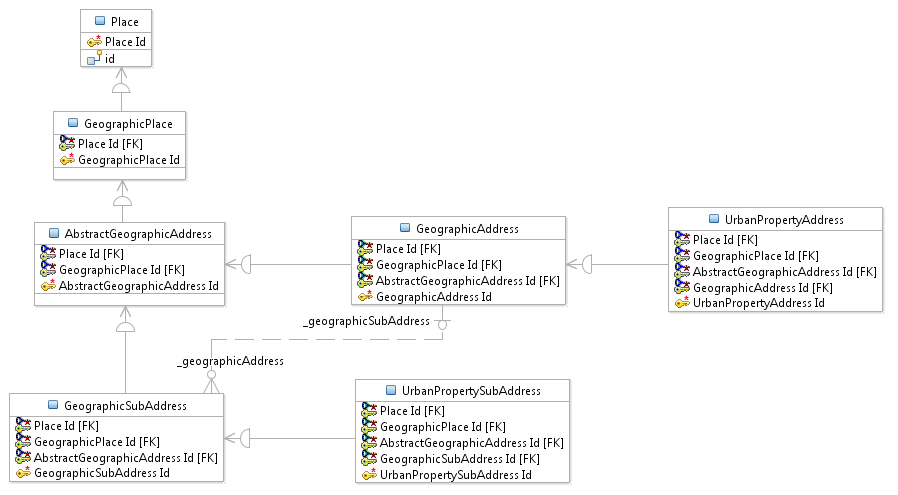


Figure U.66 – Urban Property Addresses Logical Data Model

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

Figure U.67 – Postal Delivery Address Logical Data 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.

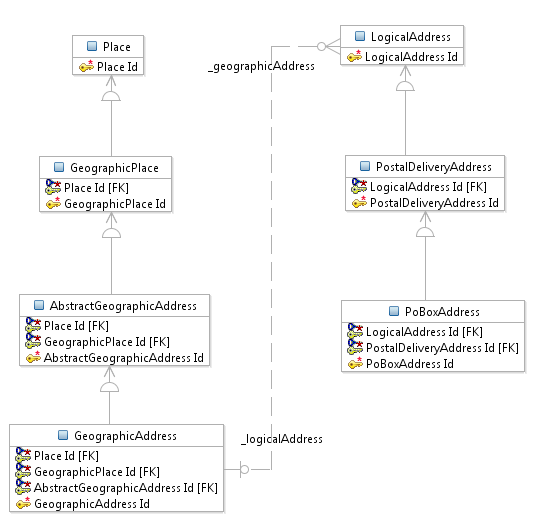


Figure U.67 – Postal Delivery Address Logical Data Model

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 U.68 – Address Physical Data Model shows a much simplified physical data model that contains only three entities.

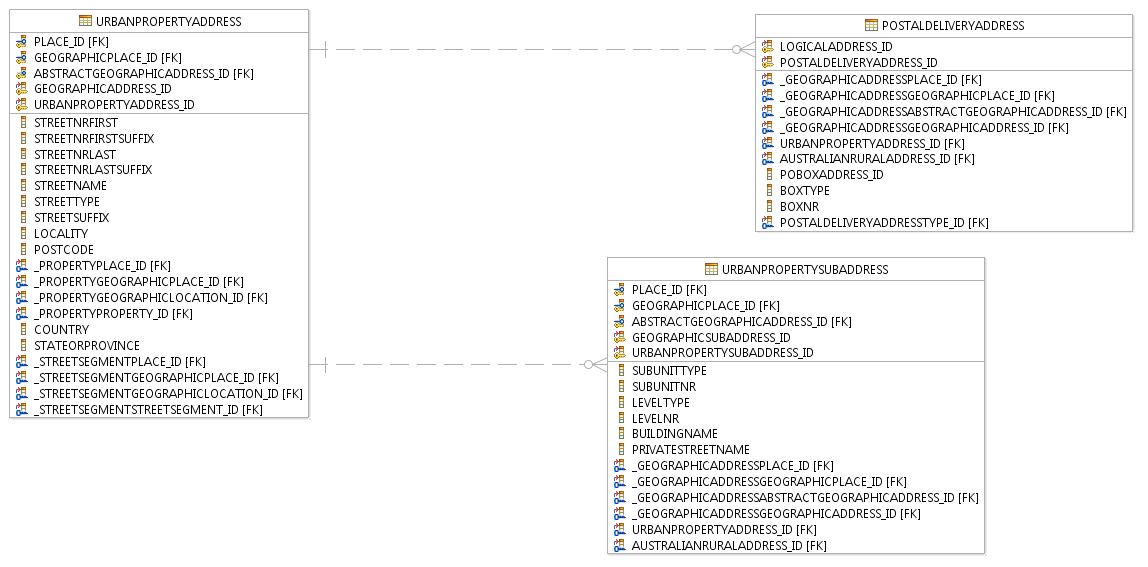


Figure U.68 – Address Physical Data Model

A number of foreign and primary keys can be deleted from the entities, such as PlaceID and 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.

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, or 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 U.69 – SID Product Offering ABE shows the Product Offering Level 1 ABE and its decomposition.

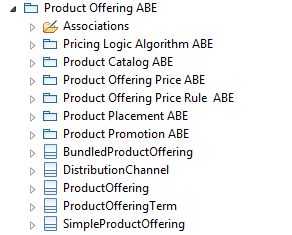


Figure U.69 – SID Product Offering ABE

Often a database 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 Level 1 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 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 modelled 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 have 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 database 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 U.70 – 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.

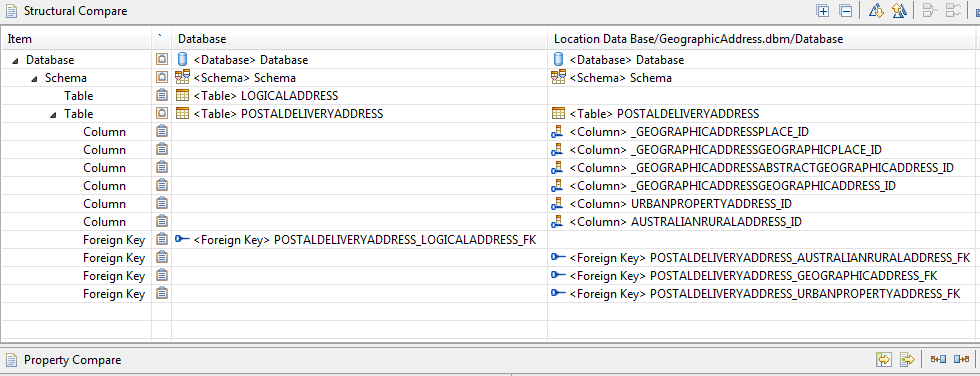


Figure U.70 – Structure Compare of Two Physical Data Models

# Administrative Appendix

This Appendix provides additional background material about the TM Forum and this document. In general, sections may be included or omitted as desired; however a Document History must always be included.

## About this document

This is a TM Forum Guidebook. The guidebook format is used when:

* The document lays out a ‘core’ part of TM Forum’s approach to automating business processes. Such guidebooks would include the Telecom Operations Map and the Technology Integration Map, but not the detailed specifications that are developed in support of the approach.
* Information about TM Forum policy, or goals or programs is provided, such as the Strategic Plan or Operating Plan.
* Information about the marketplace is provided, as in the report on the size of the OSS market.

## Document Life Cycle

This document is being issued for Member Evaluation. The purpose of an Evaluation Version is to encourage input based on experience of members and the public as they begin to use the document. Following the Evaluation Period, documents that are seen to deliver value are candidates for formal approval by the TM Forum. All documents approved by the TM Forum undergo a formal review and approval process.

This document will continue under formal change control. Supporting work will be issued as companions to this document. A document of this type is a “living document,” capturing and communicating current knowledge and practices. Further inputs will be made because of detailed work ongoing in the TM Forum and the industry.

## Document History

### Version History

| **Version Number** | **Date Modified** | **Modified by:** | **Description of changes** |
| --- | --- | --- | --- |
| 0.1 | Sep 2003 |  | First Draft |
| 0.2 | Sep 2003 |  | Second Draft based on comments from sub-team (John Strassner and Josh Salomon) |
| 0.3 | Oct 2003 |  | Additional comments from sub-team |
| 0.4 | Dec 2003 |  | Final review from sub-team, based on other Phase IV deliverable |
| 0.5 | Dec 2003 |  | Updates based on review comments by Cliff Faurer and Wayne Sigley |
| .6 | Feb 2004 | John Reilly | Updated based on SMT review |
| 6.0 | Jul 2005 | John Reilly | Updated based on member contributions. |
| 6.1 | Nov 2005 | Tina O’Sullivan | Converted to new template and corrected various administrative items. |
| 6.2 | Nov 2005 | Tina O’Sullivan | Figure label |
| 6.3 | Nov 2005 | Tina O’Sullivan | Moved fig 5. |
| 6.4 |  | Tina O’Sullivan | Updated notice statement & document status |
| 6.5 | May 2009 | Alicja Kawecki | Minor updates to reflect TM Form Approved Status |
| 8.1 | Mar 2010 | Pascale Pecha | Edits |
| 8.2 | Mar 2010 | Alicja Kawecki | Minor cosmetic corrections for web posting |
| 8.3 | June 2010 | Alicja Kawecki | Updated Notice |
| 8.4 | Dec 2010 | John Reilly | Updates based on change requests |
| 8.5 | March 2011 | Alicja Kawecki | Minor formatting corrections prior to web posting and ME |
| 8.6 | Sep 2011 | Alicja Kawecki | Updated to reflect TM Forum Approved status |
| 8.6.1 | Nov 2013 | Alicja Kawecki | Applied rebranding, updated cover, header, footer & Notice |
| 14.5.0 | Oct 2014 | John Reilly | Removed reference to system view |
| 15.0.0 | Mar 2015 | John Reilly | Removed chapter on XSDs since they are no longer generated  Added Implementing the SID appendix |
| 15.0.1 | May 2015 | Alicja Kawecki | Updated cover and Notice |
| 15.5.0 | Nov 2015 | John Reilly | Added explanation of derived associations to section1.4.5 |
| 15.5.1 | Nov 2015 | Alicja Kawecki | Updated cover, minor cosmetic fixes prior to publishing |
| 15.5.2 | Mar 2016 | Alicja Kawecki | Updated cover, header, footer and Notice to reflect TM Forum Approved status |
| 17.0.0 | Jun 2017 | Cécile Ludwichowski | Added explanation of stereotypes, IDs and attributes multiplicity. |
| 17.0.1 | 29 Jun 2017 | Alicja Kawecki | Applied rebranding and minor cosmetic edits prior to publication for Fx17 |
| 17.0.2 | 16 Nov 2017 | Adrienne Walcott | Updated to reflect TM Forum Approved Status |
| 18.0.0 | 8 May 2018 | Michel Besson | Updated section 2.2.  Updated section 2.4.2 as formally agreed on March 20, 2018.  Added guidelines in section 2.6.  Corrected sections numbering. |
| 18.0.1 | 11-Jul-2018 | Adrienne Walcott | Formatting/style edits prior to R18 publishing |
| 18.0.2 | 10-Oct-2018 | Adrienne Walcott | Updated to reflect TM Forum Approved Status |

### Release History

| **Release Number** | | **Date Modified** | **Modified by:** | | **Description of changes** | |
| --- | --- | --- | --- | --- | --- | --- |
| Release 6.0 | | 31-Oct-2005 | J. Reilly | |  | |
| 8.1 | January,2010 | | | Ken Dilbeck | | Naming changes per Marketing |
| 9.5 | December 2010 | | | John Reilly/John Wilmes | | Updates based on change requests |
| 14.5.0 | October 2014 | | | John Reilly | | Removed reference to system view |
| 15.0.0 | March 2015 | | | John Reilly | | Removed chapter on XSDs since they are no longer generated  Added Implementing the SID appendix |
| 15.5.0 | Nov 2015 | | | John Reilly | | Added explanation of derived associations to section1.4.5 |
| 17.0 | June 2017 | | | Cécile Ludwichowski | | Added explanation of stereotypes, IDs and attributes multiplicity. |
| 17.1 | November 2017 | | | Adrienne Walcott | | Updated to reflect TM Forum Approved Status |
| 18.0.0 | June 2018 | | | Michel Besson | | Updated section 2.2.  Updated section 2.4.2 as formally agreed on March 20, 2018.  Added guidelines in section 2.6.  Corrected sections numbering.  Added section 2.4.3 about Attribute Capitalization Conventions |
| 18.0.1 | 10-Oct-2018 | | | Adrienne Walcott | | Updated to reflect TM Forum Approved Status |

## Acknowledgments

This document was prepared by the members of the TM Forum Information Framework (SID) team.

The Shared Information/Data Model is a genuinely collaborative effort. The TM Forum would like to thank the following people for contributing their time and expertise to the production of this document. It is just not possible to recognize all the organizations and individuals that have contributed or influenced the introduction. We apologize to any person or organization we inadvertently missed in these acknowledgments.

Key individuals that reviewed, provided input, managed, and determined how to utilize inputs coming from all over the world, and really made this document happen were:

|  |  |
| --- | --- |
| **Name** | **Affiliation** |
| Ian Best | TM Forum |
| John Reilly | MetaSolv Software |
| Wayne Sigley | Telstra |
| John Strassner | Motorola |
| Josh Salomon | Amdocs |
| Greg Fidler | Practical Enterprise Architecture P/L |
| Cécile Ludwichowski | Orange |

1. An ABE is an Aggregated Business Entity – a collection of common classes that model a set of related concepts. Please see GB922, Concepts and Principles, for more information. [↑](#footnote-ref-1)