Oracle® Application Integration Architecture - Enterprise Object Library 2.5: UML Profile for CCTS Enterprise Business Objects Guide

Release 2.5

Part No. E15766-01

October 2009

ORACLE

Oracle Application Integration Architecture - Enterprise Object Library 2.5: UML Profile for CCTS Enterprise Business Objects Guide

Part No. E15766-01

Contents

Chapter 1: Introduction 3
Scope for This Document 3
Purpose of This Document 3
Intended Audience 4
Notation 4
Prerequisites 5
Additional Reading 5
How to Read This Document 6
Chapter 2: Definitions of Terms 7
Chapter 3: UML Profile for CCTS Enterprise Business Objects 11
CCTS Meta-Model 11
Partitioning 16
Management Package 17
Common Package 19
DataTypes Package 56
Additional Constructs 63
Other General UML Usage Rules 64
Chapter 4: Oracle CCTS UML Principles and Guidelines 65
Association Types 65
Modeling Patterns 69
Creating Business Information Entities 85
Chapter 5: Oracle UML Packaging Structure (Modularity Model) 89
Top-Level Packages 89
BusinessContext Package 91
EBO Package 91
Common Package 91
Meta 92
CommonEBO Package 92
CommonBusinessComponent Package 92
CommonComponent Package 92
Reference Package 93
PrimitiveType Package 93

DataType Package 93
Verb Package 93
EBM Package 93
CodeList Package 93
Individual Business Component Packages 94
Industry Extensions and Their Packages 95
Versioning 97
Index 103

Chapter 1: Introduction

Overall goals of this document are:

1. To have a consistent Unified Modeling Language (UML) modeling convention (UML profile) of Core Component Technical Specification (CCTS) data model for information exchange; and
2. To be able to use this convention to generate different syntax-specific data structure specifications, which are used in interface development and runtime exchange with respect to specific Oracle Naming and Design Rules (NDR) and infrastructure designs; or
3. To be able to use the convention to represent a logical model of a syntax-specific data structure specification such as XML Schema.

Although the aspiration is to have a UML convention that is independent of any syntax-specific data structure specification, the initial target syntax is the XML Schema. Hence, Oracle Messaging Architecture/Design outlined in the Oracle Enterprise Business Objects/Messages XML Naming and Design Rules (EBONDR) is the first functional target. Additional syntactical data structure specification may be considered in the future.

Scope for This Document

This document defines the conventions for capturing CCTS meta-model artifacts (including Oracle extensions) of a data model using UML class modeling elements. It also provides rules and guidelines for using the CCTS meta-model, deriving meta-model artifacts, and keeping the data model valid with respect to the meta-model. An optional version control practice has also been outlined in the very last section of the document. This practice can also be viewed as requirements for the versioning functionality in future UML modeling, semantic repository tool development, or both.

This version of UML Profile does not cover the Localization Information, Usage Rule, and Component Restriction classes in the CCTS meta-model, except for the enumeration restriction.

Purpose of This Document

The Oracle UML Profile for CCTS Enterprise Business Objects (EBOs) is instantiated to provide a standard-based semantic data modeling. This data modeling is intended to support traditional data models, syntax-specific instantiations of those data models, and syntax-specific business information exchanges – regardless of the platform, operating system, or native language being employed.

Another goal of this profile is to make CCTS-compliant information modeling accessible to a user base through standard UML tool support and to support validation of the structure and semantics of information models against the CCTS. Ultimately, the UML-created based on this profile is intended to be expressed in XML Schema using a transformation, which will serve as the transaction and validation technology governing the exchange of XML business documents.

In addition, the business goals of this specification are:

- To make CCTS compliant information modeling accessible to a broad user base through standard UML tool support.

- To provide a means to visualize CCTS-compliant information model to a broad user base through standard UML tool support.
- To support validation of the structure and semantics of information models against the CCTS.

These goals are achieved through the development of a formal UML profile for CCTS and other UML conventions. This specification is based on the UML Profile for Core Component (UPCC) and implements it with additional Oracle EBO Design/Architecture.

Intended Audience

This document is directed primarily toward developers, users, and implementers of the Enterprise Business Object (EBO) library and related projects. This group of audiences consists of business people, data modelers, business document modelers, business process modelers, and application developers of different enterprises that require common understanding and interoperability of information. The primary purpose of the audience in this group is to be able to correctly use UML constructs to capture CCTS-based data models for information exchange or to be able to comprehend CCTS UML data models created by others.

Notation

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in Internet Engineering Task Force (IETF) Request For Comments (RFC) 2119: "Key words for use in RFCs to Indicate Requirement Levels".

The following list shows specific meanings associated with fonts and words in particular formats.

- **[Definition]** – A formal definition of a term. Definitions are normative.
- **[Example]** – A representation of a definition or a rule. Examples are informative.
- **[Note]** – Explanatory information. Notes are informative.
- [R<X>] or [G<X>] – Identification of a rule (R) or guidance (G) that requires conformance, where <X> denotes an abbreviated name. Rules and guidance are normative. The criterion for characterizing a statement as a rule or guidance is an approximation that the rule is at least partially testable, while it is unlikely that guidance is programmatically testable (needs human judgment or assistance). To ensure consistency across versions of the specification, identifications that are deleted will not be reused.
- `Courier` – All words appearing in courier font are values or objects.
- `"Courier"` – Courier font enclosed in double quotation marks denotes a UML tag name.
- `["Courier1", "Courier2"]` – Courier font enclosed in double quotation marks and square brackets denotes an array of string values. In this document it is either context values or tag values.
- `<<UMLStereotype>>` – Courier font enclosed in double angle brackets denotes a UML stereotype name.
- `<Variable>` – Courier font enclosed in single angle brackets denotes a variable.

- *Italicized* – Italicized font indicates emphasized words or phrases.
- "Meta-model Class Name" – Normal font Words, which first letters are capitalized and which are enclosed in the double quotation marks, refers to a class or an attribute name within the CCTS meta-model.

Prerequisites

UN/CEFACT Core Component Technical Specification version 2.01 (CCTS 2.01) is a basis for this UML profile. It is necessary to understand CCTS terminologies to read this document.

The Oracle Enterprise Business Objects/Messages XML Naming and Design Rules Document (EBONDR) is necessary to understand Oracle business object terminology

Some background knowledge of UML, particularly the Class modeling, is necessary.

Some background knowledge of XML Schema is useful to understand this document, particularly for engineers and implementers.

Additional Reading

Readers of this document may find useful the reading of the following documents in addition to those in the Prerequisites section.

Author	Document Name	URL
UN/CEFACT	UN/CEFACT – TMG BCSS: UML Profile for Core Components based on CCTS 2.01; Candidate for Version 1.0; Working Draft for public review; 2006-10-03	http://www.untmg.org/index.php?option=com_docman&task=view_category&Itemid=137&subcat=2&catid=63&limitstart=0&limit=5
Oracle	Oracle EBO Development Methodology	
ISO	Information Technology – Metadata registries (MDR) – Part 1: Framework International Standardization Organization, ISO 11179-1:Second Edition 2004-09-15	http://standards.iso.org/ittf/PubliclyAvailableStandards/c035343_ISO_IEC_11179-1_2004(E).zip
ISO	Information Technology – Metadata registries (MDR) – Part 2: Classification, ISO 11179-2:Second Edition 2005-11-15	http://standards.iso.org/ittf/PubliclyAvailableStandards/c035345_ISO_IEC_11179-2_2005(E).zip
ISO	Information Technology – Metadata registries (MDR) – Part 3: Registry Metamodel and Basic Attributes, ISO 11179-3(e):Second Edition 2003/Cor 1:2004	http://standards.iso.org/ittf/PubliclyAvailableStandards/c031367_ISO_IEC_11179-3_2003(E).zip
ISO	Information Technology – Metadata registries (MDR) – Part 4: Formulation of Data Definitions, ISO	http://standards.iso.org/ittf/PubliclyAvailableStandards/c035346_ISO_IEC

Author	Document Name	URL
	11179-4:Second Edition 2004-07-15	_11179-4_2004(E).zip
ISO	Information Technology – Metadata registries (MDR) – Part 5: Naming and Identification Principles, ISO 11179-5:Second Edition 2005-09-01	http://standards.iso.org/ittf/PubliclyAvailableStandards/c035347_ISO_IEC_11179-5_2005(E).zip
ISO	Information Technology - Metadata registries: Registration, ISO 11179-6: Second Edition 2005-01-15	http://standards.iso.org/ittf/PubliclyAvailableStandards/c035348_ISO_IEC_11179-6_2005(E).zip

How to Read This Document

Readers, who have good background knowledge of CCTS and only need to be able to read the CCTS UML data model, may read Chapter 2: Definitions of Terms and then read Chapter 4: Oracle CCTS UML Principles and Guidelines. Data modelers who are familiar with the CCTS may skip or quickly skim Definitions of Terms, but they should read the rest of the chapters.

Chapter 2: Definitions of Terms

This section explains the terms used in this document that may be ambiguous and are not formally defined in other prerequisite documents.

Term	Definition
Data model artifacts	All UML classes, attributes, and relations in a CCTS UML data model.
Business object	This term refers specifically to enterprise-level objects. In Oracle Enterprise Object Library (known as EOL), enterprise-level object is called Enterprise Business Object (EBO). Definition of an enterprise-level object—the EBO—is defined in Common Package. A business object needs to be an ACC or ABIE, but the reverse may not be true.
Component	A general term for referring to an ACC, ABIE, BCC, BBIE, ASCC, ASBIE, BCCP, BBIEP, ASCCP, ASBIEP, Choice CC, Choice BIE, Sequence CC, or Sequence BIE that is not a business object as defined previously. Typically, BDT and CDT are not regarded as Component but data types.
Business component	A general term for referring to both the business object and the component as defined previously.
Class	A model element (construct) in UML. ACC, ABIE, ASCCP; ASBIEP, Sequence CC, Choice CC, Sequence BIE, Choice BIE, BDT, CDT; and Primitives are modeled as classes. Consequently, the term “class” in this document in certain usage contexts refer specifically to CCTS artifacts modeled as UML classes.
Object	This term can be confusing because in some object-oriented contexts (particularly object-oriented programming) it refers to an instance (i.e., individual). However, it usually refers to class (i.e., group or set) in the conceptual modeling realm. This document is about conceptual modeling; hence, the word “object” by itself or in combination with other words refers to a concept (i.e., class, group, or set) such as in “object class term,” “object role,” “parent object,” “business object,” and more. It is an intention to be consistent on that use in this document and when specifically refers to an instance, the term “object instance” is used. However, UML itself uses the term “object” to refer to an individual. Consequently, it is unavoidable to have some

Term	Definition
	occurrences of the term “object” to refer to individual as in “UML object,” “Business Context object,” or the <<Object>> stereotype.
Object instance	Instance (i.e., individual) of an object or class (as defined above). However, the term “business object instance” will specifically refer to an instance of the enterprise-level object. In other words, it is an instance of the *business object* as defined earlier.
Class instance	Instance of a class (as defined previously).
Ordinality	The order of UML class attributes kept by internal UML data.
Using class	Class on the association end without an arrow, that is, the class from which an association is directed.
Used-by class	Class on the association end with an arrow, that is, the class where the association is directed to.
Relationship	The relationship between UML classes including association, dependency, and generalization.
Client class	The class on the relationship end without an arrow head, that is, the class from which the relationship is directed. This term subsumes the term “using class.”
Supplier class	The class on the relationship end with an arrow head, that is., the class where the relationship end is directed to. This term subsumes the term “used-by class.”
Oracle CCTS UML data model	This is short for Oracle CCTS UML data model for business information exchange, which means UML class model of business information with respect to the CCTS meta-model and methodology. Creating UML convention for such a model is a primary purpose of this document.
Syntax -specific data structure specification	This refers to specification generated from a CCTS UML data model for computer (and perhaps also human) consumption that can be used as part of business information exchange production and execution. Examples of syntax-specific data structure specification are XML schema, EDI specification, database schema, or Java classes or Beans.
Enterprise Object Library (EOL)	A syntax-specific data structure specification of the Oracle CCTS UML data model using XML schema. EOLs are production releases of the Oracle CCTS UML data model. An EOL is typically said to consist of EBO and EBM definitions, whereas EBMs (enterprise business messages) are EBOs plus operations.
Repository/Library	Storage and maintenance of the Oracle CCTS UML

Term	Definition
	data model.
Universe of Contexts	All context category and values that are known to the Oracle CCTS UML data model.
CCTS 2.01 meta-model	The meta-model as published in the CCTS version 2.01.
CCTS meta-model	The meta-model which includes Oracle extension to the CCTS version 2.01 meta-model. The meta-model is as shown at the beginning of CCTS Meta-Model. It is the meta-model used in this document. The UML profile defined in this document covers this particular meta-model.
Parent class	Oftentimes in object-oriented realm, the term “parent class” refers to the super class (the more general one) in a generalization relationship. However, this term refers to a higher-level aggregate or a higher-level node in a tree in discussions related to data structure. In this document, the term “parent class” is used as in the second sense. That is, it is referring to an associating class from the associated class perspective. That is, it is synonymous with the term “using class” described earlier. Care is taken to use the term “super class” when referring to the first sense described here.

Chapter 3: UML Profile for CCTS Enterprise Business Objects

This section provides normative definitions of the UML profile for CCTS meta-model provided in CCTS Meta-Model. This meta-model is an extension to CCTS 2.01 meta-model. The profile definitions include rules and guidelines which if followed would ensure the compliance of the CCTS UML data model with the CCTS meta-model and the CCTS 2.01 specification itself.

The first subsection gives an overview of the extension of the CCTS meta-model to the CCTS 2.01 meta-model. The next section gives an overview of the UML profile partitioning into packages. The rest of the section provides definitions, rules, and guidelines associated with each UML stereotype in the profile. Mapping to the CCTS meta-model is also provided as part of those details.

CCTS Meta-Model

The CCTS meta-model, as related to this document, provides a conceptual construct for modeling business information at the conceptual as well as context-specific levels. It also provides a framework for harmonization, registration, and maintenance of the information models. The UML profile in this document is based on an extension of the CCTS 2.01 meta-model. Class diagrams in Figure 1 and Figure 3 show the meta-model used in this document, which is based on the CCTS 2.01 meta-model shown in Figure 4 and Figure 5 (the shaded area shows where the changes are affected).

The following list summarizes differences between the CCTS meta-model used in this document and the CCTS 2.01 meta-model.

- Association Core Component Property (ASCCP), Association Business Information Entity Property (ASBIEP), Business Core Component Property (BCCP), and Basic Business Information Entity Property (BBIEP) are explicitly reusable. In the new meta-model, an ACC or ABIE does not own these components but instead owns the ASCC and BCC (through the ACC Property class) or the ASBIE and BBIE (through the ABIE Property class).
- The Sequence and Choice concepts are added. The two concepts allow for modeling of sub semantics as well as alternative representations both at the Core Component (CC) and Business Information Entity (BIE) levels. At the CC level, Sequence CC and Choice CC, Sequence ASCC, and Choice ASCC are provided, while Sequence BIE, Choice BIE, Sequence ASBIE, and Choice ASBIE are provided at the BIE level.
- Change from Core Component Type (CCT) to Core Data Type (CDT) and from Data Type (DT) to Business Data Type (BDT). The change is primarily in making all secondary representation terms in the CCT to also be in the same rank as primary representation terms. Both of them are referred to as Data Type Term within the BDT.
- The Association Type concept is added. Association type is short-hand for writing several commonly used constraints in a conceptual model.
- The structure of Business Context is added. In previous versions of the CCTS meta-model, business context has existed only as descriptive texts. In this meta-model, the business

context has a formal structure that is assignable to BIEs.

- Add the generalization/extension relationship between ACCs and between ABIEs. That is, one ACC may be a conceptual extension from another ACC by adding more properties. Similarly, two ABIEs that are restrictions of two ACCs related by the generalization (i.e., extension) can also have the generalization relationship. ABIEs cannot extend ACCs. Therefore, ABIEs are still strictly restrictions of ACCs.
- The Usage Rule is called out as a separate class from the Registry Class. It is more formally defined with a data structure. It is also applied directly to CCs and BIEs.
- The Component Restriction is called out as a separate class from the Content Component and the Supplementary Component. It is provided with a more formally defined data structure.
- Note that two context categories, including the Official Constraint and Support Role contexts, are excluded from the meta-model. They may be included in the future version as needs arise. The data structure for business context value is also simplified as attributes for their meta-data are not implemented.

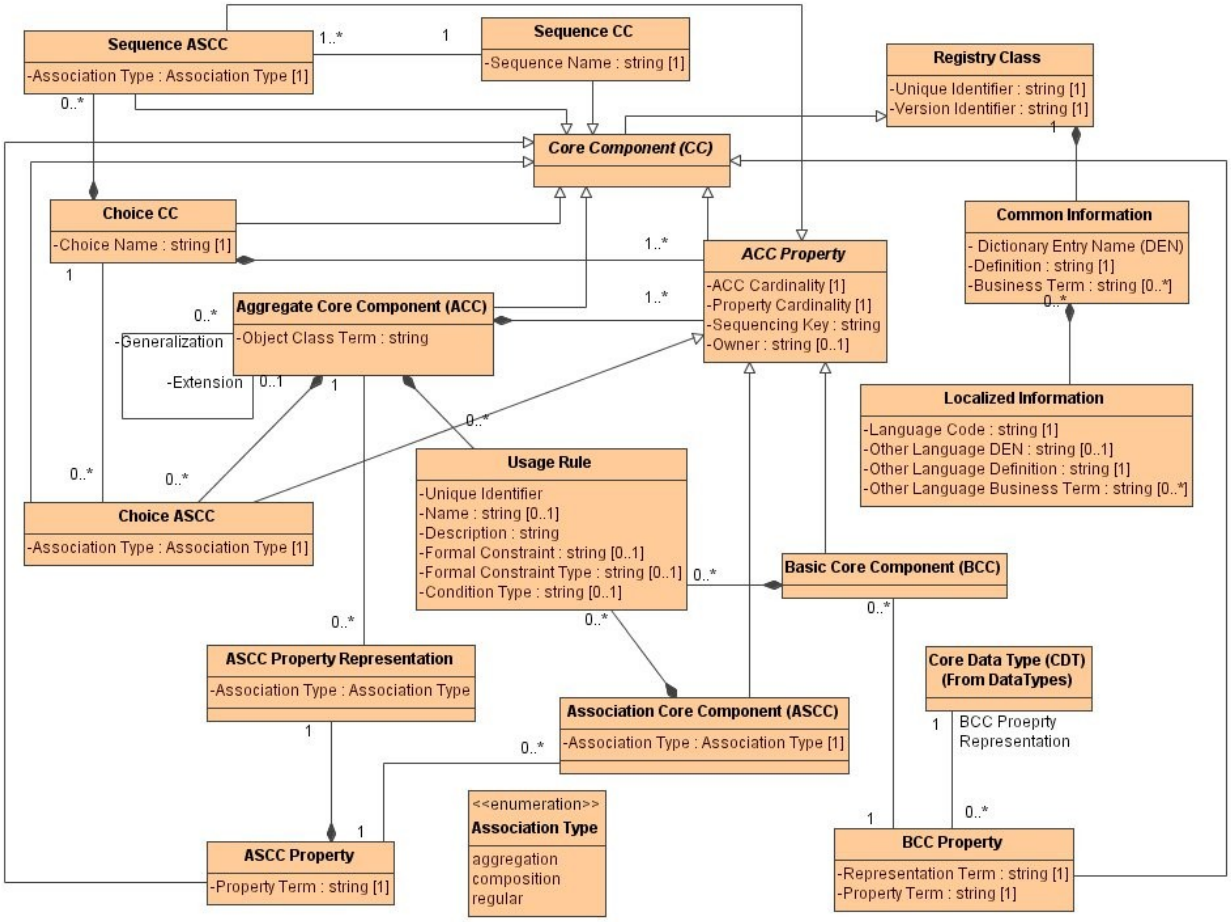

Figure 1: Class diagram showing extended CCTS meta-model related to Core Components

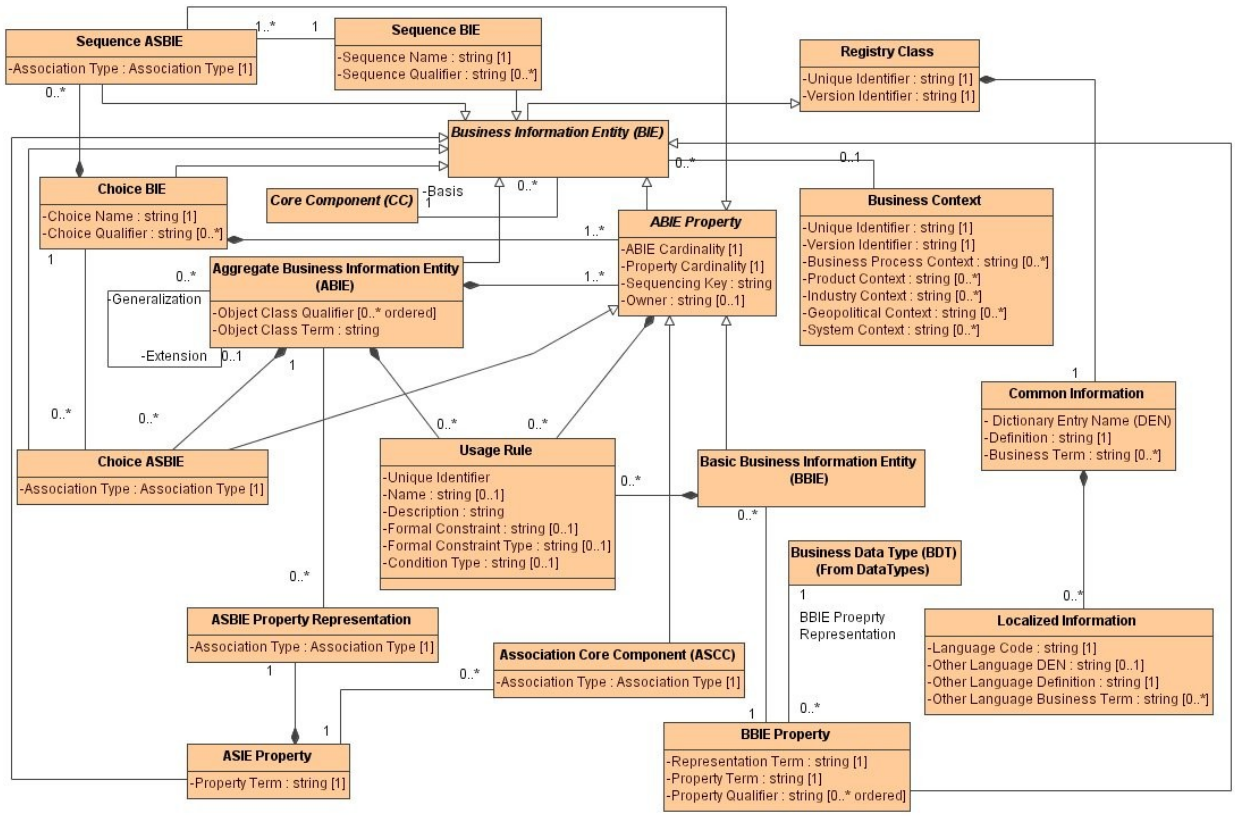

Figure 2: Class diagram showing extended CCTS meta-model related to Business Information Entities

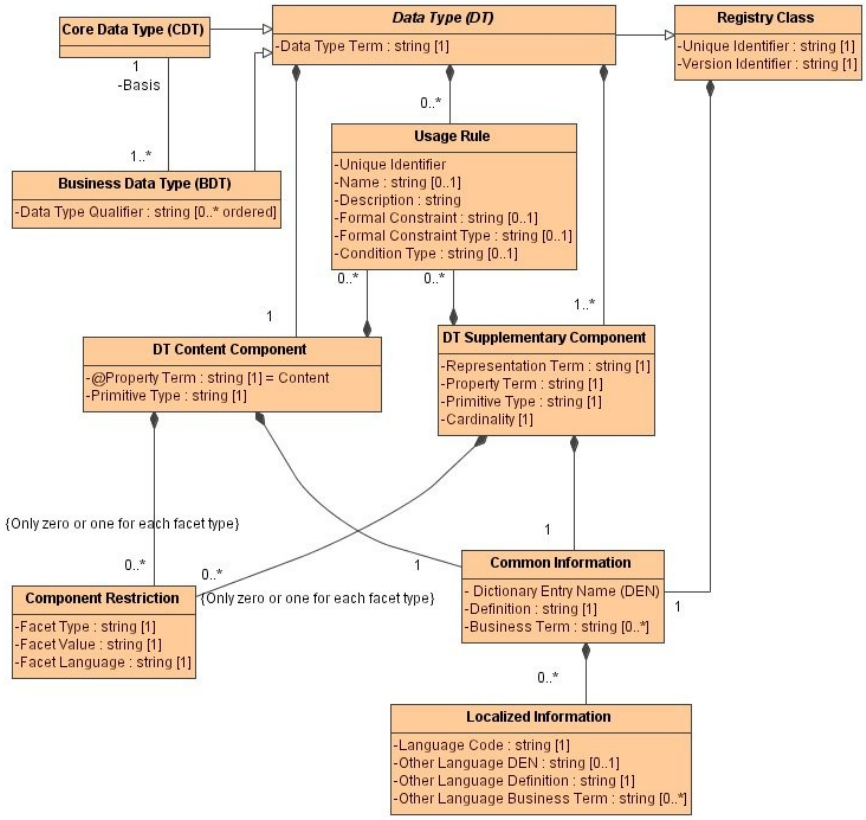

Figure 3: Class diagram showing extended CCTS meta-model related to Data Types

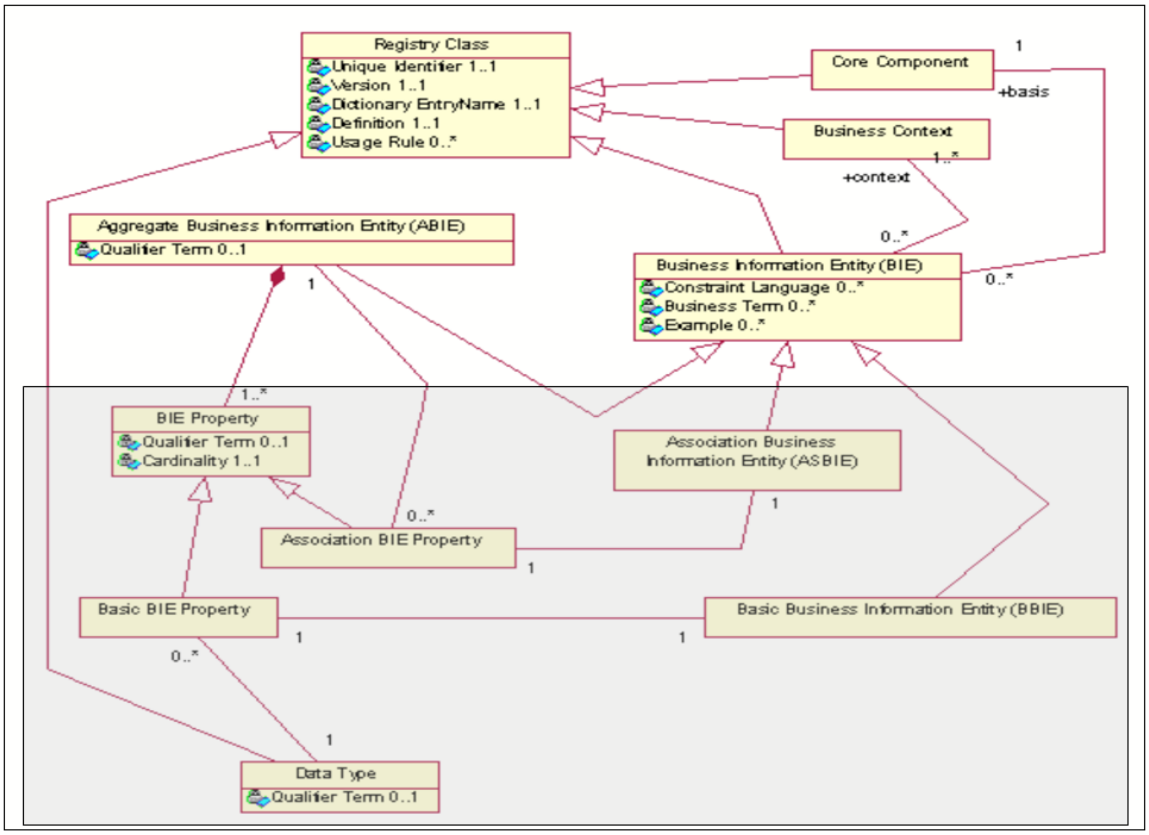

Figure 4: Diagram showing CCTS 2.01 meta-model relating to Core Components and Data Types

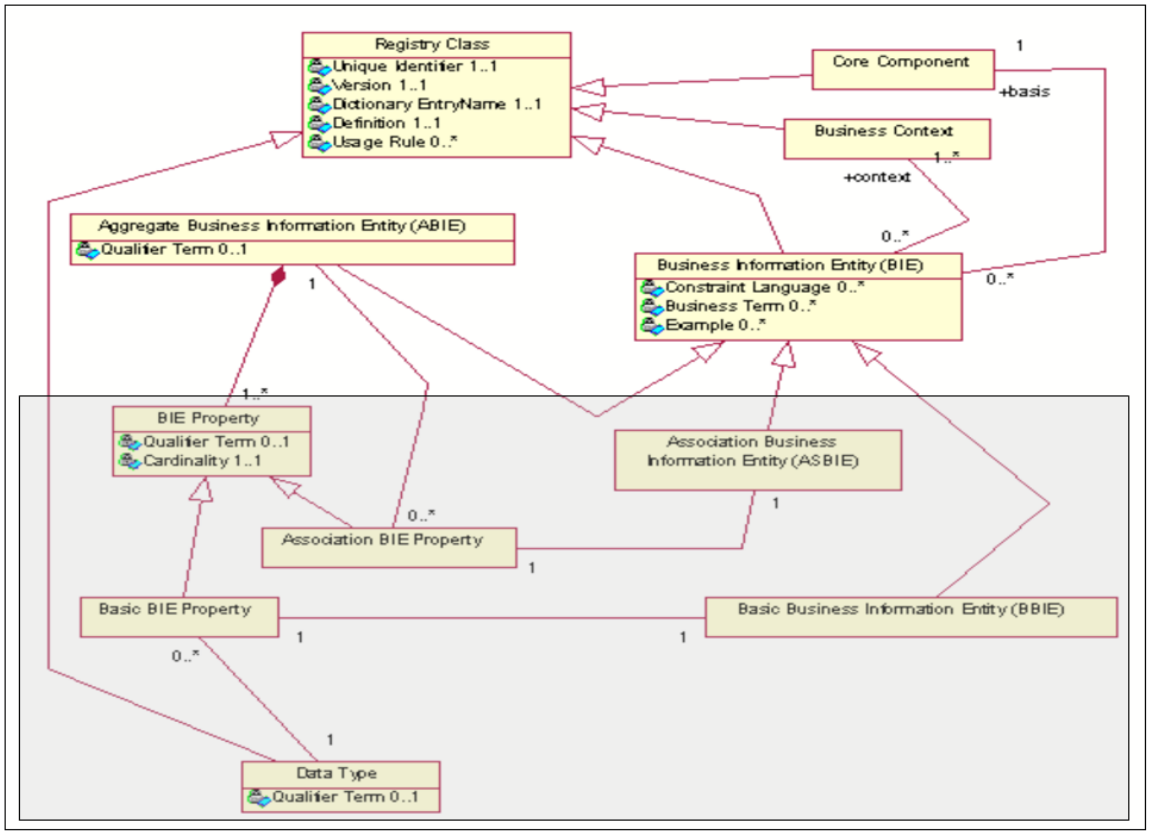

Figure 5: Class diagram showing CCTS 2.01 meta-model related to Business Information Entities

Partitioning

Oracle UML Profile for CCTS meta-model is partitioned into `Management`, `Common`, and `DataTypes` packages as shown in Figure 6. The `Management` package provides an abstract stereotype that includes tags that are common to all stereotypes in the profile. The `Common` package provides a UML Profile for the CCTS Core Component (CC) and Business Information Entity (BIE) constructs. It includes stereotypes for defining business objects and components. The `DataTypes Package` contains a UML Profile for the CCTS Core Data Type (CDT) and Business Data Type (BDT) constructs.

For more information about the DataTypes Package, see DataTypes Package.

It includes stereotypes for defining a set of valid values for UML attributes of Aggregate Core Components (ACCs) and Aggregate Business Information Entities (ABIEs) including Basic Core Components (BCCs) and Basic Business Information Entities (BBIEs).

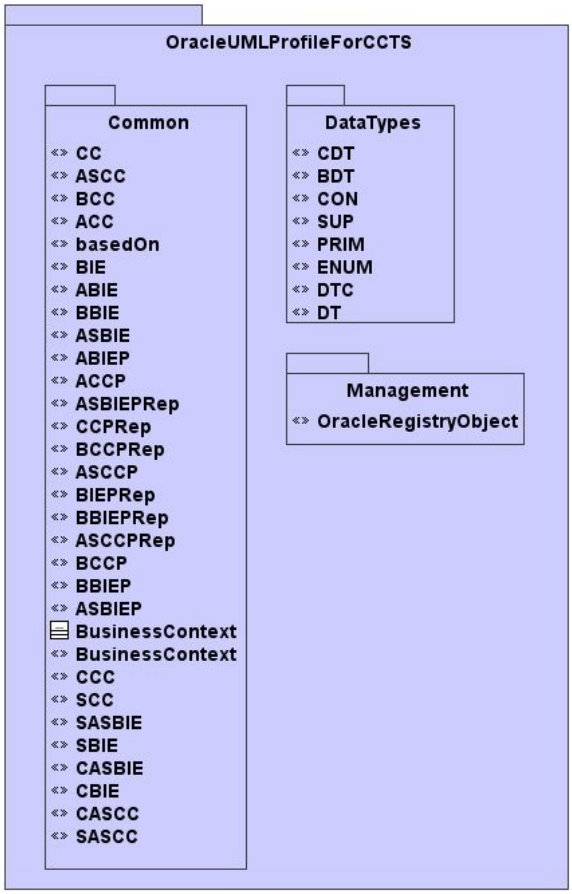

Figure 6: UML Profile for CCTS Meta-model Package Structure

The next three sections provide definitions of stereotypes in these packages.

Management Package

Figure 7 illustrates the <<`OracleRegistryObject`>> stereotype defined in the `Management` package. It provides tags that are common to all stereotypes defined in the Oracle UML Profile for CCTS for the purpose of identification and maintenance of the data model.

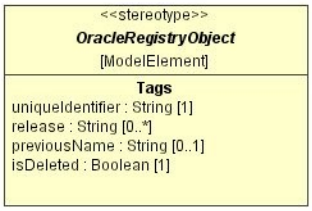

Figure 7: Stereotype definition in the Management Package

The <<OracleRegistryObject>> Stereotype

Stereotype	OracleRegistryObject (Abstract)

<table>
<tr><td>Base Class</td><td colspan="4">ModelElement (from UML 1.4.2 / 2.0 meta-model)</td></tr>
<tr><td>Parent</td><td colspan="4">None</td></tr>
<tr><td>Description</td><td colspan="4">This stereotype corresponds to the “Registry Class” in the CCTS meta-model. It is a superclass to provide tags common to all stereotypes defined in the Oracle UML Profile for CCTS. These tags provide the functionality to maintain versioning and releases of artifacts in the repository (library).</td></tr>
<tr><td rowspan="3">Tag Definitions</td><td>Tag Name</td><td>Type</td><td>Multiplicity</td><td>Description</td></tr>
<tr><td>uniqueIdentifier</td><td>String</td><td>1:1</td><td>This tag corresponds to the “Unique Identifier” attribute of the “Registry Class” in the CCTS meta-model. It serves as a globally unique key to identify artifacts in the Oracle CCTS UML data model. For example, when a business component changes its name, this tag provides the ability to indicate that they are historically the same. Different versions of business components will retain the same “<code>uniqueIdentifier</code>”.

For more information, see Versioning.</td></tr>
<tr><td>release</td><td>String</td><td>0:n</td><td>This is a CVS-like tag to indicate production releases of Oracle Enterprise Object Library (EOL) that are associated with the data model. For example, when generating a syntax-specific data structure specification from the model, a common tag value, such as <code>release1</code>, will be assigned to all classes and relationships in the model.

For more information, see Versioning.</td></tr>
</table>

	isDeleted	Boolean	0:1	This is an indictor to whether a data model artifact has been logically deleted (deprecated) from the current CCTS UML data model. The default value, if not specified, is `false`. **For more information,** see Versioning.

Common Package

The diagram in Figure 8 shows an overview of the stereotype definitions in the `Common` Package. All stereotypes are ultimately based on the <<`OracleRegistryObject`>> in the `Management` package.

Non-abstract stereotypes (those whose names are not italicized) are used with UML classes, attributes, and associations that are created to model Oracle EBOs to comply with the CCTS standard. Tag values are to be assigned for each tag definition according to the multiplicity. The rest of this section provides detailed definitions, usage rules, and guidelines that are associated with these stereotypes.

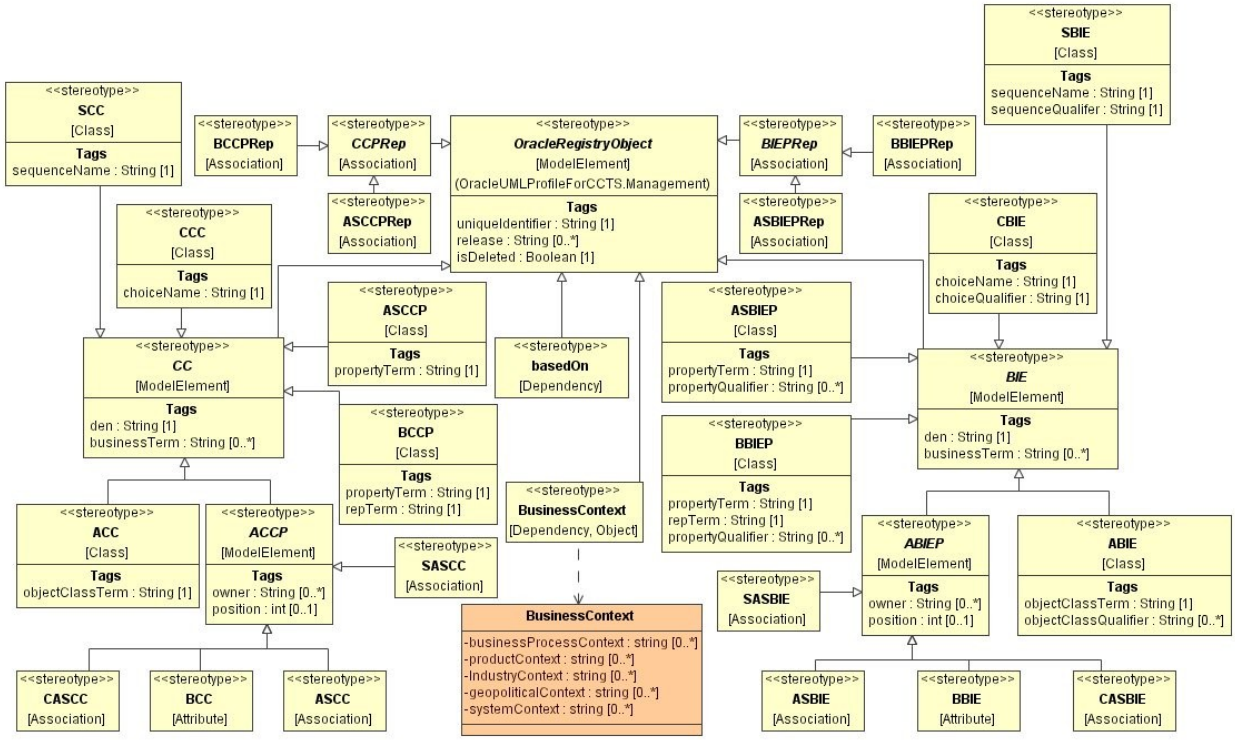

Figure 8: Stereotype definitions in the Common Package

The <<CC>> Stereotype

Stereotype	CC (Abstract)			
Base Class	ModelElement (from UML 1.4.2 / 2.0 meta-model)			
Parent	Management::OracleRegistryObject			
Description	This stereotype corresponds to the "Core Component (CC)" class in the CCTS meta-model. It is a superclass to provide tags common to <<ACC>>, <<ACCP>>, <<BCC>>, and <<ASCC>> stereotypes. A CC represents a harmonized business component that is applicable to a universe of contexts.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	den	string	1:1	This tag corresponds to the "Dictionary Entry Name (DEN)" attribute of the "Common Information" class in the CCTS meta-model. It is a slot for specifying an official name of a business component that the CC represents. See rules provided in each specific stereotype for how the DEN is derived.
	businessTerm	string	0:n	This tag corresponds to the "Business Term" attribute of the "Common Information" class in the CCTS meta-model. It represents other commonly known names/terms of business components that the core component represents that are different from the official name in the "den".

Rules and Guidelines

[GCC] A model element having one of the <<CC>> stereotypes MUST comply with the rules relating to each particular stereotype in the Core Components section of the CCTS 2.01.

[RCCDEN] The "den" tag of a model element having one of the <<CC>> stereotypes MUST comply with the rules relating to DEN of each particular stereotype in the Core Component Rules for Dictionary Entry Name section of the CCTS 2.01.

[GCCBizTerm] The "businessTerm" of a model element having one of the <<CC>> stereotypes MUST comply with the rules relating to the CC Business Term in the Rules for Core Component Business Terms section of the CCTS 2.01.

[RCCDef] A model element having one of the <<CC>> stereotypes MUST have a definition defined in the UML documentation. This UML documentation corresponds to the "Definition" attribute of the "Common Information" class in the CCTS meta-model. The documentation MUST comply with the rules relating to CC Definition in the Core Component Rules for Definitions section of the CCTS 2.01.

The <<BIE>> Stereotype

Stereotype	BIE (Abstract)			
Base Class	ModelElement (from UML 1.4.2 / 2.0 meta-model)			
Parent	Management::OracleRegistryObject			
Description	This stereotype corresponds to the "Business Information Entity (BIE)" class in the CCTS meta-model. It is a superclass to provide tags common to <<ABIE>>, <<ABIEP>>, <<BBIE>>, and <<ASBIE>> stereotypes. BIE is a business component in a specific set of contexts. A BIE is derived from a CC and their relationship is established by using the <<baseOn>> stereotyped dependency relationship. A BIE may be viewed as capabilities and requirements of an integration interface, a connector module, or an underlying application.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	Den	string	1:1	This tag corresponds to the "Dictionary Entry Name (DEN)" attribute of the "Common Information" class in the CCTS meta-model. Its function is the same as that of the <<CC>>. That is, it is a slot for specifying an official name of an associated business component that the BIE represents. See rules provided in each specific stereotype for how the DEN is derived.

	businessTerm	string	0:n	This tag corresponds to the "Business Term" attribute of the "Common Information" class in the CCTS meta-model. It represents other commonly known names of the business component the BIE represents that are different from the official name in the "den".

Rules and Guidelines

[GBIE] A model element having one of the <<BIE>> stereotypes MUST comply with the rules relating to each particular stereotype in the Business Information Entities section of the CCTS 2.01.

[RBIEDEN] The "den" tag of a model element having one of the <<BIE>> stereotypes MUST comply with rules related to the DEN of each particular stereotype in the Rules for Business Information Entity Dictionary Entry Names section of the CCTS 2.01. In addition, the qualifier terms in the DEN SHALL reflect the *functional* restriction of the object class or property term with which they are used.

[GBIEBizTerm] The "businessTerm" of a model element having one of the <<BIE>> stereotypes MUST comply with the rules relating to the BIE Business Term in the Rules for Business Information Entity Business Terms section of the CCTS 2.01.

[RCCDef] A model element having one of the <<BIE>> stereotypes MUST have a definition defined in the UML documentation. This UML documentation corresponds to the "Definition" attribute of the "Common Information" class in the CCTS meta-model. The documentation MUST comply with the rules relating to BIE Definition in the Business Information Entity Rules for Definitions section of the CCTS 2.01.

The <<ACC>> Stereotype

Stereotype	ACC			
Base Class	Class (from UML 1.4.2 / 2.0 meta-model)			
Parent	CC			
Description	This stereotype corresponds to the "Aggregate Core Component (ACC)" class in the CCTS meta-model. ACC is a collection of related business information (properties) that together carry a distinct meaning from each individual property. It is commonly known as components. Some ACCs can also be functioned as business objects. Specifically, an ACC is a harmonized, cross-business-areas business component or concept that makes it functionally applicable to a universe of contexts. A UML class having the <<ACC>> stereotype is regarded as an ACC class.			
Tag	Tag Name	Type	Multiplicity	Description

Definitions	objectClassTerm	string	1:1	This tag corresponds to the "Object Class Term" attribute of the "Aggregate Core Component (ACC)" class in the CCTS meta-model. This tag is a building block for the "den" tag of an ACC class and its UML class name. The object class term conceptually represents the business component being modeled.

Rules and Guidelines

[RACCObj] The "objectClassTerm" MUST comply with rules related to the Object Class in the Core Component Rules for Dictionary Entry Name section of the CCTS 2.01.

[RACCUnique] The "objectClassTerm" MUST retain its uniqueness among ACC classes within a CCTS UML data model. The consequence of compliance with this rule is the uniqueness of the ACC classes' "den" and their UML class names within the data model.

[RACCClassName] UML class name of an ACC class MUST be syntactically derived from its "objectClassTerm" by concatenating all words to create an upper camel case name. For example, if an "objectClassTerm" of an ACC is Sales Order, then its UML class name must be SalesOrder.

[RACCDEN] The "den" of an ACC class SHALL be syntactically derived from the "objectClassTerm" by appending the dot character, a space character, and the word "Details."

For example, if the "objectClassTerm" of an ACC is Sales Order, then its "den" must be Sales Order. Details.

[RACCAttr] An ACC classes MUST NOT contain any UML attribute other than those of the <<BCC>> stereotype.

The <<ACCP>> Stereotype

Stereotype	ACCP (abstract)			
Base Class	ModelElement (from UML 1.4.2 / 2.0 meta-model)			
Parent	CC			
Description	This stereotype corresponds to the "ACC Property" class in the CCTS meta-model. It is a superclass to provide tags common to the <<ASCC>> and <<BCC>>. ACC Property represents a property of an ACC.			
Tag	Tag Name	Type	Multiplicity	Description

Definitions	owner	string	0:n	This tag corresponds to the "Owner" attribute of the "ACC Property" class in the CCTS meta-model. It is used to indicate an ACC owner of an ACC Property (ASCC or BCC) when that property is a reference or repeat of information from another source. **For more information** about the usage of the owner tag, see Oracle CCTS UML Principles and Guidelines.
	position	integer	0:1	This tag corresponds to the "Sequencing Key" attribute of the "ACC Property" class in the CCTS meta-model. Its value indicates the occurrence order of properties in a syntax-specific data structure specification so that the resulting specification is deterministic in every generation.

Rules and Guidelines

[RACCP] An ACC class SHALL contain at least one ACC Property, which SHALL be either a UML attribute with a <<BCC>> stereotype (the BCC attribute as defined in The <<BCC>> Stereotype), a UML association with an <<ASCC>> stereotype (the ASCC association as defined in The <<ASCC>> Stereotype), or a UML association with a <<CASCC>> stereotype (the CASCC association as defined in The <<CASCC>> Stereotype).

[GACCP] Within an ACC class, all embedded BCC attributes, ASCC associations, and CASCC associations SHALL be related to the concept of the aggregate.

[RUniqueACCP] ACC Properties MUST be unique within an ACC. That is, there SHALL NOT be any ASCC association, CASCC association, and BCC attributes whose "den" tags are identical within an ACC class. Alternatively stated, there MUST NOT be more than one association link between any two UML classes.

[RCirACCP] An ACC Property that is an ASCC and CASCC associations MUST be devoid of mandatory circular references. That is, an ACC class SHALL never contain—directly or at any nested level—a mandatory ASCC and CASCC associations for which associated ACC class is the same as itself or a higher level ACC class.

[RACCPPosSem] The ordering of properties through the "position" tags SHALL convey no business semantics. That is, different orderings of properties do not make one ACC class semantically different from another. Logically stated, there SHALL NOT be two ACC classes having the same set of properties.

[RACCPPosUnique] The "position" tag MUST be unique within the set of BCC attributes and within the set of ASCC and CASCC associations under the context of an ACC class (set of ASCC and SASCC association under the context of a CCC class).

For example, if an ACC has 3 BCCs and 2 ASCCs, then the position values among the 3 BCCs must be different and among the 2 ASCCs must be different; however, the two sets may overlap. The positions of the 3 BCCs may be [2, 4, 5] and the 2 ASCCs may be [1, 2]. That is, they only need to be unique and do not necessarily have to be incremental by 1. In fact, the BCC position is not necessary; therefore, this example is for explanation only.

[RACCPPosImpl] An implementation of a syntax-specific data exchange specification generation from the Oracle CCTS UML model SHALL use the "position" tag to order the ACCP associations under the same parent class in ascending order.

[RACCPPos] The "position" tag MUST be specified for every ASCC, CASCC, and SASCC association.

[RACCPOwner] The "owner" tag in an ASCC, CASCC, and SASCC association MUST point to the UML class name of one of the ACC classes having a composition association directed toward the ASCCP class that the ASCC points to or its ancestor ACC class in the composition association chain. The "owner" tag in a BCC attribute MUST point to the UML class name of one of the ACC classes using the BCCP class that particular BCC attribute uses or its ancestor ACC class in the composition association chain.

The <<ASCCP>> Stereotype

Stereotype	ASCCP			
Base Class	Class (from UML 1.4.2 / 2.0 meta-model)			
Parent	CC			
Description	This stereotype corresponds to the "ASCC Property" class in the CCTS meta-model. ASCC Property is a reusable, complex business characteristic. An ASCC Property is associated with an ACC that indicates how the property or the business characteristic is described. Together, they create a business component that is reusable in another ACC. A UML class with this <<ASCCP>> stereotype is regarded as an ASCCP class.			
Tag	Tag Name	Type	Multiplicity	Description

Definitions	propertyTerm	string	1:1	This tag corresponds to the "Property Term" attribute of the "ASCC Property" class in the CCTS meta-model. It is a semantically meaningful name for the characteristic that the ASCC Property represents. Typically, it represents the nature of the association to the associated ACC.

Rules and Guidelines

[GASCCPProp] The "`propertyTerm`" of an ASCCP class MAY consist of more than one word. A multiworded property term of an ASCC Property SHALL have a unique semantic meaning compared to the words separately and compared to any other combination of these words.

[RASCCPUnique] The "`propertyTerm`" MUST retain its uniqueness among ASCCP and BCCP classes within the repository. The consequence of compliance with this rule is the uniqueness of the ASCCP and BCCP classes' "`den`" and UML class name within the repository.

[RASCCPAssoc] An ASCCP class SHALL have an association link with one and only one ACC class. In addition, the association type MUST be unidirectional with an <<`ASCCPRep`>> stereotype pointing from the ASCCP class to the ACC class. The association type on the ASCCP class side MUST be either composition or aggregation; however, the association type on the ACC class side MUST always be regular. Details about the usage of these association types will be given in Oracle CCTS UML Principles and Guidelines.

[RASCCPDEN] The "`den`" of the ASCCP class SHALL be syntactically derived from its "`propertyTerm`" and the "`objectClassTerm`" of the associated ACC class. That is, it is a concatenation of the "`propertyTerm`", a dot, a space character, and the "`objectClassTerm`".

[RASCCPClassName] The UML class name of the ASCCP class SHALL be syntactically derived from its "`propertyTerm`". The name is an upper camel case concatenation of all the words in its "`propertyTerm`" by removing all space characters.

Figure 9 illustrates two examples of ASCCP classes' "`den`" and UML class name. Note that the `_v1` suffix at the end of the class name indicates version of model artifacts, and details about versioning is given in Versioning.

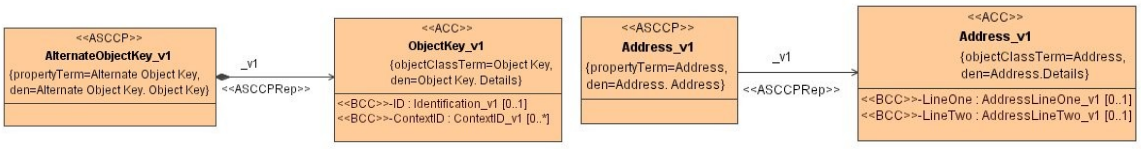

Figure 9: ASCCP naming

The <<ASCC>> Stereotype

Stereotype	ASCC

Base Class	Association (from UML 1.4.2 / 2.0 meta-model)			
Parent	ACCP			
Description	This tag corresponds to the "Association Core Component (ASCC)" class in the CCTS meta-model. ASCC indicates a complex property/characteristic of an associating ACC by establishing an association link with an ACCP. A UML association with the <<ASCC>> stereotype is regarded as an ASCC association.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

Rules and Guidelines

[RASCCDEN] The "den" tag of an ASCC MUST be syntactically derived from the "objectClassTerm", "choiceName", or "sequenceName" of the associating ACC, CCC, or SCC class and the "den" of the associated ASCCP class. That is, it is a concatenation of the "objectClassTerm", "choiceName", or "sequenceName" and the "den" with a dot character and a space character separating the two. The property term section of the ASCCP class's "den" MUST be truncated as follows. The word or words at the beginning of the property term section of the ASCCP class's "den" that is repeated at the end of the "objectClassTerm", "choiceName", or "sequenceName" are truncated. However, the truncation is voided if this results in the whole property term section being truncated.

In the example in Figure 5, the resulting "den" from the "objectClassTerm" - Customer Party and the ASCCP's "den" - Customer Party Billing Profile. Customer Party Billing Profile is Customer Party. Billing Profile. Customer Party Billing Profile. The words "Customer" and "Party" at the beginning of the property term section of the ASCCP's "den" are truncated.

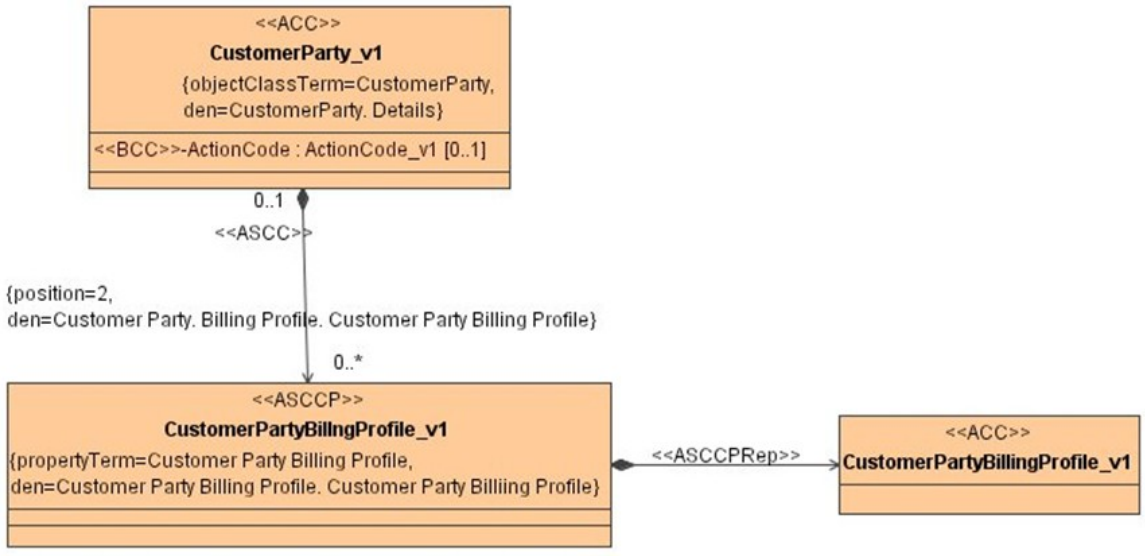

Figure10: ASCC dictionary entry naming

[RASCCAssoc] An ASCC association MUST be a unidirectional association directed from an ACC, CCC, or SCC class to an ASCCP class. The association type on the ACC class end MUST be either composition, aggregation, or regular. Additional details about using of these association types will be provided in Oracle CCTS UML Principles and Guidelines. This association type corresponds to the "Association Type" attribute of the "Association Core Component (ASCC)" class in the CCTS meta-model. The association type on the ASCCP class end MUST be regular.

[RASCCCar] An ASCC association MUST have UML cardinalities specified on both ends. The UML cardinality on the ACC class association end corresponds to the "ACC Cardinality" attribute of the "ACC Property" class in the CCTS meta-model; while the UML cardinality on the ASCCP class association end corresponds to the "Property Cardinality" attribute.

[RASCCUnique] There SHALL NOT be two or more association links between a pair of an ACC, CCC, or SCC class and an ASCCP class. The compliance with this rule ensures unique ASCC class's "`den`" within the repository.

The <<BCCP>> Stereotype

<table>
<tr><td>Stereotype</td><td colspan="4">BCCP</td></tr>
<tr><td>Base Class</td><td colspan="4">Class (from UML 1.4.2 / 2.0 meta-model)</td></tr>
<tr><td>Parent</td><td colspan="4">CC</td></tr>
<tr><td>Description</td><td colspan="4">This stereotype corresponds to the "BCC Property" class in the CCTS meta-model. BCCP is a reusable, singular business characteristic. BCC Property is associated with a class with the <<CDT>> stereotype (core data type in the DataTypes Package), which indicates how the property or the business characteristic is described.

For more information about the DataTypes Package, see DataTypes Package.

Together, they create a business component reusable across ACCs. A UML class with this <<BCCP>> stereotype is regarded as a BCCP class.</td></tr>
<tr><td rowspan="2">Tag Definitions</td><td>Tag Name</td><td>Type</td><td>Multiplicity</td><td>Description</td></tr>
<tr><td>propertyTerm</td><td>string</td><td>1:1</td><td>This tag corresponds to the "Property Term" attribute of the "BCC Property" class in the CCTS meta-model. It is a semantically meaningful name for a unique characteristic that can be used in an ACC object class.</td></tr>
</table>

	repTerm	string	1:1	This tag corresponds to the "Representation Term" attribute of the "BCC Property" class in the CCTS meta-model. It indicates the value space of the BCC Property, that is, how the BCC Property may be described.

Rules and Guidelines

[RBCCPProp] The "propertyTerm" of a BCCP class MAY consist of more than one word. A multiworded property term of a BCCP Property SHALL have a unique semantic meaning compared to each separate word and compared to any other combination of these words. In addition, the "propertyTerm" SHALL NOT contain the following CCTS data type term - "Text," "Numeric," and its derivatives except "Rate" and "Binary Object" and its derivatives.

[RBCCPUnique] The "propertyTerm" MUST comply with the rule [RASBIEPUnique].

[RBCCPAssoc] A BCCP class SHALL have an association link with one and only one CDT class (a class with <<CDT>> stereotype defined in the DataTypes Package).

For more information about the DataTypes Package, see DataTypes Package.

In addition, the association MUST be a unidirectional association with a <<BCCPRep>> stereotype directed from the BCCP class to the CDT class. The association type MUST be composition on the BCCP class end and regular on the CDT class end.

[RBCCPRep] The "repTerm" tag of the BCCP class SHALL be the "dataTypeTerm" of the CDT class with which it is associated.

[RBCCPDEN] The "den" of the BCCP class SHALL be syntactically derived from its "propertyTerm" and its "repTerm" (note that the "propertyTerm" is a data modeler decision based on its semantics and description). That is, the "den" is a concatenation of the "propertyTerm", a dot character, a space character, and the "repTerm".

[RBCCPClassName] The UML class name of the BCCP class SHALL be syntactically derived from its "propertyTerm". The name is an upper camel case concatenation of all words in its "propertyTerm" by removing all space characters. In addition, if the word "Identifier" or "Identification" presents in the "propertyTerm", abbreviate it to ID.

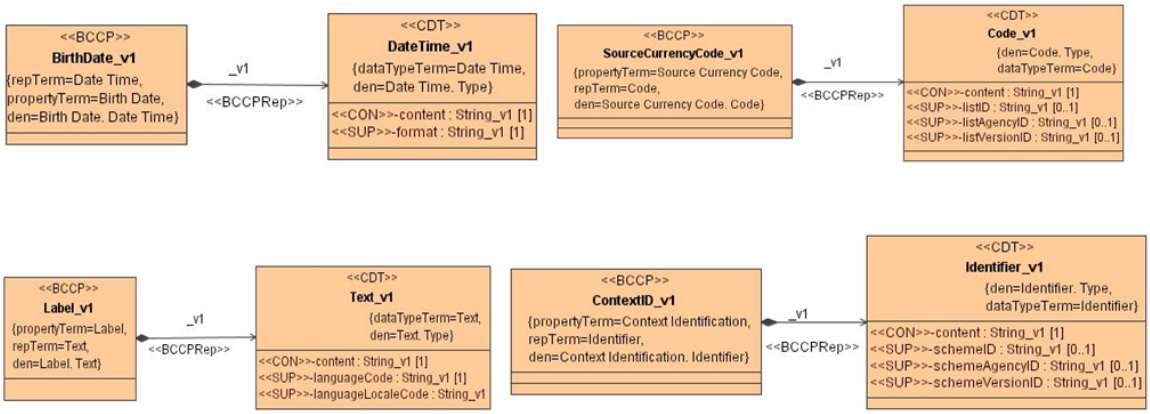

Figure11: BCCP naming

Figure 11 illustrates four examples of BCCP classes' "`den`" and UML class name. In particular, the "`propertyTerm`" of the `Label` BCCP class does not include the term "Text," and the term "Identification" in the "`propertyTerm`" of the `ContextID` BCCP class is abbreviated to `ID`.

The <<BCC>> Stereotype

Stereotype	BCC			
Base Class	Attribute (from UML 1.4.2 / 2.0 meta-model)			
Parent	ACCP			
Description	This stereotype corresponds to the "Basic Core Component (BCC)" class in the CCTS meta-model. BCC indicates a singular business characteristic of an ACC by using a BCCP class as its type. A UML attribute with this <<`BCC`>> stereotype is regarded as a BCC attribute.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

Rules and Guidelines

[RBCCType] The UML type of a BCC attribute SHALL be one of the BCCP classes.

[RBCCDEN] The "`den`" of a BCC attribute SHALL be syntactically derived from the "`objectClassTerm`", "`choiceName`", or "`sequenceName`" of the parent ACC, CCC, or SCC class and the "`den`" of the BCCP class it uses (as the UML attribute type). That is, it SHALL be a concatenation of the "`objectClassTerm`", "`choiceName`", or "`sequenceName`" and the "`den`" with a dot character and a space character separating the two. The property term section of the BCCP class's "`den`" MUST be truncated as follows. The word or words at the beginning of the property term section of the BCCP class's "`den`" that are repeated at the end of the parent's "`objectClassTerm`", "`choiceName`", or "`sequenceName`" are truncated. However, the truncation is voided if this results in the whole property term section of the BCCP class's "`den`" being truncated.

For example, if the "objectClassTerm" of an ACC class is Identification and "den" of the BCCP class it uses is Identification. Identifier, then the resulting "den" of the BCC attribute is Identification. Identification. Identifier, that is, no truncation occurs. In another example, if the "objectClassTerm" of an ACC class is Address and "den" of a BCCP class is Address Line One. Text, then the resulting "den" of the BCC attribute is Address. Line One. Text.

[RBCCAttr] This rule is subdivided into two cases.

1. If no "owner" tag is specified, the UML attribute name of the BCC attribute SHALL be syntactically derived from the property term section of its "den". That is, the name of the attribute is an upper camel case concatenation of all words in the property term section. In addition, if the word "Identifier" or "Identification" is in the property term section, abbreviate it to ID.
2. If one or more "owner" tags are specified, the UML attribute name of the BCC attribute SHALL be syntactically derived from the property term section of its "den" and all of the "owner" tags. That is, the name of the attribute is an upper camel case concatenation of all "owner" tags and then all of the words in the property term section. In addition, if the word "Identifier" or "Identification" is in the property term section, abbreviate it to ID. The order of the "owner" tag SHALL follow the hierarchical path leading to the BCCP class being referenced. The words in the "owner" tags MUST be truncated if repeated words are next to each other.

Using the preceding two examples, the UML attribute names of the two BCC attributes are ID and LineOne.

[RBCCUnique] No two BCCs SHALL be using the same BCCP class within a single ACC class. Compliance with this rule ensures unique BCC attribute name within a single ACC class and unique BCC attribute's "den" within the repository.

[RBCCPos] The "position" tag of the BCC attribute within the Oracle CCTS UML model MUST NOT be specified. The order of the BCC SHALL be implied from its ordinality in the UML model.

[RBCCCar] A BCC attribute MUST have a UML cardinality specified. The UML cardinality on the attribute corresponds to the "Property Cardinality" attribute in the "ACC Property" class in the CCTS meta-model. For BCC, the "ACC Cardinality" attribute in the "ACC Property" class in the CCTS meta-model is always single (1).

The <<CCPRep>> Stereotype

Stereotype	CCPRep (Abstract)			
Base Class	ModelElement (from UML 1.4.2 / 2.0 meta-model)			
Parent	Management::OracleRegistryObject			
Description	This stereotype is an abstract super class of the <<ASCCPRep>> and <<BCCPRep>> stereotypes providing common rules and guidelines for the two stereotypes.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

Rules and Guidelines

[RCCPRepUnid] A UML association having one of the <<CCPRep>> stereotypes MUST be unidirectional.

[RCCPRepCard] A UML association having one of the <<CCPRep>> stereotypes MUST NOT have any UML cardinality specified on either of the two association ends (it can also be interpreted as 1 on both ends). That is, the cardinalities of an ACCP model element (either an ASCC association or a BCC attribute) are to be reflected directly on the ASCC association or the BCC attribute cardinalities.

The <<ASCCPRep>> Stereotype

Stereotype	ASCCPRep			
Base Class	Association (from UML 1.4.2 / 2.0 meta-model)			
Parent	CCPRep			
Description	This stereotype corresponds to the "ASCC Property Representation" class in the CCTS meta-model. An ASCC Property Representation is an association between an ASCCP class and an ACC class. It allows the nature and value domain of an ASCCP to be indicated (i.e., how the property is represented). A UML association having the <<ASCCPRep>> stereotype is regarded as an ASCCPRep association.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

Rules and Guidelines

[RASCCPRepUML] The "Association Type" attribute of the "ASCC Property Representation" class in the CCTS meta-model MUST be represented by the UML association type on the ASCCPRep association. The UML association type MUST be on the ASSCP class end.

[RASCCPRepAssoc] An ASCCPRep association MUST be only a unidirectional association directed from an ASCCP class to an ACC class. The association type on the ASCCP class MUST be either composition or aggregation, and the association type on the ACC class MUST be regular.

The <<BCCPRep>> Stereotype

Stereotype	BCCPRep
Base Class	Association (from UML 1.4.2 / 2.0 meta-model)
Parent	CCPRep
Description	This stereotype corresponds to the "BCC Property Representation" role in the CCTS meta-model. A BCC Property Representation is an association between a BCCP class and a CDT class (from the DataTypes Package.) **For more information** about the DataTypes Package, see DataTypes Package.

	It allows for the nature and value domain of a BCC Property to be indicated. A UML association having the <<BCCPRep>> stereotype is regarded as a BCCPRep association.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

Rules and Guidelines

[RBCCPRepAssoc] A BCCPRep association MUST be only a unidirectional association directed from a BCCP class to a CDT class. The association type on the BCCP class end MUST be composition, while the association type on the CDT class end MUST be regular.

The <<CCC>> Stereotype

Stereotype	CCC			
Base Class	Class (from UML 1.4.2 / 2.0 meta-model)			
Parent	CC			
Description	This stereotype corresponds to the "Choice CC" class in the CCTS meta-model. It allows for alternative subsemantics or representations of a single concept. The Choice CC is used with the Choice ASCC, Sequence CC, and Sequence ASCC. A UML class with a <<CCC>> stereotype is regarded as a CCC class.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	choiceName	string	1:1	This tag semantically conveys what the choice is about. It provides a unique name among CCC classes within the repository.

Rules and Guidelines

[RCCCUnique] The "choiceName" MUST retain its uniqueness among CCC classes within a CCTS UML data model. The consequence of compliance with this rule is the uniqueness of the CCC classes' "den" and their UML class names within the data model.

[RCCCDEN] The "den" of a CCC class SHALL be a concatenation of a "choiceName", a dot character, a space character, and the word "Choice."

For example, a Choice class representing alternative ways of communication may have a "den" - Communication. Choice.

[GChoiceName] The "choiceName" MUST comply with the rules related to the Object Class in the Core Component Rules for Dictionary Entry Name section of the CCTS 2.01.

[RCCCClassName] The UML class name of a CCC class SHALL be syntactically derived from its "den" by concatenating all the words and removing the dot and space characters.

For example, the UML class name of the Communication. Choice is CommunicationChoice.

[RCCC] A CCC class SHALL contain at least two ACC Properties that SHALL be either a BCC attribute, an ASCC association, a CASCC association (choice under choice), or an SASCC association.

The <<CASCC>> Stereotype

Stereotype	CASCC			
Base Class	Association (from UML 1.4.2 / 2.0 meta-model)			
Parent	ACCP			
Description	This stereotype corresponds to the "Choice ASCC" in the CCTS meta-model. It allows for the use of a CCC class to indicate alternatives. A UML association with a <<CASCC>> stereotype is regarded as a CASCC association.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

Rules and Guidelines

[RCASCC] The CASCC association MUST be used only between an ACC, SCC, or CCC class (the using class) and another CCC class, and it MUST be unidirectional and directed from the using class to the CCC class being used. The association type on the using class association end MUST be one of aggregation, regular, or composition. The association type on the other association end MUST be regular.

[RCCCAssoc] The association type on the CCC class end of the association that is directed from that CCC class MUST reflect its parent association type. That is, it MUST be the same as that of the association end on the parent class association that is directed toward the CCC class.

[RCASCCDEN] The "den" of a CASCC association is a concatenation of the "objectClassTerm", "sequenceName", or "choiceName" of the ACC, SCC, or CCC class, a dot, a space, and the "den" of the CCC class being used. The "choiceName" within the "den" of the CCC class being used MUST be truncated as follows. The word or words at the beginning of the "choiceName" of the CCC class being used that is repeated at the end of the "objectClassTerm", "choiceName", or "sequenceName" of the using class are truncated. However, the truncation is void if this results in the whole "choiceName" being truncated.

For example, if an ACC class whose "den" is Communication. Details uses the CCC class whose "den" is Communication. Choice, the "den" of the CASCC association between the two classes is Communication. Communication. Choice. That is, the "choiceName" is not truncated because the truncation would result in the whole "choiceName" being truncated.

[RCASCCCar] A CASCC association MUST have UML cardinalities specified on both ends. The UML cardinality on the using class association end corresponds to the "ACC Cardinality" attribute of the "ACC Property" class in the CCTS meta-model; while the UML cardinality on the CCC class being used corresponds to the "Property Cardinality" attribute.

[RCASCCUnique] There SHALL NOT be two or more association links between a pair of ACC, CCC, or SCC class and CCC class. The compliance with this rule ensures unique CASCC class's "den" within the repository.

The <<SCC>> Stereotype

Stereotype	SCC			
Base Class	Class (from UML 1.4.2 / 2.0 meta-model)			
Parent	CC			
Description	This stereotype corresponds to the "Sequence CC" class in the CCTS meta-model. Sequence CC is used with the Sequence ASCC and the Choice CC to allow for grouping of several components for a single choice. A UML class with an <<SCC>> stereotype is regarded as an SCC class.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	sequenceName	string	1:1	This tag semantically conveys what the Sequence is about. It provides a unique name among SCC classes within the repository.

Rules and Guidelines

[RSCCUnique] The "sequenceName" MUST retain its uniqueness among SCC classes within a CCTS UML data model. The consequence of compliance with this rule is the uniqueness of the SCC classes' "den" and their UML class names within the data model.

[RSequenceName] The Sequence Name MUST comply with the rules related to the Object Class in the Core Component Rules for Dictionary Entry Name section of the CCTS 2.01.

[RSCCDEN] The "den" of an SCC class SHALL be a concatenation of its "sequenceName", a dot character, a space character, and the word "Sequence".

For example, an SCC class grouping information about ways of communication using the Internet may have a "den" – Internet Communication. Sequence.

[RSCC] An SCC class SHALL contain at least two ACC Properties which SHALL be either a BCC attribute, an ASCC association, or a CASCC association (choice under choice). It MUST NOT directly contain another SCC class.

[RSCClassName] The UML class name of an SCC class SHALL be syntactically derived from its "den" by concatenating all the words and removing the dot and space characters.

For example, the UML class name of a Sequence class with the "den" - Internet Communication. Sequence - is InternetCommunicationSequence.

The <<SASCC>> Stereotype

Stereotype	SASCC
Base Class	Association (from UML 1.4.2 / 2.0 meta-model)
Parent	ACCP

Description	This stereotype corresponds to the "Sequence ASCC" class in the CCTS meta-model. It allows for the use of SCC class to indicate an alternative within a choice as group of components. A UML class with an <<SASCC>> stereotype is regarded as an SASCC class.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

Rules and Guidelines

[RSASCC] The SASCC association MUST be used only between a CCC class and an SCC class, and it MUST be unidirectional and directed from the CCC class to the SCC class. The association type on the CCC class association end MUST be either regular, composition, or aggregation depending on the parent association type (see also the rule [RCCAssoc]), while the other end MUST be regular.

[RSCCAssoc] The association type on the SCC class of any association link that is directed from that class MUST reflect its parent association type. That is, it MUST be the same as that of the association end on the parent class association that is directed to the CCC class.

[RSASCCDEN] The "den" of the SASCC association is a concatenation of the "choiceName" of the CCC class using the SCC class, a dot, a space, and the "den" of the SCC class. The "sequenceName" of the SCC class's "den" MUST be truncated as follows. The word or words at the beginning of the "sequenceName" that is repeated at the end of the "choiceName" are truncated. However, the truncation is voided if this results in the whole "sequenceName" being truncated.

For example, if a CCC class whose "choiceName" is Communication uses an SCC class whose "den" is Internet Communication. Sequence, then the "den" of the SASCC association between the two classes is Communication. Internet Communication. Sequence.

[RSASCCCar] An SASCC association MUST have UML cardinalities specified on both ends. The UML cardinality on the using CCC class association end corresponds to the "ACC Cardinality" attribute of the "ACC Property" class in the CCTS meta-model; while the UML cardinality on the SCC class being used corresponds to the "Property Cardinality" attribute.

[RSASCCUnique] There SHALL NOT be two or more association links between a pair of ACC, CCC, or SCC class and CCC class. The compliance with this rule ensures unique SASCC class's "den" within the repository.

Extensive examples of Sequence and Choice constructs are provided in Choice Pattern.

The <<BusinessContext>> Stereotype

Stereotype	BusinessContext (Abstract)
Base Class	Dependency or Object (from UML 1.4.2 / 2.0 meta-model)
Parent	Management::OracleRegistryObject
Description	The <<BusinessContext>> stereotype, when used with the UML object, corresponds to the "Business Context" class in the CCTS meta-model. The <<BusinessContext>> stereotype, when used with the UML dependency, corresponds to the relationship between the "Aggregate Business Information Entity (ABIE)" class and the "Business Context" class in the CCTS meta-model. A Business Context object allows for creation of BIEs based on CCs. It is used to indicate a functional context of a

	CC. A UML object having the <<BusinessContext>> stereotype is regarded as a Business Context object. A UML dependency having the <<BusinessContext>> stereotype is regarded as a Business Context dependency.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

Rules and Guidelines

[RBizCtxObject] A Business Context object SHALL be an instance of the `BusinessContext` class as defined in The BusinessContext class.

The BusinessContext class

Class	BusinessContext			
Parent Class	NA			
Description	The BusinessContext class provides a template for creating a Business Context object.			
Attribute Definitions	Attribute Name	Type	Multiplicity	Description
	businessProcessContext	string	0:n	Identify applicable business processes or business integration scenarios.
	productContext	string	0:n	Identify applicable products.
	industryContext	string	0:n	Identify applicable industries.
	geopoliticalContext	string	0:n	Identify applicable countries or regions.
	systemContext	string	0:n	Identify applicable applications or systems.

[GNoContextCategory] If a context category is not specified in a Business Context object, it SHALL be interpreted that the Business Context object covers any context values within that context category. That is, a depending BIE is applicable in all contexts within that category. This may be because the particular BIE is independent of that particular context category.

The <<basedOn>> Stereotype

Stereotype	basedOn
Base Class	Dependency (from UML 1.4.2 / 2.0 meta-model)
Parent	Management::OracleRegistryObject

Description	This stereotype corresponds to the relation between the "Business Information Entity (BIE)" class and the "Core Component (CC)" class in the CCTS meta-model. It indicates the "Basis" role of a CC for a BIE. A UML dependency relationship having the <<basedOn>> stereotype is regarded as a Based-on dependency.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

The <<ABIE>> Stereotype

Stereotype	ABIE			
Base Class	Class (from UML 1.4.2 / 2.0 meta-model)			
Parent	BIE			
Description	This stereotype corresponds to the "Aggregate Business Information Entity (ABIE)" class in the CCTS meta-model. ABIE is based on an ACC. It is a *functional* restriction of an ACC in a specific business context. A UML class having the <<ABIE>> stereotype is regarded as an ABIE class.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	objectClassQualifier	string	0:n	This tag corresponds to the "Object Class Qualifier" attribute of the "Aggregate Business Information Entity (ABIE)" class in the CCTS meta-model. An Object Class Qualifier is a word or words that reflect a *functional* restriction of the ABIE on the ACC on which it is based.
	objectClassTerm	string	1:1	This tag corresponds to the "Object Class Term" attribute of the "Aggregate Business Information Entity (ABIE)" class in the CCTS meta-model. This tag is a building block for the "den" tag of the ABIE class and its UML class name. The term conceptually represents the business component being modeled.

Rules and Guidelines

[RABIEObj] The "objectClassTerm" of an ABIE class MUST be identical to that of the ACC class on which it is based.

[RABIEUnique] The combination of an ABIE class's "objectClassTerm" and all "objectClassQualifier" MUST retain its uniqueness among ABIE classes within a repository. The consequence of compliance with this rule is the uniqueness of the ABIE class's "den" and UML class name within the repository.

[RABIEACC] An ABIE class MUST have one and only one Based-on dependency relationship with an ACC class.

[RABIE] An ABIE classes MUST NOT contain any UML attribute other than those of the <<BBIE>> stereotype.

[RABIEContext] An ABIE MUST have one and only one Business Context dependency relationship with a Business Context object.

[GABIEQ] An "objectClassQualifier" SHALL be semantically derived from the Business Context object it uses.

For example, when the context values are systemContext = ["Siebel"], geopoliticalContext = ["US", "CA"], businessContext = ["O2C2"], the "objectClassQualifier" may be ["Siebel"] or ["Siebel", "North America"] for a particular based ACC class. The former one, which has only one qualifier term, can be valid if only one ABIE class is based on that ACC class in all ["Siebel"] context combinations. The latter one, which has two qualifier terms, can be valid if only one ABIE class is based on that ACC class in all ["Siebel"] and ["US", "CA"] context combinations. This means that if a particular ABIE is applicable to many countries, but not all, in Western Europe and no need exists for an ABIE for other western European countries anyway, one can choose to use the term "Western Europe" as a qualifier term in the DEN instead of using an exhaustive list of country codes specified in the Business Context object.

[RABIEDEN] The "den" tag of an ABIE class SHALL be syntactically derived from its "objectClassTerm" and "objectClassQualifier" tags. That is, it SHALL be a concatenation of each "objectClassQualifier" suffixed by an underscore and a space character, followed by the "objectClassTerm", and a dot and a space character, and the word "Details." The order of the "objectClassQualifier" tags SHALL be the same order as that is specified in the UML model.

For example, if the "objectClassTerm" of an ACC on which an ABIE class is based is Sales Order and its "objectClassQualifier" tags include ["Siebel", "North America"], then the "den" of the ABIE class is Siebel_ North America_ Sales Order. Details.

[RABIEQOrder] The order of the "objectClassQualifier" SHALL NOT be used to differentiate one ABIE class from another.

[RABIEClassName] The UML class name of an ABIE class MUST be derived from the UML class name of the ACC class on which it is based and its "objectClassQualifier". That is, the name MUST be an upper camel case of concatenated "objectClassQualifier" tags followed by the class name of the ACC on which it is based. The order of the "objectClassQualifier" SHALL be the same order as that is specified in the UML model.

Using the preceding example, the UML class name of the ABIE class is SiebelNorthAmericaSalesOrder.

The <<ABIEP>> Stereotype

Stereotype	ABIEP (abstract)			
Base Class	ModelElement (from UML 1.4.2 / 2.0 meta-model)			
Parent	BIE			
Description	This stereotype corresponds to the "ABIE Property" class in the CCTS meta-model. It is a superclass to provide tags common to <<ASBIE>> and <<BBIE>> stereotypes. ABIE Property represents a property of an ABIE.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	owner	String	0:n	This tag corresponds to the "Owner" attribute of the "ABIE Property" class in the CCTS meta-model. It is used to indicate an ABIE owner of an ABIE property (ASBIE or BBIE) when that property is a reference or repeat of information from another source. See more detail about the usage of the owner tag in Oracle CCTS UML Principles and Guidelines.
	position	Integer	0:1	This tag corresponds to the "Sequencing Key" attribute of the "ABIE Property" class in the CCTS meta-model. Its value indicates the occurrence order of properties in a syntax-specific data structure specification so that the resulting specification is deterministic in every generation.

Rules and Guidelines

[RABIEP] An ABIE class SHALL contain at least one ABIE Property which SHALL be either a UML attribute with a <<BBIE>> stereotype (i.e., the BBIE attribute as defined in The <<BBIE>> Stereotype), a UML association with an <<ASBIE>> stereotype (the ASBIE association as defined in The <<ASBIE>> Stereotype), or a UML association with a <<CASBIE>> stereotype (the CASBIE association as defined in The <<CASBIE>> Stereotype).

[GABIEP] Within an ABIE class, all embedded BBIE attributes, ASBIE associations, and CASBIE associations SHALL be related to the concept of the aggregate.

[RUniqueABIEP] ABIE Properties MUST be unique within an ABIE. That is, there SHALL NOT be any ASBIE association, CASBIE association, and BBIE attributes for which "`den`" tags are identical within an ABIE class. Alternatively stating, there MUST NOT be more than one association link between any two UML classes.

[RCirABIEP] An ABIE Property that is an ASBIE and CASBIE associations MUST be devoid of mandatory circular references. That is, an ABIE class SHALL never contain—directly or at any nested level—a mandatory ASBIE and CASBIE associations for which the associated ABIE class is the same as itself or a higher level ABIE class.

[RABIEPPosSem] The ordering of properties through the "`position`" tag SHALL convey no business semantics. That is, different orderings of properties do not make one ABIE class semantically different from another.

Note that unlike the similar rule in the <<`ACCP`>> stereotype, two ABIEs might have the same set of properties.

[RABIEPPosUnique] The "`position`" tag MUST be unique within the set of BBIE attributes and within the set of ASBIE and CASBIE associations under the context of an ABIE class (set of ASBIE and SASBIE associations under the context of a CBIE class).

For example, if an ABIE has 3 BBIEs and 2 ASBIEs, then the position values among the 3 BBIEs must be different and those among the 2 ASBIEs must be different; however, the two sets may overlap. The positions of the 3 BBBIEs may be [2, 4, 5] and the 2 ASBIEs may be [1, 2]. That is, they only need to be unique and do not necessarily have to be incremental by 1. In fact, the BBIE position is not necessary; therefore, this example is for explanation only.

[RABIEPPosImpl] An implementation of a syntax-specific data structure specification generation from the Oracle CCTS UML model SHALL use the "`position`" tag to order the ABIEP associations under the same parent class in ascending order.

[RABIEPPos] The "`position`" tag MUST be specified for every ASBIE, CASBIE, and SASBIE association.

[RABIEPOwner] The "`owner`" in an ASBIE association MUST be the same as that of the ASCC association on which it is based. The "`owner`" tag in a BBIE attribute MUST be the same as that of the BCC attribute on which it is based. The "`owner`" in a CASBIE association MUST be the same as that of the CASCC association on which it is based. The "`owner`" in a SASBIE association MUST be the same as that of the SASCC association on which it is based.

The <<ASBIEP>> Stereotype

Stereotype	ASBIEP
Base Class	Class (from UML 1.4.2 / 2.0 meta-model)
Parent	BIE
Description	This stereotype corresponds to the "ASBIE Property" class in the CCTS meta-model. ASBIE Property is a reusable complex business characteristic in a specific business context. An ASBIE Property is associated with the appropriate ABIE, which indicates how the property or the business characteristic is described in a particular business context. Together, they create a context-specific component reusable in another ABIE. A UML class with this <<`ASBIEP`>> stereotype is regarded as an ASBIEP class.

	Tag Name	Type	Multiplicity	Description
Tag Definitions	propertyQualifier	String	0:n	This tag corresponds to the "Property Qualifier" attribute of the "ASBIE Property" class in the CCTS meta-model. A Property Qualifier is a word or words that reflect a *functional* restriction of the ASBIEP on the ASCCP on which it is based.
	propertyTerm	String	1:1	This tag corresponds to the "Property Term" attribute of the "ASBIE Property" class in the CCTS meta-model. It is a semantically meaningful name for the characteristic that the ASBIE Property represents. Typically, it represents the nature of the association with the associated ACC.

Rules and Guidelines

[RASBIEPBase] An ASBIEP class SHALL have one and only one Based-on dependency relationship with an ASCCP class.

[RASBIEPContext] An ASBIEP MUST have one and only one Business Context dependency relationship with a Business Context object.

[GASBIEPQ] A "`PropertyQualifier`" SHALL be semantically derived from the Business Context object it uses.

[RASBIEPProp] The "`propertyTerm`" of an ASBIEP MUST be the same as the ASCCP class on which it is based.

[RASBIEPUnique] The combination of the "`propertyTerm`" and "`propertyQualifier`" MUST retain its uniqueness among ASBIEP and BBIEP classes within the repository. The consequence of compliance with this rule is the uniqueness of the ASBIEP and BBIEP class's "`den`" and UML class name within the repository.

[RASBIEPAssoc] An ASBIEP class SHALL have an association link with one and only one ABIE class. In addition, the association type MUST be unidirectional with an <<`ASBIEPRep`>> stereotype directed from the ASBIEP class to the ABIE class. The association types on the ASBIEP and the ABIE classes MUST be identical to the respective ASCCP and ACC classes on which they are based.

[RASBIEPDEN] The "`den`" of the ASBIEP class SHALL be syntactically derived from its "`propertyQualifier`" tags, the "`den`" of its base ASCCP class, and the "`objectClassQualifier`" tags of the associated ABIE class. That is, its "`den`" MUST be created by inserting each "`propertyQualifier`" in front of the property term section and each "`objectClassQualifier`" in front of the representation term section of the ASCCP class's "`den`". In addition, each "`propertyQualifier`" and "`objectClassQualifier`" MUST be suffixed by an underscore and a space character. All qualifiers are ordered according to their order in the UML model.

Figure 12 shows an example of an ABIEP class's "`den`".

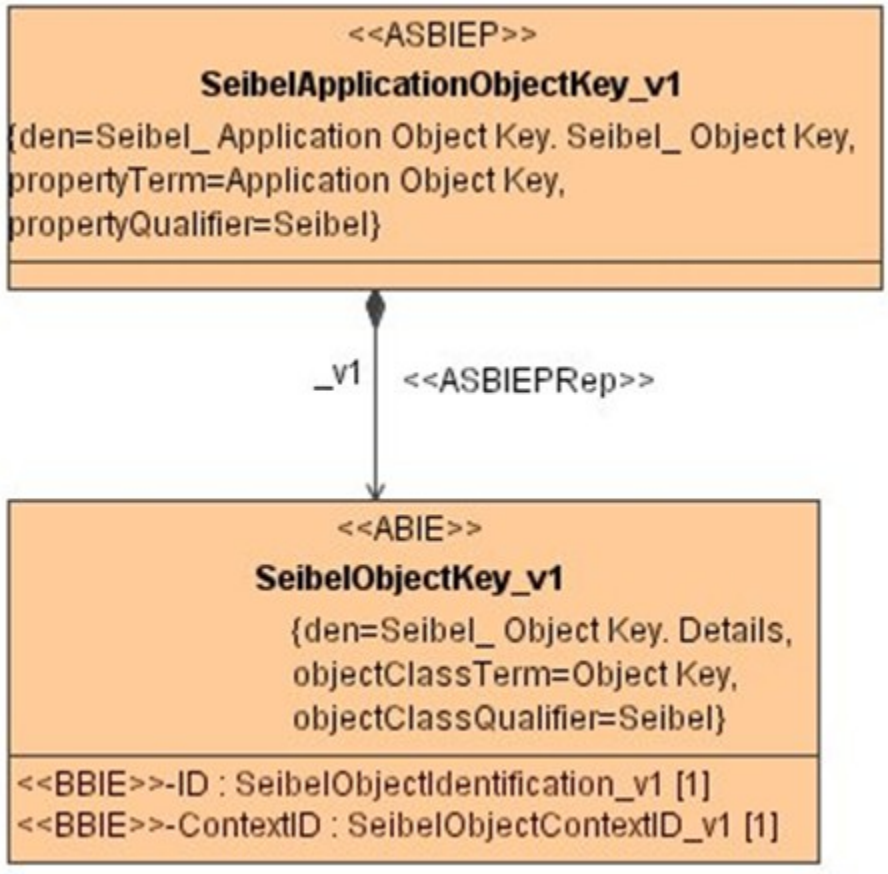

Figure 12: ASBIEP naming

[RASBIEPClassName] The UML class name of the ABIEP class SHALL be syntactically derived from its "`propertyQualifier`" and its "`propertyTerm`". That is, the name MUST be an upper camel case concatenation of the "`propertyQualifier`" followed by the "`propertyTerm`". All qualifiers are ordered according to their order in the UML model.

The <<ASBIE>> Stereotype

Stereotype	ASBIE
Base Class	Association (from UML 1.4.2 / 2.0 meta-model)
Parent	ABIEP

Description	This stereotype corresponds to the "Association Business Information Entity (ASBIE)" class in the CCTS meta-model. ASBIE is based on an ASCC. ASBIE indicates a complex property/characteristic of an associating ABIE by establishing an association link with an ASBIEP. The functional restriction of the parent ABIE drives the functional restriction of the ASBIE and ASBIEP that it uses downward. A UML association having a <<ASBIE>> stereotype is regarded as an ASBIE association.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

Rules and Guidelines

[RASBIEBase] Every ASBIE association MUST have a valid base ASCC association. This can be automatically validated and recognized by the classes it links and find out if bases of those classes have an ASCC association link. Therefore, drawing a Based-on dependency relationship between an ASBIE and the based ASCC is not necessary. The following add-on to this rule ensures a consistent look and feel and also prevents mistakes: An ASBIE association SHALL NOT physically have any Based-on dependency relationship.

[RASBIEContext] An ASBIE always takes business contexts from other classes it associates with. This rule ensures a consistent look and feel and also prevents mistakes. An ASBIE association MUST NOT physically have any Business Context dependency relationship with a Business Context object.

[RASBIEDEN] The "`den`" of an ASBIE MUST be syntactically derived from the "`den`" of the ASCC on which it is based, each "`objectClassQualifier`", "`choiceQualifier`", or "`sequenceQualifier`" of the parent ABIE, CBIE, or SBIE class, each "`propertyQualifier`" of the ASBIEP class it uses, and each "`objectClassQualifier`" of the ABIE class used by the ASBIEP class. That is, its "`den`" MUST be created by inserting the first "`objectClassQualifier`" tags, "`choiceQualifier`" tags, or "`sequenceQualifier`" tags in front of the object class term section, each "`propertyQualifier`" in front of the property term section, and the other "`objectClassQualifier`" tags in front of the representation term section of the ASCC's "`den`". In addition each of the "`propertyQualifier`" and "`objectClassQualifier`", "`choiceQualifier`", or "`sequenceQualifier`" MUST be suffixed by an underscore and a space characters. This rule needs the rule [RASBIEBase] validated first to identify the based ASCC. Note that this rule is a relaxed implementation of the rule [B29] in the CCTS 2.01. That rule truncates the property term when it is not qualified.

For example, as shown in Figure 13, the "`den`" of the ASBIE association is `Siebel_ Identification. Siebel_ Application Object Key. Siebel_ Object Key.`

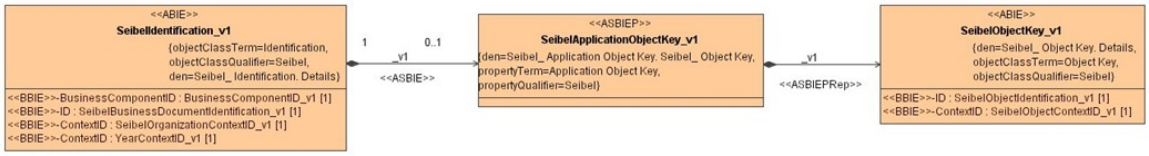

Figure13: ASBIE dictionary entry naming

[RASBIEAssoc] An ASBIE association MUST be a unidirectional association directed from an ABIE, CBIE, or SBIE class to an ASBIEP class. The association type MUST also be identical to the based ASCC association. This rule needs the rule [RASBIEBase] validated first to identify the based ASCC.

[RASBIECar] An ASBIE association MUST have UML cardinalities specified on both ends. The UML cardinality on the ABIE, CBIE, or SBIE class association end corresponds to the "ABIE Cardinality" attribute of the "ABIE Property" class in the CCTS meta-model, while the UML cardinality on the ASBIEP class association end corresponds to the "Property Cardinality" attribute. These UML cardinalities MUST either be identical or a restriction must be placed on those on the based ASCC association. This rule needs the rule [RASBIEBase] validated first to identify the based ASCC.

[RASBIEUnique] A particular pair of an ABIE, CBIE, or SBIE class and an ASBIEP class MUST have at most one association link. The compliance with this rule ensures unique ASBIE class's "`den`" within the repository.

[RASBIEOwner] If the ASCC association on which the ASBIE is based has an "`owner`" specified, the information MUST also be carried onto the ASBIE's "`owner`." This rule needs the rule [RASBIEBase] validated first to identify the based ASCC.

The <<BBIEP>> Stereotype

<table>
<tr><td>Stereotype</td><td colspan="4">BBIEP</td></tr>
<tr><td>Base Class</td><td colspan="4">Class (from UML 1.4.2 / 2.0 meta-model)</td></tr>
<tr><td>Parent</td><td colspan="4">BIE</td></tr>
<tr><td>Description</td><td colspan="4">This stereotype corresponds to the "BBIE Property" class in the CCTS meta-model. BBIEP is a reusable, singular business characteristic in a specific business context. BBIE Property is associated with a class with the <<BDT>> stereotype (business data type stereotype is defined in the DataTypes Package), which indicates how the property or the business characteristic is described in a particular business context.

For more information about the DataTypes Package, see DataTypes Package.

Together, they create a context-specific business component that is reusable across ABIEs. A UML class with this <<BBIEP>> stereotype is regarded as a BBIEP class.</td></tr>
<tr><td rowspan="2">Tag Definitions</td><td>Tag Name</td><td>Type</td><td>Multiplicity</td><td>Description</td></tr>
<tr><td>propertyQualifier</td><td>string</td><td>0:n</td><td>This tag corresponds to the "Property Qualifier" attribute of the "BBIE Property" class in the CCTS meta-model. A Property Qualifier is a word or words that reflect a functional restriction of the BBIEP on the BCCP on which it is based.</td></tr>
</table>

	propertyTerm	string	1:1	This tag corresponds to the "Property Term" attribute of the "BBIE Property" class in the CCTS meta-model. It is a semantically meaningful name for a unique characteristic of an ABIE object class
	repTerm	string	1:1	This tag corresponds to the "Representation Term" attribute of the "BBIE Property" class in the CCTS meta-model. It indicates the value space of the BBIE Property, which is how the BBIE Property may be described.

Rules and Guidelines

[RBBIEPBase] A BBIEP class SHALL have one and only one Based-on dependency relationship with a BCCP class.

[RBBIEPContext] A BBIEP MUST have a Business Context dependency relationship with a Business Context object.

[GBBIEPQ] A "`PropertyQualifier`" SHALL comply with the guideline in [GASBIEPQ].

[RBBIEPProp] The "`propertyTerm`" of a BBIEP class MUST be identical to that of the BCCP class on which it is based.

[RBBIEPUnique] The "`propertyTerm`" MUST comply with the rule [RASBIEPUnique].

[RBBIEPAssoc] A BBIEP class SHALL have an association link with one and only one BDT class (a class with <<`BDT`>> stereotype defined in the `DataTypes Package`).

For more information about the `DataTypes Package`, see DataTypes Package.

The associated BDT class MUST be based on the CDT class associated with the BCCP class on which the BBIEP class is based (A CDT class is a class with <<`CDT`>> stereotype defined in the `DataTypes Package`). In addition, the association MUST be a unidirectional association with a <<`BBIEPRep`>> stereotype pointing from the BBIEP class to the BDT class. Other association information MUST be the same as that on the BCCPRep association counterpart.

[RBBIEPRep] The "`repTerm`" of the BBIEP class SHALL be the representation term section of the BDT class' "`den`" with which it is associated (this is data type qualifier plus the data type term).

[RBBIEPDEN] The "den" tag of the BBIEP class SHALL be syntactically derived from the "den" of its base BCCP by inserting its "propertyQualifier" tags in front of the property term section and its BDT class's "dataTypeQualifier" tags in front of the representation term section of the "den". In addition, each of the "propertyQualifier" and "dataTypeQualifier" MUST be suffixed by an underscore and a space character.

Figure 14 illustrates an example BBIEP derived from the Context Identification. Identifier BCCP. The "propertyQualifier" tags of the BBIEP include ["Siebel", "Object"] and the "dataTypeQualifier" tags of the associated BDT include ["Siebel", "Business Object Type"]; consequently, the "den" of the BBEIP is Siebel_ Object_ Context Identification. Siebel_ Business Object Type_ Identifier.

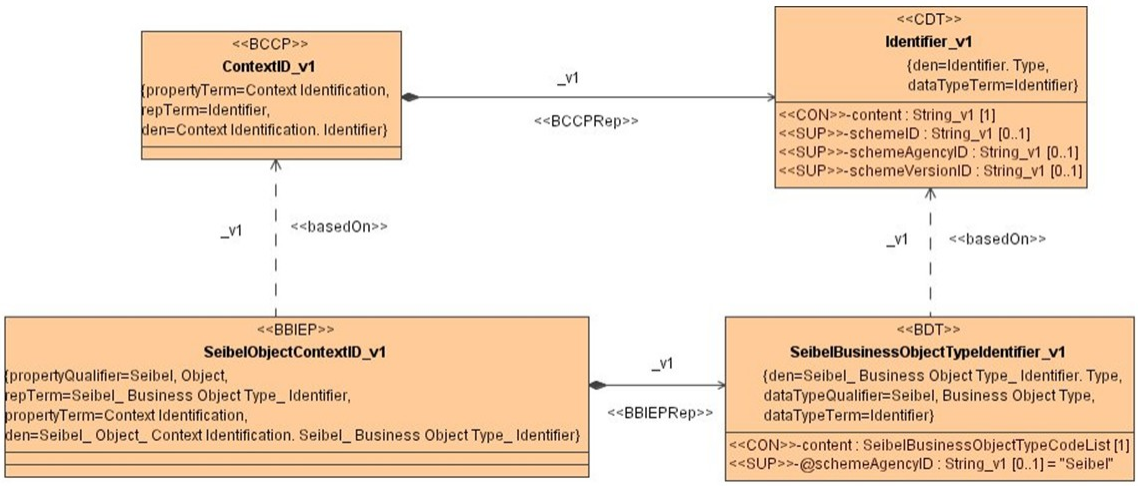

Figure 14: BBIEP dictionary entry naming

[RBBIEPClassName] The UML class name of the BBIEP class SHALL be syntactically derived from its "propertyQualifier" and class name of the based BCCP class. That is, the name MUST be an upper camel case concatenation of the "propertyQualifier" tags followed by the based BCCP class name.

Using the example in Figure 14, the UML class name of the BBIEP is SiebelObjectContextID because its based BCCP class name is ContextID and its "propertyQualifier" includes ["Siebel", "Object"].

The <<BBIE>> Stereotype

Stereotype	BBIE			
Base Class	Attribute (from UML 1.4.2 / 2.0 meta-model)			
Parent	ABIEP			
Description	This tag corresponds to the "Basic Business Information Entity (BBE)" class in the CCTS meta-model. BBIE is based on a BCC. BBIE indicates a singular business characteristic of an ABIE by using a BBIEP class as its type. The functional restriction of the parent ABIE drives the functional restriction of the BBIE and BBIEP that it uses. A UML attribute with this <<BBIE>> stereotype is regarded as a BBIE attribute.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

Rules and Guidelines

[RBBIEType] The UML type of a BBIE attribute SHALL be one of the BBIEP classes.

[RBBIEBase] Every BBIE attribute MUST have valid base BCC attribute. This can be validated automatically and recognized by its parent class (ABIE, CBIE, or SBIE) and the BBIEP class that it uses (this is because a BCC usage of a BCCP class is unique within a class). Therefore, an explicit dependency relationship between a BBIE and a BCC that it is based on is not necessary.

[RBBIEContext] A BBIE always takes business contexts from its parent ABIE and the BBIEP that it uses. This rule ensures a consistent look and feel and also prevents mistakes. A BBIE MUST NOT physically have a Business Context dependency relationship with a Business Context object.

[RBBIEDEN] The "`den`" tag of a BBIE class MUST be syntactically derived from the "`den`" of the BCC class on which it is based, the "`objectClassQualifier`", "`sequenceQualifier`", or "`choiceQualifier`" tags of the parent ABIE, SBIE, or CBIE class, the "`propertyQualifier`" tags of the BBIEP class that it uses, and the "`dataTypeQualifier`" tags of the BDT class that is used by the BBIEP class. That is, its "`den`" MUST be created by inserting the each "`objectClassQualifier`", "`sequenceQualifier`", or "`choiceQualifier`" in front of the object class term section, each "`propertyQualifier`" in front of the property term section, and each "`dataTypeQualifier`" in front of the representation term section of the BCC's "`den`". In addition, each "`propertyQualifier`", "`objectClassQualifier`", and "`dataTypeQualifier`" MUST be suffixed by an underscore and a space character. Note that this rule needs the rule [RBBIEBase] validated first to identify the based BCC class. This rule is a relaxed implementation of the rule [B29] in the CCTS 2.01. That rule truncates the property term when it is not qualified.

For example, if "`den`" of a BCC attribute is `Identification. Identification. Identifier`, and an "`objectClassQualifier`" is ["`Siebel`"], a "`propertyQualifier`" tags include ["`Siebel`", "`Object`"], and a "`dataTypeQualifier`" tags include ["`Siebel`", "`Business Object Type`"], then the "`den`" of the corresponding BBIE is `Siebel_ Identification. Siebel_ Object_ Identification. Siebel_ Business Object_ Type Identifier`.

[RBBIEAttr] The UML attribute name of the BBIE attribute SHALL be identical with that of the BCC on which it is based. Note that this rule needs the rule [RBBIEBase] validated first to identify the based BCC class.

[RBBIEUnique] No two BBIEs SHALL be using the same BBIEP class within a single ABIE, SCIE, or CBIE class. Compliance with this rule ensures that no duplicate BBIE attribute's "`den`" exists.

[RBBIEPos] The position tag of the BBIE attribute within the Oracle CCTS UML model MUST NOT be specified. The order of the BCC SHALL be implied from its ordinality in the model.

[RBBIECar] A BBIE attribute MUST have a UML cardinality specified. The cardinality of the attribute corresponds to the "Property Cardinality" attribute of the "ABIE Property" class in the CCTS meta-model. The "ABIE Cardinality" attribute of the "ABIE Property" class in the CCTS meta-model of a BBIE is always single (1) without needing to be explicitly specified. The cardinality MUST be either identical or a restriction of the cardinality on the based BCC attribute. This rule needs the rule [RBBIEBase] validated first to identify the based BCC attribute.

[RBBIEOwner] If the BCC attribute, on which a BBIE is based, has an "`owner`" specified, then the information MUST be carried over to the BBIE attribute. Note that this rule needs the rule [RBBIEBase] validated first to identify the based BCC class.

The <<BIEPRep>> Stereotype

Stereotype	BIEPRep (Abstract)			
Base Class	ModelElement (from UML 1.4.2 / 2.0 meta-model)			
Parent	Management::OracleRegistryObject			
Description	This stereotype is an abstract super class of the <<ASBIEPRep>> and <<BBIEPRep>> stereotypes providing common rules and guidelines for the two stereotypes.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

Rules and Guidelines

[RBIEPRepUnid] A UML association having one of the <<BIEPRep>> stereotypes MUST be unidirectional.

[RBIEPRepCard] A UML association having one of the <<BIEPRep>> stereotypes MUST NOT have any UML cardinality specified on either of the two association ends (it can also be interpreted as 1 on both ends). That is, the cardinalities of an ABIEP model element (either an ASBIE association or a BBIE attribute) are to be reflected directly on the ASBIE association or the BBIE attribute cardinalities.

The <<ASBIEPRep>> Stereotype

Stereotype	ASBIEPRep			
Base Class	Association (from UML 1.4.2 / 2.0 meta-model)			
Parent	BIEPRep			
Description	This stereotype corresponds to the “ASBIE Property Representation” class in the CCTS meta-model. An ASBIE Property Representation is an association that is directed from an ASBIEP class to an ABIE class. It allows for an ASBIEP class to be a functional restriction of an ASCCP class by establishing an association link to a functionally restricted ABIE class that is based on the ACC class used by the ACCP class. A UML association having the <<ASBIEPRep>> stereotype is regarded as an ASBIEPRep association.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

Rules and Guidelines

[RASBIEPRepUML] The “Association Type” attribute of the “ASBIE Property Representation” class in the CCTS meta-model MUST be represented by the UML association type on the association directed from an ASBIEP class to an ABIE class. The association type is to be specified on the ASBIEP class association end.

[RASBIEPRepBase] Every ASBIEPRep association MUST have a valid base ASCCPRep association. This can be automatically validated and recognized by the ASBIEP and ABIE classes that it links (this is because each ASCCP class has only one association to an ACC class). Therefore, drawing a Based-on dependency relationship between an ASBIEPRep and the ASCCPRep it is based on is not necessary. The following add-on to this rule ensures a consistent look and feel and also prevents mistakes: An ASBIEPRep association SHALL NOT physically have any Based-on dependency relationship.

[RASCCPRepAssoc] The ASBIEP association MUST only be a unidirectional association directed from an ASBIEP class and an ABIE class. The association types on the ASBIEP and ABIE classes MUST be identical with the corresponding base ASCCP and ACC classes. This rule needs the rule [RASBIEPRepBase] validated first to identify the based ASCCPRep association.

The <<BBIERep>> Stereotype

Stereotype	BBBIEPRep			
Base Class	Association (from UML 1.4.2 / 2.0 meta-model)			
Parent	BIEPRep			
Description	This stereotype corresponds to the "BBIE Property Representation" role in the CCTS meta-model. A BBIE Property Representation is an association between a BBIEP class and a BDT class (from the `DataTypes Package`). **For more information** about the `DataTypes Package`, see DataTypes Package. It allows for a BBIEP class to be a functional restriction of a BCCP class by establishing an association link to a functionally restricted BDT class. A UML association having the <<`BBIEPRep`>> stereotype is regarded as a BBIEPRep association.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

Rules and Guidelines

[RBBIEPRepBase] Every BBIEPRep association MUST have a valid base BCCPRep association. This can be validated automatically and recognized by the BBIEP and BDT classes that it links (this is because each BCCP class has only one association to a CDT class). Therefore, drawing a Based-on dependency relationship between a BBIEPRep and the BCCPRep it is based on is not necessary. The following add-on to this rule ensures a consistent look and feel and also prevents mistakes: A BBIEPRep association SHALL NOT physically have any Based-on dependency relationship.

[RBBIEPRepAssoc] A BBIEPRep association MUST only be a unidirectional association directed from a BBIEP class and a BDT class. The association types on the BBIEP and BDT classes MUST be identical with the corresponding base BCCP and CDT classes. This rule needs the rule [RBBIEPRepBase] validated first to identify the based BCCPRep association.

The <<CBIE>> Stereotype

Stereotype	CBIE

Base Class	Class (from UML 1.4.2 / 2.0 meta-model)			
Parent	BIE			
Description	This stereotype corresponds to the "Choice BIE" class in the CCTS meta-model. Choice BIE is based on a Choice CC. It is a functional restriction of a Choice CC in a specific business context. A UML class with a <<CBIE>> stereotype is regarded as a CBIE class.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	choiceName	string	1:1	This tag semantically conveys what the choice is about. It provides a unique name among CCC classes within the repository.
	choiceQualifier	string	0:n	This tag corresponds to the "Choice Qualifier" attribute of the "Choice BIE" class in the CCTS meta-model. A Choice Qualifier is a word or words that reflect a *functional* restriction of the CBIE class on the CCC class on which it is based.

Rules and Guidelines

[RChoiceName] The "choiceName" of a CBIE class MUST be identical to that of the CCC class on which it is based.

[RCBIEUnique] The combination of a CBIE class's "choiceName" and all of its "choiceQualifier" tags MUST retain its uniqueness among CBIE classes within a repository. The consequence of compliance with this rule is the uniqueness of the CBIE class's "den" and UML class name within the repository.

[RCBIECCC] A CBIE class MUST have one and only one Based-on dependency relationship with a CCC class.

[RCBIE] A CBIE classes MUST NOT contain any UML attribute other than those of the <<BBIE>> stereotype.

[RCBIEContext] A CBIE MUST have one and only one Business Context dependency relationship with a Business Context object.

[GCBIEQ] A "choiceQualifier" SHALL be semantically derived from the Business Context object it uses that is similar to that described in [GABIEQ].

[RCBIEDEN] The "den" of a CBIE class SHALL be syntactically derived from its "choiceName" and "choiceQualifier" tags. That is, it SHALL be a concatenation of each "choiceQualifier" suffixed by an underscore and a space characters, followed by the "choiceName", a dot and a space characters, and the word "Choice." The order of each "choiceQualifier" SHALL be in the same order as that is specified in the UML model.

For example, if the "choiceName" of a CCC class on which a CBIE class is based is Communication and its "choiceQualifier" includes ["Siebel"], then the "den" of the CBIE class is Siebel_ Communication. Choice.

[RCBIEQOrder] Order of the "choiceQualifier" SHALL NOT be used to differentiate one CBIE class from another.

[RCBIEClassName] UML class name of a CBIE class MUST be derived from the UML class name of the CCC class on which it is based and its "choiceQualifier" tags. That is, the name MUST be an upper camel case concatenation of all "choiceQualifier" tags followed by the class name of the based CCC. The order of the "choiceQualifier" tags SHALL be in the same order as that is specified in the UML model.

Using the preceding example, the UML class name of the CBIE is SiebelCommunicationChoice.

A CBIE class might not really be a choice any more due to the nature of the restriction, that is, all alternatives may be disallowed but one.

The <<CASBIE>> Stereotype

Stereotype	CASBIE			
Base Class	Association (from UML 1.4.2 / 2.0 meta-model)			
Parent	ACCP			
Description	This stereotype corresponds to the "Choice ASBIE" in the CCTS meta-model. A Choice ASBIE is based on a Choice ASCC. It allows for a functionally restricted CBIE class to be used by another class. A UML association with a <<CASBIE>> stereotype is regarded as a CASBIE association.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

Rules and Guidelines

[RCASBIEBase] Every CASBIE association MUST have a valid base CASCC association. This can be validated automatically and recognized by identifying the classes it links and determining whether bases of those classes have a CASCC association link. Therefore, drawing a Based-on dependency relationship between a CASBIE and the based CASCC is not necessary. The following add-on to this rule ensures a consistent look and feel and also prevents mistakes: A CASBIE association SHALL NOT physically have any Based-on dependency relationship.

[RCASBIEContext] A CASBIE always takes business contexts from other classes with which it associates. This rule ensures a consistent look and feel and also prevents mistakes. A CASBIE association MUST NOT physically have any Business Context dependency relationship with a Business Context object.

[RCASBIEDEN] The "den" of a CASBIE MUST be syntactically derived from the "den" of the CASCC on which it is based, the "objectClassQualifier", "choiceQualifier", or "sequenceQualifier" tags of the parent ABIE, CBIE, or SBIE class, and the "choiceQualifier" tags of the CBIE class it uses. That is, its "den" MUST be created by inserting the first set of "objectClassQualifier", "choiceQualifier", or "sequenceQualifier" tags in front of the object class term section and the "choiceQualifier" tags of the associated CBIE class in front of the property term section of the CASCC's "den". In addition, each "propertyQualifier" and "objectClassQualifier", "choiceQualifier", or "sequenceQualifier" MUST be suffixed by an underscore and a space character. This rule needs the rule [RCASBIEBase] validated first to identify the based CASCC. Note that this rule is a relaxed implementation of the rule [B29] in the CCTS 2.01. That rule truncates the property term when it is not qualified.

For example, if the "den" of a CASBIE is an association between the Siebel_ Communication. Details ABIE class and the Siebel_ Communication. Choice CBIE class, the resulting CASBIE's "den" is Siebel_ Communication. Siebel_ Communication. Choice because the based CASCC's "den" is Communication. Communication. Choice (that is, no truncation occurred).

[RCASBIEAssoc] A CASBIE association MUST be a unidirectional association directed from an ABIE, CBIE, or SBIE class to a CBIE class. The association type MUST also be identical to the based CASCC association. This rule needs the rule [RCASBIEBase] validated first to identify the based CASCC.

[RCASBIECar] A CASBIE association MUST have UML cardinalities specified on both ends. The UML cardinality on the ABIE, CBIE or SBIE class association end corresponds to the "ABIE Cardinality" attribute of the "ABIE Property" class in the CCTS meta-model; while the UML cardinality on the CBIE class association end corresponds to the "Property Cardinality" attribute. These UML cardinalities MUST be either identical or a restriction on those on the based CASCC association. This rule needs the rule [RCASBIEBase] validated first to identify the based CASCC.

[RCASBIEUnique] A pair of an ABIE, CBIE, or SBIE class and a CBIE class MUST have at most one association link. The compliance with this rule ensures unique CASBIE class's "den" within the repository.

[RCASBIEOwner] If the CASCC association on which the CASBIE is based has an "owner" specified, the information MUST be also carried onto the CASBIE's "owner". This rule needs the rule [RCASBIEBase] validated first to identify the based CASCC.

The <<SBIE>> Stereotype

Stereotype	SBIE			
Base Class	Class (from UML 1.4.2 / 2.0 meta-model)			
Parent	BIE			
Description	This stereotype corresponds to the "Sequence BIE" class in the CCTS meta-model. Sequence BIE is based on a Sequence CC. It is a functional restriction of a Sequence CC in a specific business context. A UML class with a <<SBIE>> stereotype is regarded as an SBIE class.			
Tag	Tag Name	Type	Multiplicity	Description

Definitions	sequenceName	string	1:1	This tag semantically conveys what the Sequence is about. It provides a unique name among SCC classes within the repository.
	sequenceQualifier	string	0:n	This tag corresponds to the "Sequence Qualifier" attribute of the "Sequence BIE" class in the CCTS meta-model. A Sequence Qualifier is a word or words that reflect a *functional* restriction of the SBIE class on the SCC class on which it is based.

Rules and Guidelines

[RSBIESCC] An SBIE class MUST have one and only one Based-on dependency relationship with a SCC class.

[RSBIE] An SBIE class MUST NOT contain any UML attribute other than those of the <<BBIE>> stereotype.

[RSBIEContext] An SBIE MUST have one and only one Business Context dependency relationship with a Business Context object.

[RSBIEUnique] The combination of an SBIE class's "sequenceName" and all "sequenceQualifier" tags MUST retain its uniqueness among SBIE classes within a repository. The consequence of compliance with this rule is the uniqueness of the SBIE class's "den" and UML class name within the repository.

[RSequenceName] The "sequenceName" of an SBIE class MUST be identical to that of the SCC class on which it is based.

[GSBIEQ] A "sequenceQualifier" SHALL be semantically derived from the Business Context object it uses in the similar way to that described in [GABIEQ].

[RSBIEQOrder] The order of the "sequenceQualifier" tags SHALL NOT be used to differentiate one SBIE class from another.

[RSBIEDEN] The "den" of a SBIE class SHALL be syntactically derived from its "sequenceName" and "sequenceQualifier" tags. That is, it SHALL be a concatenation of each "sequenceQualifier" suffixed by an underscore and a space character, followed by the "sequenceName", and a dot and a space characters, and the word "Sequence." The order of the "sequenceQualifier" tags SHALL be in the same order specified in the UML model.

For example, if the "sequenceName" of an SCC class on which an SBIE class is based is Internet Communication and the "sequenceQualifier" tags of the SBIE are ["Siebel", "O2C2"], then the "den" of the SBIE class is Siebel_ O2C2_ Internet Communication. Sequence.

[RSBIEClassName] The UML class name of an SBIE class SHALL be syntactically derived from its "den" by concatenating all the words and removing the dot, space, and underscore characters.

Using the preceding example, the UML class name of the SBIE is Siebel_O2C2_Internet_Communication_Sequence.

An SBIE class might not really be a sequence any more due to the nature of the restriction. That is, all structures within the sequence may be disallowed but one.

The <<SASBIE>> Stereotype

Stereotype	SASBIE			
Base Class	Association (from UML 1.4.2 / 2.0 meta-model)			
Parent	ABIEP			
Description	This stereotype corresponds to the "Sequence ASBIE" class in the CCTS meta-model. A Sequence ASBIE is based on a Sequence ASCC. It allows for a functionally restricted SBIE class to be used in a CBIE class. A UML class with an <<SASBIE>> stereotype is regarded as an SASBIE class.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

Rules and Guidelines

[RSASBIEBase] Every SASBIE association MUST have a valid base SASCC association. This can be validated and recognized automatically by identifying the classes it links and determining whether bases of those classes have an SASCC association link. Therefore, drawing a Based-on dependency relationship between a SASBIE and the based SASCC is not necessary. The following add-on to this rule ensures a consistent look and feel and also prevents mistakes: A SASBIE association SHALL NOT physically have any Based-on dependency relationship.

[RCASBIEContext] An SASBIE always takes business contexts from other classes with which it associates. This rule ensures a consistent look and feel and also prevents mistakes. A CASBIE association MUST NOT physically have any Business Context dependency relationship with a Business Context object.

[RCASBIEDEN] The "den" tag of an SASBIE MUST be syntactically derived from the "den" of the SASCC on which it is based, the "choiceQualifier" tags of the parent CBIE class, and the "sequenceQualifier" tags of the SBIE class it uses. That is, its "den" MUST be created by inserting the first set of "choiceQualifier" tags in front of the object class term section and the "sequenceQualifier" tags of the associated SBIE class in front of the property term section of the SASCC's "den". In addition, each "choiceQualifier" and "sequenceQualifier" MUST be suffixed by an underscore and a space character. This rule needs the rule [RSASBIEBase] validated first to identify the based SASCC. Note that this rule is a relaxed implementation of the rule [B29] in the CCTS 2.01. That rule truncates the property term when it is not qualified.

For example, if a CBIE class whose "den" is Siebel_ Communication. Choice uses an SBIE class whose "den" is Siebel_ O2C2_ Internet Communication. Sequence, the "den" of the SASBIE between the two classes is Siebel_ Communication. Siebel_ O2C2_ Internet Communication. Sequence.

[RCASBIEAssoc] A SASBIE association MUST be a unidirectional association directed from a CBIE class to a SBIE class. The association type MUST also be identical to the based SASCC association. This rule needs the rule [RSASBIEBase] validated first to identify the based CASCC.

[RASBIECar] A SASBIE association MUST have UML cardinalities specified on both ends. The UML cardinality on the CBIE class association end corresponds to the “ABIE Cardinality” attribute of the “ABIE Property” class in the CCTS meta-model; while the UML cardinality on the SBIE class association end corresponds to the “Property Cardinality” attribute. These UML cardinalities MUST be either identical or a restriction on those on the based SASCC association. This rule needs the rule [RSASBIEBase] validated first to identify the based SASCC.

[RSASBIEUnique] A pair of a CBIE class and an SBIE class MUST have at most one association link. The compliance with this rule ensures unique SASBIE class’s “`den`” within the repository.

[RSASBIEOwner] If the SASCC association on which the SASBIE is based has an “owner” specified, the information MUST also be carried onto the SASBIE’s “owner”. This rule needs the rule [RSASBIEBase] validated first to identify the based SASCC.

DataTypes Package

The diagram in Figure 15 shows an overview of the stereotype definitions in the DataTypes package. All stereotypes are ultimately based on the <<`OracleRegistryObject`>> in the Management package.

Non-abstract stereotypes (name is not italicized) are used with classes and attributes that are created to model permissible values for the CC and BIE simple properties (the BCC and BBIE, respectively). Tag values are to be assigned for each tag definition according to the multiplicity when the UML class or attribute is instantiated. The rest of this section provides detailed definition, usage rules, and guidelines associated with these stereotypes.

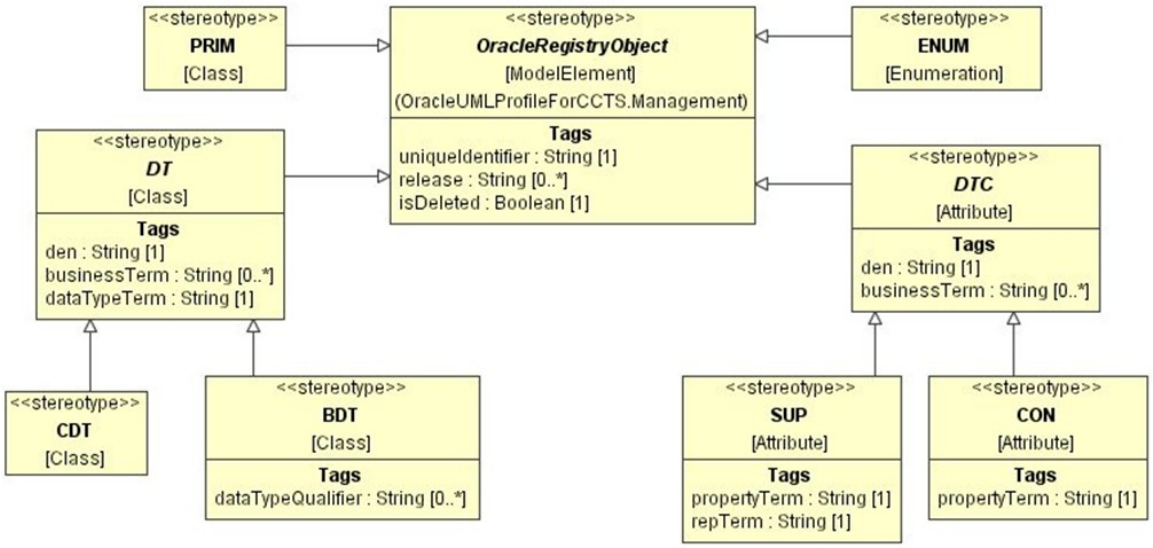

Figure 15: Stereotypes and tag definitions for model elements in the Common package

The <<DT>> Stereotype

Stereotype	DT (Abstract)
Base Class	Class (from UML 1.4.2 / 2.0 meta-model)
Parent	Management::OracleRegistryObject

<table>
<tr><td>Description</td><td colspan="4">This stereotype corresponds to the “Data Type (DT)” class in the CCTS meta-model. It is a superclass to provide tags, rules, and guidelines common to the <<CDT>> and <<BDT>> stereotypes.</td></tr>
<tr><td rowspan="4">Tag Definitions</td><td>Tag Name</td><td>Type</td><td>Multiplicity</td><td>Description</td></tr>
<tr><td>dataTypeTerm</td><td>string</td><td>1:1</td><td>This tag corresponds to the “Data Type Term” attribute of the “Data type (DT)” class in the CCTS meta-model. The tag semantically conveys the forms and the set of valid values for a simple business characteristic modeled by the BCC and BBIE.</td></tr>
<tr><td>den</td><td>string</td><td>1:1</td><td>This tag corresponds to the “Dictionary Entry Name (DEN)” attribute of the “Common Information” class in the CCTS meta-model. It is a slot for specifying an official name of a data type. See rules provided in each specific stereotype for how the DEN is derived.</td></tr>
<tr><td>businessTerm</td><td>string</td><td>0:n</td><td>This tag corresponds to the “Business Term” attribute of the “Common Information” class in the CCTS meta-model. It represents other commonly known names of the data type that are different from the official name in the “den” tag.</td></tr>
</table>

Rules and Guidelines

[RDTDataTypeTerm] The “dataTypeTerm” SHALL be one of the permissible representation terms in the Table 8-3 of the CCTS 2.01.

[GDTBizTerm] The “businessTerm” of a model element having one of the <<DT>> stereotypes (<<CDT>> or <<BDT>>) MUST comply with the rules relating to the CC Business Term in the Rules for Core Component Business Terms section of the CCTS 2.01.

[RDT] A UML class with one of the DT stereotypes MUST contain one and only one content component as a UML attribute of stereotype <<CON>> and at least one supplementary component as a UML attribute of stereotype <<SUP>>.

[RDTAttr] A UML class with one of the DT stereotypes (<<CDT>> or <<BDT>>) MUST NOT contain any UML attribute other than those of stereotype <<CON>> or <<SUP>>.

[RDTDef] A UML class having one of the <<DT>> stereotypes SHALL have UML documentation. This serves as a map to the "Definition" attribute of the Common Information class in the CCTS meta-model.

The <<CDT>> Stereotype

Stereotype	CDT			
Base Class	Class (from UML 1.4.2 / 2.0 meta-model)			
Parent	DT			
Description	This stereotype corresponds to the "Core Data Type (CDT)" class in the CCTS meta-model. The CDT is a complex data type that must be one of the approved types in the CCTS Data Type Catalog specification version 2.01. A CDT defines a generic value domain for a BCC property. A UML class with a <<CDT>> stereotype is regarded as a CDT class.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

Rules and Guidelines

[RCDT] All CDT classes MUST comply with the CCTS Data Type Catalog specification version 2.01.

[RCDTDEN] The "den" of the CDT class SHALL comply with the rules related to Core Component Type in the Core Component Rules for Dictionary Entry Name section of the CCTS 2.01. The term "Representation Term" in those rules SHALL be replaced with the term "Data Type Term." That is, the "den" of the CDT SHALL be syntactically derived from its "dataTypeTerm" by appending a dot character, a space character, and the word "Type."

For example, if a CDT class has a "dataTypeTerm" that is Code, then its "den" is Code. Type.

[RCDTAttrType] The type of a UML attribute of the CDT class MUST be a class of stereotype <<PRIM>> (defined in The <<PRIM>> Stereotype). This type association corresponds to the "Primitive Type" attributes of the "DT Content Component" and "DT Supplementary Component" classes in the CCTS meta-model.

The <<BDT>> Stereotype

Stereotype	BDT
Base Class	Class (from UML 1.4.2 / 2.0 meta-model)
Parent	DT
Description	This stereotype corresponds to the "Business Data Type (BDT)" class in the CCTS meta-model. The BDT is a complex data type that is a restriction of the CDT. A BDT defines a specific value domain for a BBIE property. A UML class with a <<BDT>> stereotype is regarded as a BDT

	class.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	dataTypeQualifier	string	0:n	This tag corresponds to the "Data Type Qualifier" attribute of the "Business Data Type (BDT)" class in the CCTS meta-model. The Data Type Qualifier prefixed to the Data Type Term semantically conveys a more restrictive value domain than the Data Type Term alone.

Rules and Guidelines

[RBDT] A BDT class SHALL comply with rules related to Data Types in the Data Types section of the CCTS 2.01. This particularly means compliance with the next rule [RBDTBasedOn].

[RBDTBasedOn] A BDT MUST have a Based-on dependency (as defined in the `Common` package) with one and only one CDT class. This relation corresponds to the "Basis" role of the CDT in the CCTS meta-model.

[RBDTDataTypeTerm] The "`dataTypeTerm`" of a BDT class MUST be identical to the "`dataTypeTerm`" of the CDT class on which it is based.

[RBDTDEN] The "`den`" of the BDT class SHALL comply with the rules related to Data Types in the Rules for Data Types for Dictionary Entry Name section of the CCTS 2.01. The terms "Data Type," "Representation Term," and "Qualifier Term" in those rules SHALL be replaced with the term "Business Data Type," "Data Type Term," and "Data Type Qualifier" when interpreting those rules. That is, the "`den`" of a BDT class SHALL be syntactically derived from its "`dataTypeQualifier`" tags, and "`dataTypeTerm`" by concatenating each "dataTypeQualifier" with an underscore and a space character, followed by the "`dataTypeTerm`" a dot character, a space character, and the word "Type."

For example, the "`den`" of a BDT class for which "`dataTypeTerm`" is `Code` and "`dataTypeQualifier`" tags include ["`ISO 4217`", "`Currency`"] is `ISO 4217_ Currency_ Code. Type`.

[RBDTAttrType] The type of a UML attribute of the BDT class MUST be a class of stereotype <<`PRIM`>> or <<`ENUM`>>. The <<`PRIM`>> type association corresponds to the "Primitive Type" attributes of the "DT Content Component" and "DT Supplementary Component" classes in the CCTS meta-model, while the <<`ENUM`>> type association corresponds to a form of "Component Restriction". Other forms of the Component Restriction will be represented by other means not yet addressed in this version of the UML profile.

[RBDTAttr] A BDT class MUST have the same set or subset of UML attributes as the CDT class on which it is based (however, the content component must always be included). The value domain of each attribute may be a restricted value domain of the corresponding CDT's attribute.

The <<DTC>> Stereotype

Stereotype	DTC (Abstract)			
Base Class	Attribute (from UML 1.4.2 / 2.0 meta-model)			
Parent	Management::OracleRegistryObject			
Description	This stereotype is a superclass to provide tags, rules, and guidelines common to the <<CON>> and the <<SUP>> stereotypes.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	den	string	1:1	This tag corresponds to the "Dictionary Entry Name (DEN)" attribute of the "Common Information" class in the CCTS meta-model. It is a slot for specifying an official name of the "DT Content Component" and "DT Supplementary Component" in the CCTS meta-model. See rules provided in each specific stereotype for how the DEN is derived.
	businessTerm	string	0:n	This tag corresponds to the "Business Term" attribute of the "Common Information" class in the CCTS meta-model. It represents other commonly known names/terms of the data type that are different from the official name in the "den" tag.

Rules and Guidelines

[GDTCBizTerm] The "businessTerm" of a model element having one of the <<DTC>> stereotypes MUST comply with the rules relating to the CC Business Term in the Rules for Core Component Business Terms section of the CCTS 2.01.

The <<CON>> Stereotype

Stereotype	CON			
Base Class	Attribute (from UML 1.4.2 / 2.0 meta-model)			
Parent	DTC			
Description	This stereotype corresponds to the "DT Content Component" class in the CCTS meta-model. The DT Content Component attribute carries the actual content of its CDT and BDT. A UML attribute with stereotype <<CON>> is regarded as a CON attribute.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	propertyTerm	string	1:1	This tag corresponds to the "Property Term" attribute of the "DT Content Component" class in the CCTS meta-model. The value is fixed to the word "Content." It is used to create the "den" of the DT Content Component.

Rules and Guidelines

[GCONPropTerm] The "propertyTerm" of a CON attribute MUST always be the word "Content."

[RCONDEN] The "den" of a CON attribute is the "dataTypeTerm" of its parent class suffixed by a dot character, a space character, and its "propertyTerm".

For example, the "den" of a CON attribute of a Code. Type CDT is Code. Content.

The <<SUP>> Stereotype

Stereotype	SUP			
Base Class	Attribute (from UML 1.4.2 / 2.0 meta-model)			
Parent	DTC			
Description	This stereotype corresponds to the "DT Supplementary Component" class in the CCTS meta-model. The DT Supplementary Component provides additional information, which allows for the content component of a CDT and BDT to be meaningful. A UML attribute having the <<SUP>> stereotype is regarded as a SUP attribute.			
Tag	Tag Name	Type	Multiplicity	Description

Definitions	propertyTerm	string	1:1	This tag corresponds to the "Property Term" attribute of the "DT Supplementary Component" in the CCTS meta-model. This Property Term provides supporting information to the content of the CDT or BDT so that their content is (computationally) meaningful.
	repTerm	string	1:1	This tag corresponds to the "Representation Term" attribute of the "DT Supplementary Component" class in the CCTS meta-model. It indicates the value space of the supplementary component.

Rules and Guidelines

[RSUP] A SUP attribute MUST be one of the approved Supplementary Components for the related Core Data Type according to the CCTS Data Type Catalog specification 2.01.

[GSUPPropTerm] The "`propertyTerm`" SHALL follow the same guideline associated with the "`propertyTerm`" of the BCCP class as defined in [RBCCPProp].

[RSUPUnique] The SUP attribute's "`propertyTerm`" MUST be unique within a CDT or a BDT class.

[GSUPRepTerm] The "`repTerm`" of the SUP attribute MUST be one of the approved Data Type Term in the CCTS Data Type Catalog specification 2.01.

[RSUPDEN] The SUP attribute's "`den`" SHALL correspond to the DEN of the supplementary component it represents in the CCTS Data Type Catalog 2.01.

[RSUPAttr] The UML attribute name of the SUP attribute SHALL be syntactically derived from its "`den`". That is, it MUST be a lower camel case concatenation of the property term and the representation term sections of its "`den`" with these additional rules.

Drop word or words in the property term section if they are the same as those in the representation term section. Words of different parts of speech, such as nouns and verbs, are considered the same.

Abbreviate the word "Identifier" in the representation term section to `ID`.

Drop the CCTS data type term word in the representation term section when it is "Text," or "Numeric" and its derivatives except "Rate," or "Binary Object" and its derivatives.

The <<PRIM>> Stereotype

Stereotype	PRIM			
Base Class	Class (from UML 1.4.2 / 2.0 meta-model)			
Parent	Management::OracleRegistryObject			
Description	A PRIM (Primitive Type) stereotyped class represents one of the CCTS Core Data Type Catalogue-defined primitive types. A UML class having a <<PRIM>> stereotype is regarded as a PRIM class.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	None			

Rules and Guidelines

[RPRIM] A PRIM class MUST correspond to one of the approved primitives in the CCTS Data Type Catalog specification 2.01.

[RPRIMUse] A PRIM class MUST only be used as a UML attribute type of the CON or SUP attribute.

The <<ENUM>> Stereotype

Stereotype	ENUM			
Base Class	Enumeration (from UML 1.4.2 / 2.0 meta-model)			
Parent	Management::OracleRegistryObject			
Description	The ENUM (Enumeration Type) stereotyped class represents an enumeration type. The ENUM stereotype is used to define a restriction on either a content or supplementary component of a BDT. It is a form of the "Component Restriction" class in the CCTS meta-model. A UML enumeration having the <<ENUM>> stereotype is regarded as an ENUM enumeration.			
Tag Definitions	Tag Name	Type	Multiplicity	Description
	N/A			

Rules and Guidelines

[RENUMUse] An ENUM enumeration MUST only be used as a UML attribute type of the CON or SUP attribute of the BDT class.

Additional Constructs

This section indicates other constructs that are not explicitly shown in the UML profile definitions.

The UML Generalization

The UML generalization is also regarded as a specialization, extension, or subclass in a reversal way. That is, a statement "class A is a generalization of another class B" can also be stated as "class B is a specialization/extension/subclass of class A".

The UML generalization corresponds to the “Generalization” and “Extension” roles on the “Aggregate Core Component (ACC)” and “Aggregate Business Information Entity (ABIE)” classes in the CCTS meta-model.

The UML generalization is used to relate two ACC or two ABIE classes that are semantically (or conceptually) inclusive of one another. It is also an implication that the respective ABIE classes, which are based on two ACCs having a generalization relationship, also hold the semantically inclusive meaning regardless of the business contexts.

[RGEN] The UML generalization relation MUST be used only between either a pair of ACC classes or a pair of ABIE classes.

Generalization is primarily used for industry extensions and specific design patterns. The industry extension is described in Industry Extensions and Their Packages and other design patterns are described in Oracle CCTS UML Principles and Guidelines.

One of the formal consequences of the UML generalization relationship is the attributes and associations inheritance. This formal semantics is held both when applying the UML generalization between ACC classes and between ABIE classes.

The generalization is transitive. That is, if `A` is a subclass of `B` and `B` is a subclass of `C`, then `A` is also a subclass of `C`. This implies, for example, that `A` inherits attributes and associations from `B` and also from `C`.

Another formal consequence of the UML generalization relationship is the subsumption semantics. That is, if class `B` is a subclass of class `A`, then an instance of class `B` is an instance of class `A`. This semantics will be used only informally to restrict interpretation of the model. However, there is no intention to convey that mapping of the super class can be reused in the mapping of the subclass. On the other hand, the relationship can be used syntactically when generating a syntax-specific data structure specification. That is, the outcome will be different from using another UML relationship. Additional detail about this will be given in Oracle CCTS UML Principles and Guidelines.

Other formal consequences of the UML generalization, such as operation inheritance, public/private accessibility, and so on, are out of scope and will not influence the data model.

Version Suffix in the UML Names

The “Version Identifier” attribute of the “Registry Class” in the CCTS meta-model is implemented as suffix to the UML names of model elements. This is an optional practice. It can be used as an interim solution if the UML modeling tool does not support versioning of individual model elements. Versioning provides detail about this practice. It also serves as a documentation of the versioning functional requirements.

Other General UML Usage Rules

This section documents additional rules, which are general to the Oracle CCTS UML data model.

[RCCTSStereotypes] Stereotypes that are not defined in this document SHALL not be used in the Oracle CCTS UML data model.

[ROneStereotype] All model elements MUST have one and only one stereotype among those defined in this UML profile.

Chapter 4: Oracle CCTS UML Principles and Guidelines

This section discusses association types and modeling patterns used within the UML profile.

Association Types

Three association types are used within this UML profile. They have been referred to as regular, composition, and aggregation. The use of association within this profile is limited to unidirectional association to explicate the role of object, property, or representation.

Only the origin association end can have either one of the three association types, while the association type on the destination end is always regular. This section gives general guidelines to the use of these three different association types on the origin association end. Specific interpretations in particular usage contexts are also given. The Modeling Patterns section also gives additional guidelines to the usages of these association types.

Aggregation Association Type

The aggregation association type is used specifically in this profile to mean that the class on the aggregation end (the end with the white diamond) refers to the class on the aggregated end by using an identity reference. It is also implied that the class on this aggregated end is *shared*. This association type is used primarily in the object reference pattern, such as to create a `PartyReference` ACC or to indicate that a `ShipToParty` ASCCP is a reference to an instance of a `Party` ACC (a role that a party plays in a particular business transaction/collaboration). The aggregation association type implies that a syntax-specific data structure specification must allow for exchange of the identity of the ACC class on the aggregated end by an identifier or a combination of identifiers. More detail about this will be given in the Modeling Patterns section.

Regular Association Type

A unidirectional regular association type is used specifically in this profile to mean that the class on the origin association end (the associating class) refers to the class on the destination association end (the associated class). In contrary to the aggregation association type, in this case the reference is by content. The affect, however, is the same in that the associated class is *shared*. That is, the associating class will repeat information in an instance of the associated class. With this semantics, the implication is that a class must exist that is an owner of that associated class (which may be that class itself). The "`owner`" tag, defined in the <<`ACCP`>> and <<`ABIEP`>> stereotypes (including their specialized stereotypes), is used to indicate the owner class. This semantics of the regular association type is used primarily in the object reference pattern described later in one of the Modeling Patterns.

As a brief example, a PartyReference ACC class may refer to one of the addresses in the Party ACC class. When this reference needs to be done by content, a regular ASCC association line is drawn from the PartyReference ACC class to an Address ASCCP class and indicates in the "owner" tag that the Party ACC is the owner of that property. This association link also implies that the information about an Address within an instance of a PartyReference ACC must correspond to one of the Address instances of a particular Party ACC.

Choosing between the aggregation and regular association type is quite discrete. If something cannot be referenced by identity, then it must be referenced by content. In effect, a BCC is always referenced by content. In another situation, when additional information beyond the identity information that cannot be obtained by dereferencing the identity is necessary, then the regular association is the choice. In the example above a Party instance can have multiple Address instances. Although the PartyReference refers to a particular Party instance, it must have only one Address instance (e.g., to indicate where exactly to ship to). It is clear that if the PartyReference instance only has an identity of the referred to Party instance, it is not possible to figure out which instance of the Address instance to use. One could argue for why not just exchanging the Address identity (through the Party identity). In this case, the justification is based on 1) whether there is a business requirement to keep the detail of the Address or 2) whether there is a service interface to deference the Address identity.

Also note that the regular association automatically implies that a syntax-specific data structure specification must allow for a full content of the class on the destination association end to be exchanged.

Composition Association Type

A composition association type is used to imply an ownership of information (or object instance). After an instance of an owned-by class is attached to the owner class (the class that has the solid diamond), that cannot be changed throughout the life of the owned-by class. That is, if an owner instance gets deleted, then the owned-by class will also cease to exist.

In this UML profile, a specific semantics is also made with the multiple composition associations. When multiple composition associations are directed toward a class, the interpretation is always as a mandatory and mutually exclusive choice of ownership (this is different from the typical interpretation where a conjunction interpretation would). That means an instance of a composited class must be owned/created as part of one and only one of the compositing classes—it must not be more than one and one (not zero ownership) must exist.

Consider a simple class diagram in Figure 16a. Class A and B each have a composition association with class C (note that in this case classes A and B are said to be compositing classes and class C is a composited class). That is, two composition associations are directed toward class C. According to the prescribing semantics, this diagram carries a constraint that an instance of class C must be instantiated as part of either an instance of class A or class B but not both, and it cannot be instantiated by itself either (even the cardinality on the composition association ends indicate optional). Consequently, the diagram and Figure 16a is logically equivalent to the diagram in Figure 16b. This is different from the typical interpretation where it would mean an instance of Class C can be instantiated also by itself or owned by instances of Class A and C in Figure 16a and Class C must be own by instances of Class A and C.

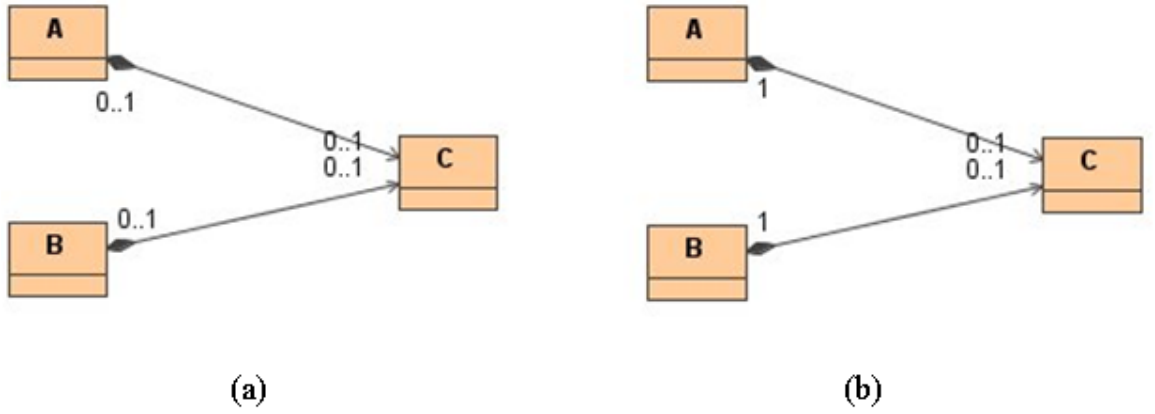

Figure 16: Simple composition associations

Similar to the regular association type, the composition association type also automatically implies that a syntax-specific data exchange specification must allow for the full content of the composited class to be exchanged.

Summary of Association Types Applicability and Specific Meaning

Table 1 provides a summary of how different association types are used with CCTS-based model elements and specific effects in each particular usage. Although the table mentioned only families CC model elements, the same applies to BIE model elements that are based on each of those CC model elements.

Table 1: Summary of association types applicability and specific meaning

Relation Type/ Association Type	ASCCPRep[(a)]	ASCC/CASCC/SASCC[(b)]	BCCPRep[(c)]	BCC[(d)]
Aggregation	This is primarily used in the case in which an ASCCP class represents a role of the associated ACC class in a particular transaction/operation context. The consequences are that no class/object can own an instance of the ACC under the property represented by that ASCCP (that is, that property is always a reference and no ACC class can have a composition association with the	This is primarily used in two cases: 1) when an ACC class associates with an ASCCP class, which represents a role of another ACC class; or 2) when an ACC class references a property and associated ACC owned by another class. In both cases, it specifically indicates that the reference will be done using identity. That is, a syntax-specific data structure specification shall provide a means to exchange identity of the ACC class associated with the ASCCP class. It indicates an existence of	Not used; no specific semantics.	Not used; no specific semantics because a BCCP class cannot be referenced using identity.

Relation Type/ Association Type	ASCCPRep[a]	ASCC/CASCC/SASCC[b]	BCCPRep[c]	BCC[d]
	ASCCP), and it also indicates an existence of referenced information from an instance of an associated ACC class.	identity information from an instance of the ACC class indicated as the owners.		
Regular	Not used; no specific semantics.	This use is similar to the aggregation; however, in this case the reference is to be done using content. Because this is a reuse/recast of information, a syntax-specific data structure specification must provide a means to exchange the content of the associated property. It indicates an existence of referenced information from an instance of the ACC class indicated as the owners.	Not used; no specific semantics.	This is used primarily when an ACC reference class reuses/recasts a BCC that is owned by another ACC class. It indicates an existence of referenced information from the ACC class(es) indicated as the owner(s).Note that this association type is not symbolized in the class diagram; however, it is signified when a BCC has the "`owner`" specified.
Composition	When the ASCCPRep association is a composition, one or more ACC classes can have composition associations with the ASCCP class involved in that ASCCPRep association (when more than one exists, the mandatory and mutually exclusive semantics apply as described in Composition Association Type). It is imperative that at	An instance of the associating ACC class can be a sole owner of the property and an instance of the ACC class associated with the ASCCP class. The association from an ACC class to an ASCCP class cannot be a composition if the association from the ASCCP to its (representation) ACC class is an aggregation.	This is always the case because a BCC is always owned or is intrinsic to its parent object, and that reference to a BCC must always be going through one or more ACC classes.	This is a typical ACC to BCCP usage and conveys the ACC class ownership of the BCC attribute. Note that this association type is not symbolized in the diagram; however, it is signified when a BCC does not have the "`owner`" specified.

Relation Type/ Association Type	ASCCPRep[a]	ASCC/CASCC/SASCC[b]	BCCPRep[c]	BCC[d]
	least one ACC class must exist in the model having a composition association with that ASCCP class. There can be other ACC classes having a regular or aggregation association with the ASCCP class though.			

(a): From ASCCP to ACC.

(b): From ACC to ASCCP/CCC, CCC to ASCCP/CCC/SCC, SCC to ASCCP/CCC. Note that the CCC and SCC should be viewed just as middlemen.

(c): From BCCP to CDT.

(d): From ACC/CCC/SCC to BCCP. Note that the CCC and SCC should be viewed just as middlemen.

Modeling Patterns

This section provides reusable modeling patterns for modeling different kinds of concepts. It also provides a set of comprehensive examples to demonstrate the usage of the UML profile.

Typical Core Component Pattern

A typical core component pattern has all its associations as the composition type. That is, all associated object instances are owned. Class diagrams in Figure 18 and Figure 17 illustrate examples of the typical core component pattern. They include definitions for the `Identification` ACC and the `Revision` ACC. The `Revision` ACC is used within the `Identification` ACC. All related BCCPs and CDTs are also included in the diagrams.

The `Revision` ACC can be used by another ACC to capture metadata about changes to object the ACC represents.

The `Identification` ACC can be used by another ACC to capture information to uniquely identify the object that the ACC represents. It includes several other identifications which, when combined, can uniquely identify an object among heterogeneous systems.

The `Revision` ACC has only simple properties; hence, it has only BCCs. On the contrary, the `Identification` ACC has both simple and complex properties; hence, it has both ASCCs and BCCs. We will first describe the detail construction of the `Revision` ACC and then the `Identification` ACC.

Recall that the association type between BCCP and CDT is always composition as reflected in the diagram. Because all properties of the `Identification` ACC live and cease with it, all of its association types are compositions. Also, recall that all BCCs that have no "`owner`" specified are of composition association type.

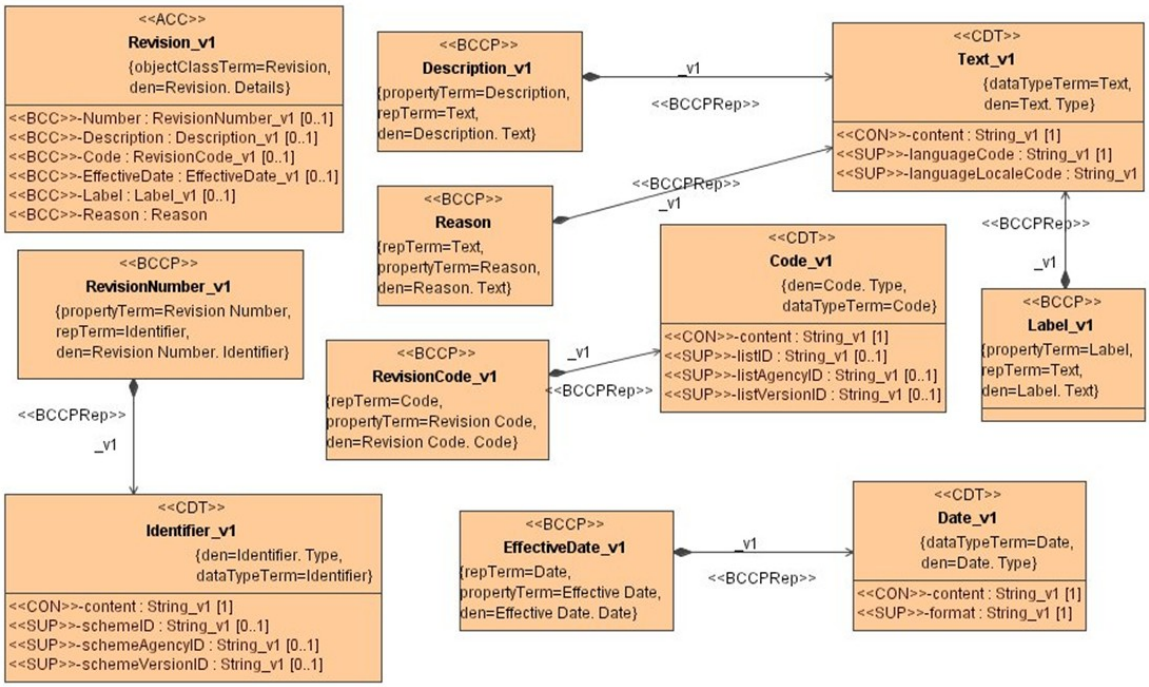

Figure17: UML class diagram of the `Revision` ACC

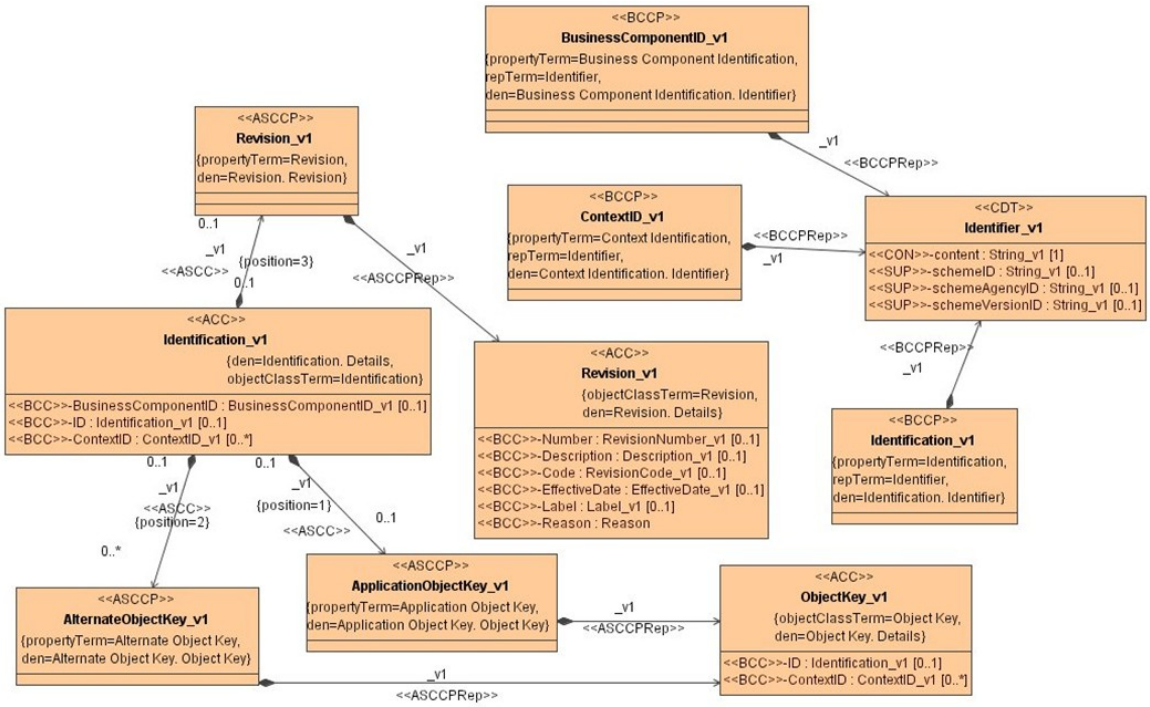

Figure 18: UML Class diagram of the `Identification` ACC

Deciding whether to specify the BCCP property term or the BCC UML attribute names of an ACC first is not easy. Documenting the UML attribute names of the ACC first seems more natural. Then you would create BCCP classes with property term, CDT, and DEN. And then you could use these terms to work backward and check for compliance with the BCC attribute naming derivative rules. Table 2, Table 3, and Table 4 provide detailed discussion about the BCC and BCCP naming for properties of the `Revision` ACC. Refer to related naming rules in the <<`BCCP`>> and <<`BCC`>> stereotype definitions in The <<BCCP>> Stereotype and The <<BCC>> Stereotype, particularly the rules [RBCCPDEN], [RBCCPClassName], [RBCCDEN], and [RBCCAttr]. Table 8 and Table 9 provide detailed discussion about the ASCCP and ASCC naming for properties of the `Identification` ACC. Refer to related naming rules in the <<`ASCCP`>> and <<`ASCC`>> stereotype definitions in The <<BCCP>> Stereotype and The <<BCC>> Stereotype, particularly the rules [RASCCPDEN], [RASCCPClassName], and [RASCCDEN]. Note that Table 9 provides no discussion, because no truncation occurs because no word at the end of the object class term is repeated at the beginning of the property term section of the ASCCPs. The field name of the ASCC in a syntax-specific data structure specification depends on specific NDRs. Some may choose to include both the object class term and property term sections, while some may include only the property term sections (or the choice may depend on other model data).

Table 2: BCC UML attribute names of the Revision ACC

BCC UML Attribute Names	Attribute Description (BCC dictionary definition)	Attribute Name Discussion
Number	Unique identifier for the revision.	A business term of this BCC may be ID and Identifier.
Description	Descriptive information about the revised object.	As a general guideline, a name/term should be a noun phrase.
Code	A finite set of revision identifiers. This is an alternative way of identifying a revision as opposed to using the indefinite ID. For example, some application objects may go through finite stages of revisions, for example, pre-beta, beta, and release.	As a general rule, you do not need to repeat the object class term in its attribute name (note that attribute name may be different from property term).
EffectiveDate	Date and time at which the revision is effective	Representation of the attribute is almost always part of the attribute name except when it is Text, Numeric and its derivatives (for example, Integer), and Binary Object. This is because the term for which representation is one of those exceptions in many cases readily conveys what the representation is.
Label	A short text labeled for ease of recognizing the revision.	
Reason	Descriptive reason for changes associated with the revision	

Table 3: BCCPs used by the Revision ACC

BCCP Property Term	CDT Data Type Term	Property Term Discussion	BCCP DEN*/ BCCP UML Class Name**
Revision Number	Identifier	Generally, the property term should not be just `Number`, because it does not add on any semantics to the CDT's data type term (in principle the CDT does not have business semantics). If the property term were specified as `Number`, then the BCCP's DEN would be `Number. Identifier`. One would not even be able to make any sense out of the name as to what the property is about. The property term should generally convey both characteristics and value domain. However, sometimes value domain can be implied or commonly known from the characteristics. This can, however, be language-specific, domain–specific, or both. With the `Revision Number` as a property term, the application of the [RBCCAttr] BCC attribute naming rule will still yield the `Number` as the attribute name.	Revision Number. Identifier
			RevisionNumber (If the `Revision Identification` were chosen as the property term, the class name would be `RevisionID` as the term `Identification` gets abbreviated)

BCCP Property Term	CDT Data Type Term	Property Term Discussion	BCCP DEN*/ BCCP UML Class Name**
Description	Text	First, the data type term `Text` is not part of the property term according to the BCCP property term rule. Then, `Revision Description` property term is not necessary because the term `Description` by itself already provides additional semantics to the CDT. The `Description` property term would also allow for the BCCP to be more reusable. In addition, because its dictionary definition is given as "Description is a human understandable detailed explanation about *the subject the associated parent object represents*," the term `Description` seems sufficiently representative of the property. That is, when reusing the property as different BCCs, no need exists to add any additional word to make the property representative of its definition except replacing the phrase "*the subject the associated parent object represents*" with the ACC's object class term.	Description. Text
			Description

BCCP Property Term	CDT Data Type Term	Property Term Discussion	BCCP DEN*/ BCCP UML Class Name**
Revision Code	Code	The property term should not simply be `Code` because no semantics would be added on to the CDT. With the `Revision Code` as a property term, the application of the [RBCCAttr] BCC attribute naming rule will still yield the `Code` as the attribute name.	Revision Code. Code
			RevisionCode
Effective Date	Date	The property term should not simply be `Effective` because that would result in an incomplete noun phrase. Similar to the BCC attribute name, representation term of the BCCP is almost always part of the property term except when it is Text, Numeric and its derivatives (for example, Integer), and Binary Object.	Effective Date. Date
			EffectiveDate
Label	Text	First, the data type term `Text` is not part of the property term according to the BCCP property term rule. Then, the term `Label` seems to sufficiently represent the meaning of the	Label. Text

BCCP Property Term	CDT Data Type Term	Property Term Discussion	BCCP DEN*/ BCCP UML Class Name**
		property when its dictionary definition is "Label is a brief, mnemonic name of the *parent object* used primarily for presentation purpose" and the same reasoning as the Description applies.	Label
Reason	Text	If the CDT is `Code`, the property term would have been `Reason Code`. For similar reasons to the Description and Label, the word `Revision` and `Text` are not needed in the property term.	Reason. Text
			Reason

* Application of the rule [RBCCPDEN], ** Application of the rule [RBCCPClassName]

Table 4: The Revision ACC's BCC DEN and UML attribute name derivations

BCCP DEN	BCC DEN* (`Revision` is the object class term)	BCC DEN Discussion	BCC UML Attribute Name**
Revision Number. Identifier	Revision. Number. Identifier	The word `Revision` in the property term section of the BCCP DEN is truncated because it is repeated in the object class term section.	Number (If the BCC DEN were `Revision. Identification. Identifier`, then the attribute name would be `ID` because the term `Identification` gets abbreviated)
Description. Text	Revision. Description. Text		Description
Revision Code. Code	Revision. Code. Code	The word `Revision` in the property term section of the BCCP DEN is truncated because it is repeated in the object class term section.	Code
Effective Date. Date	Revision. Effective Date. Date		EffectiveDate
Label. Text	Revision. Label. Text		Label

BCCP DEN	BCC DEN* (`Revision` is the object class term)	BCC DEN Discussion	BCC UML Attribute Name**
Reason. Text	Revision. Reason. Text		Reason

* Rule [RBCCDEN], ** Rule [RBCCAttr]

Table 5: BCC UML attribute names of the Identification ACC

BCC UML Attribute Names	Attribute Description (BCC dictionary definition)
BusinessComponentID	Unique Key for the application agnostic representation of the object instance. Business Documents generated by AIA applications will have the BusinessComponentID necessarily populated. The BusinessComponentID will be generated using the API provided by AIA Infrastructure.
ID	Business Friendly Identifier found in the participating application for this object instance. Business documents generated by AIA applications will have these populated wherever they are applicable. PO Number, Order Number are some of the examples
ContextID	Optional element to identify additional qualifiers for the ID. Used in the case of multipart keys, for example, if an Item is unique within a set, then the Item Number would be the ID and the set ID value would be a ContextID value

Table 6: BCCPs used by the Identification ACC

BCCP Property Term	CDT Data Type Term	Property Term Discussion	BCCP DEN*/ BCCP UML Class Name**	BCCP DEN Discussion/ BCCP UML Class Name Discussion
Business Component Identification	Identifier		Business Component Identification. Identifier	
			BusinessComponentID	The word `Identification` in the property term is abbreviated to `ID`.
Identification	Identifier	Generally, the property	Identification. Identifier	

BCCP Property Term	CDT Data Type Term	Property Term Discussion	BCCP DEN*/ BCCP UML Class Name**	BCCP DEN Discussion/ BCCP UML Class Name Discussion
		term should not be just `Identification`, because no business semantics is added to the CDT. The object class of (one of its) ACCs using it is Identification, and it does not make sense to have an `Identification Identification` as a property term. This may indeed signal that the object is inappropriately modeled or the desired attribute name is inadequate. See further discussion in the next row.	Identification	According to the BCCP UML class naming rule, the class name should be `ID`.
Context Identification	Identifier	The BCC definition indicates that the `ContextID` is used with the `ID` property as a key combination. This may indicate that the `ID` and `ContextID` should be grouped together into another component. This can result in a better property term for the `Identification` property in the previous row.	Context Identification. Identifier	
			ContextID	The word `Identification` in the property term is abbreviated to `ID`.

Table 7: Revision ACC's BCC DEN and UML attribute name derivations

BCCP DEN	BCC DEN*	BCC DEN Discussion	BCC UML Attribute Name**	BCC UML Attribute Discussion
Business Component	Identification.		BusinessComponentID	The word

BCCP DEN	BCC DEN*	BCC DEN Discussion	BCC UML Attribute Name**	BCC UML Attribute Discussion
Identification. Identifier	Business Component Identification. Identifier			`Identification` in the property term section is abbreviated to `ID`.
Identification. Identifier	Identification. Identification. Identifier	The `Identification` in the property term cannot be truncated because that would result in the whole property term being truncated.	ID	The word `Identification` in the property term section is abbreviated to `ID`.
Context Identification. Identifier	Identification. Context Identification. Identifier		ContextID	The word `Identification` in the property term section is abbreviated to `ID`.

Table 8: ASCCPs used by the Identification ACC

ASCCP Property Term	Associated ACC	Property Term Discussion	ASCCP DEN*/ ASCCP UML Class Name**
Revision	Revision	An ACC's object class term already has business semantics; so generally, property term of an ASCCP can be the same as the object class term of the ACC it uses.	Revision. Revision
			Revision
Application Object Key	Object Key	The `Object Key` is an ACC for which the dictionary definition is "A key to an object instance". The term `Application` adds on semantics to the `Object Key` that "It is a key generated by the participating application (in the integration)."	Application Object Key. Object Key
			ApplicationObjectKey
Alternate Object Key	Object Key	The term `Alternate` adds on semantics to the	Alternate Object Key. Object Key

ASCCP Property Term	Associated ACC	Property Term Discussion	ASCCP DEN*/ ASCCP UML Class Name**
		term `Object Key` that "Object keys generated by others are to be put here."	AlternateObjectKey

Table 9: Identification ACC's ASCC DEN

ASCCP DEN	ASCC DEN*
Revision. Revision	Identification. Revision. Revision
Application Object Key. Object Key	Identification. Application Object Key. Object Key
Alternate Object Key. Object Key	Identification. Alternate Object Key. Object Key

Choice Pattern

Occasionally a need arises to delineate alternatives between two types of contents. In effect, a choice exists that reflects a branching of business knowledge in one direction or another.

The <<`CCC`>> stereotyped class (the CCC class) allows the choice to be defined, and the <<`CASCC`>> stereotyped association (the CASCC association) allows the choice to be used. The CCC class and CASCC association indicates that the relationship is structural.

For simple use cases, a CCC class can have BCC attributes and ASCC associations directly to represent each alternative. However, more complex constructs exist that need additional direction. One such use case is when a particular alternative needs to be a series of properties (ASCCs, BCCs, or both). In this case, the <<`SCC`>> and <<`SASCC`>> stereotypes (the SCC class and SASCC association) become handy.

In this section, examples of the CCC class, CASCC association usages are illustrated. The case where the SCC class and SASCC association are used is also illustrated.

Figure 19 illustrates example usages of the four stereotypes where various ways of communications can be specified using structured or unstructured content. The `Communication` ACC class has a `CommunicationChoice` CCC class and the association between them is the CASCC association. This indicates the intention of the modeler to create a data structure with alternatives and that the `CommunicationChoice` class need not be generated as named data structure in a syntax-specific data structure specification of the model. Three communication alternatives are allowed in the example.

1. By postal mail: The postal mail is specified by the `PostalMailAddress ASCCP`, which in turn is represented by the `Address ACC`.

2. By internet communication: The `InternetCommunication` class shows an example usage of the SCC class and SASCC association. It indicates the grouping of simple and aggregate components. The intention of the data modeler is that the internet communication can be an email, a chat, or both. The chat information itself needs more than one property to specify. The intention of the data modeler who uses the Sequence here is to indicate that these properties do not constitute a business concept. A side effect of that is that no property or generalization can be built upon the Sequence. The by-product is also that neither the SCC class nor the SASCC association needs a corresponding named data structure in a syntax-specific data structure specification, and the `Email` and the `ChatCommunication` can be direct children of the `Communication` class/structure in a data exchange message instance. The name of a sequence class is for ease of understanding and readability. Note that the SCC class can have more BCC attributes as necessary to constitute a single alternative.
3. By telephone: The telephone can be specified in two ways:
    a. By a single long phone number: the `CompletePhoneNumber` BCC
    b. By a sequence of components in which the phone number is broken down: The `StructuredPhoneNumber` class could have been assigned the <<`SCC`>> stereotype. The `StructuredPhoneNumber` provides excellent examples on BCCP and BCC naming that readers may want to pay attention to. However, if this option is chosen, the `StructuredPhoneNumber` class would not be reusable as a logical component having its own business semantics.

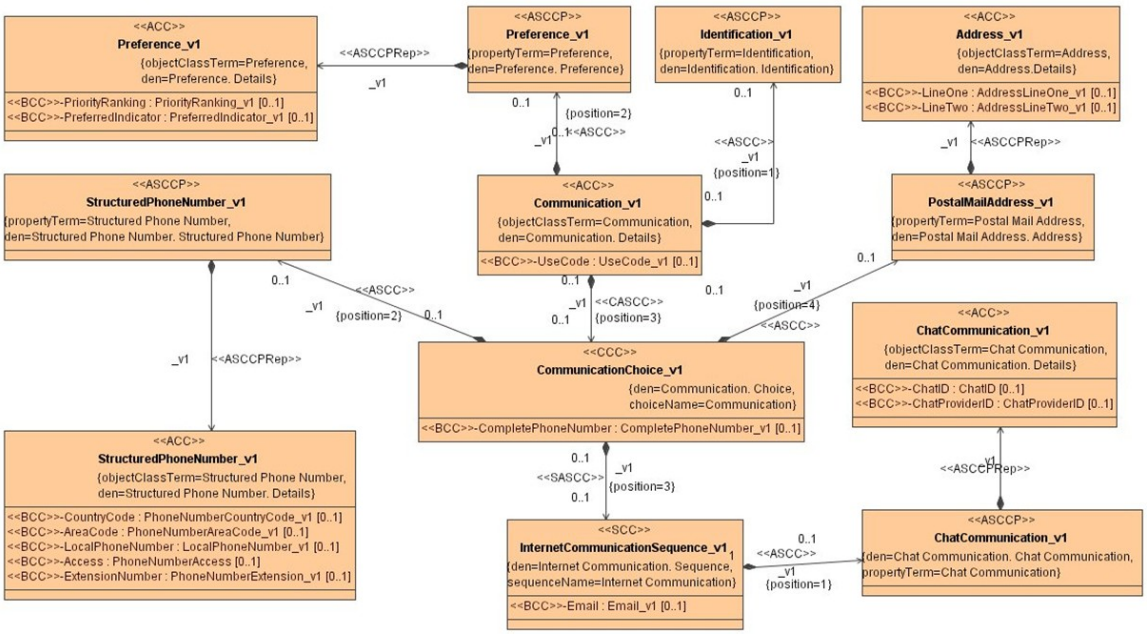

Figure 19: An example model illustrating usages of Choice and Sequence

While no need exists to name the SCC and CCC classes because structure grouping exists, we require that the two types of classes be named for good practice leading toward the ease of understanding of the model. As in the example in Figure 19, the model is easier to understand with names of the SCC and CCC classes.

Note that because the association type from the `Communication` ACC class and the `CommunicationChoice` class is composition, the association types between the `CommunicationChoice` class and its children must also be composition.

Consider the example in Figure 20, which may be considered as an alternative to the model in Figure 19. The `CompletePhoneNumber` BCC is moved into the `UnstructuredPhoneNumber` SCC class. This model violates the rule regarding the use of the SCC class, in that it must have two or more properties. This rule indeed prevents unnecessary use of the SCC class.

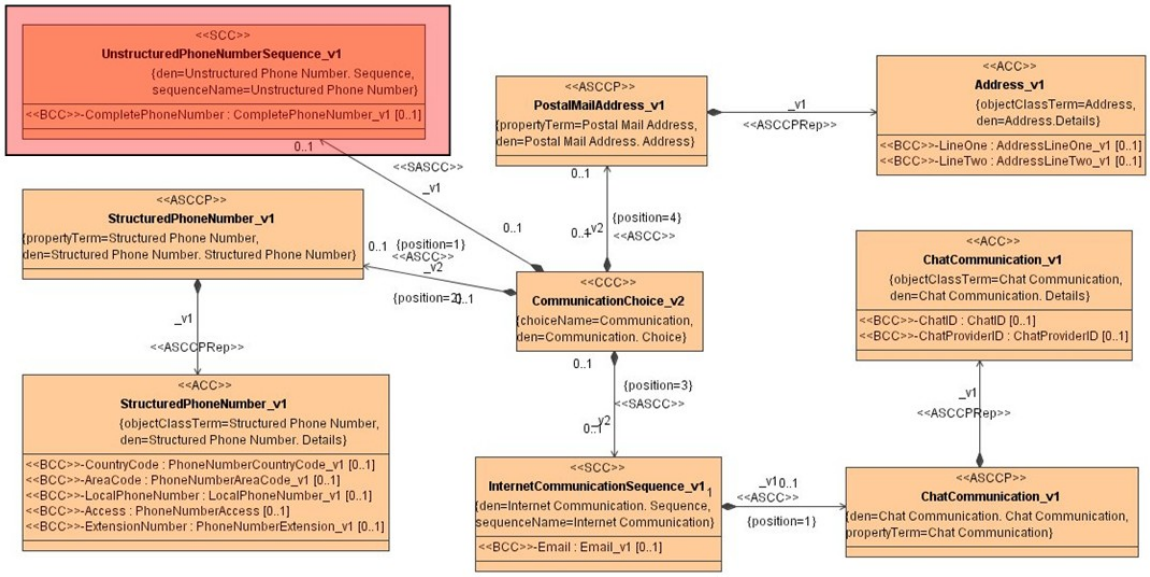

Figure 20: An incorrect usage of the SCC class

On the contrary, consider the examples (A) and (B) in Figure 21. The two models are not semantically the same. The model in (A) would result in three choices while the model in (B) would result in only two choices.

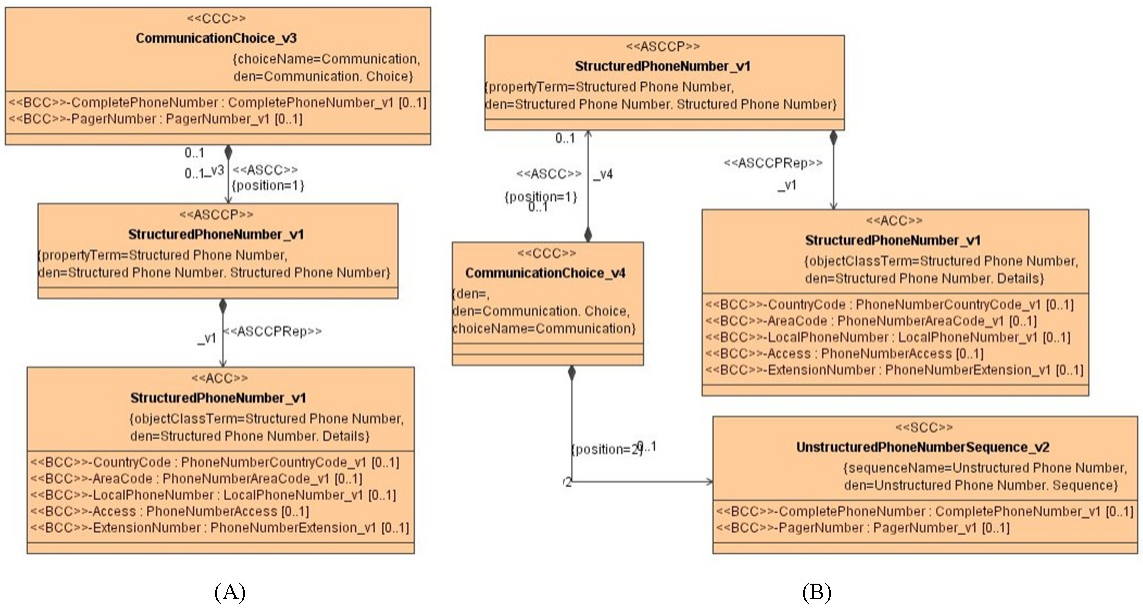

Figure 21: An example illustrating two similar but different choice models

Object Role Constraint Pattern

Sometimes expressing concepts as different roles of a particular object (another concept) is necessary. Such expression clarifies that instances of those concepts must have a corresponding instance of that particular object. Figure 22 includes two examples of object role constraints. The `ShipToParty` ASCCP and the `ShipFromParty` ASCCP are two object role constraints of the `Party` ACC. Note the use of the aggregation association type from each of the two ASCCPs to the `Party` ACC. This means that `Party` instances of the `ShipToParty` property or of the `ShipFromParty` property can in fact talk about the same instance of the `Party` ACC. In many implementation situations, `Party` instances of the `ShipToParty` and `ShipFromParty` can also be validated with a master `Party` data. That is, information in a `ShipToParty` or `ShipFromParty` property in a particular instance of its parent ACC (which has an association with the `ShipToParty` or `ShipFromParty` ASCCP) must correspond to information in a `Party` instance somewhere. However, an ACC may not want to simply refer to an identity of the `Party` instance associated with the `ShipToParty` or `ShipFromParty` ASCCP, because in many cases the need exists to indicate and keep different pieces of specific information from the master for each particular role (that is, when an object is under different usages) in different transactions. This requirement is where the reference object pattern applies. This pattern is described next.

Note that this pattern is called Object Role Constraint as opposed to just Object Role, because generally every ASCCP can be considered an object role. However, in this case a constraint is placed upon the ASCCP that no ACC can own an instance of the ACC object associated with that ASCCP (no composition association to that ASCCP is allowed).

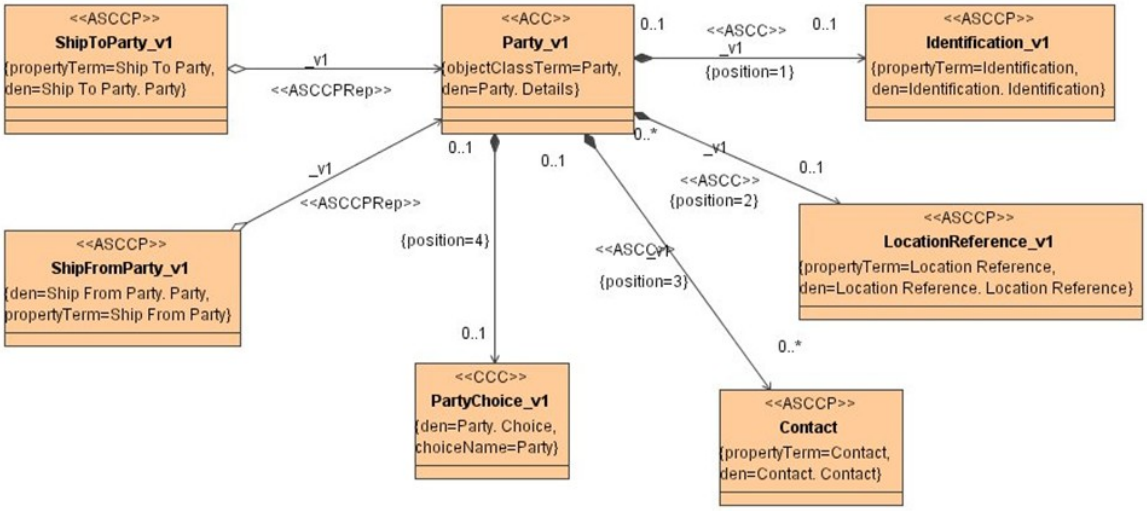

Figure 22: Examples of object role constraint pattern

Reference Object Pattern

The reference object pattern is used in conjunction with the object role constraint pattern to indicate the desired data structure of the object role (the property).

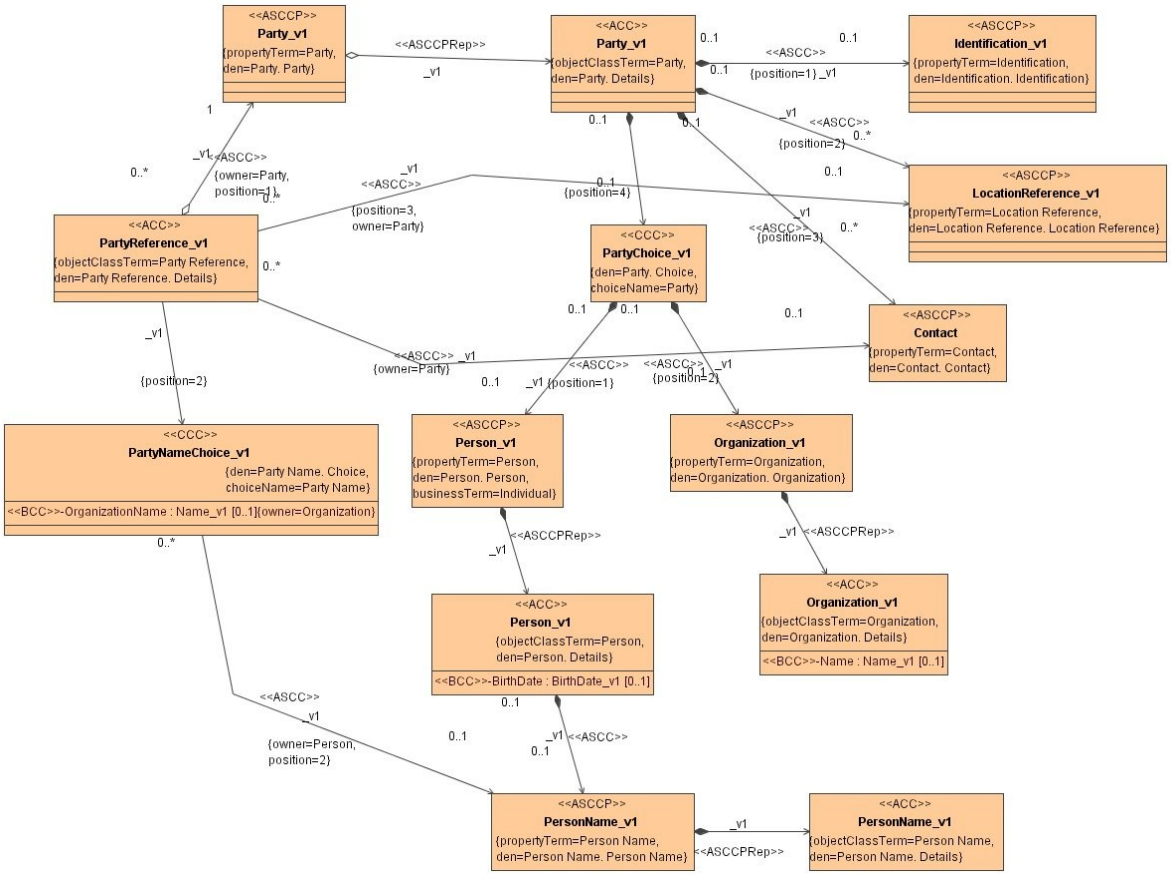

Figure 23: An example illustrating definition of a reference object

Figure 23 illustrates a `PartyReference` ACC. It defines a data structure for referencing information in the `Party` ACC. To create a reference ACC, an object role must first be created, which is the `Party` ASCCP in the diagram. A reference ACC is then created as an indirect association with the ASCCP (note that this is also due to the CCTS meta-model; an ACC cannot have a direct association with another ACC). The association is an aggregation type indicating the reference nature of the relationship (note that if the `Party` ASCCP association type is a composition, then the reference ACC is like a reference to an object instance owned by another ACC). An owner ACC is also indicated on the association. In this case, it is the `Party` ACC itself. Note that the cardinality on the `Party` ASCCP association end is always strictly one, because a reference object must always point to one and only one (master) object instance. This aggregation association also indicates that an underlying implementation, such as in a syntax-specific data structure specification, must have a provision for conveying identity of the target object (the `Party` ACC). In this case, because the “owner” on the association is the `Party` ACC itself, this provision needs to include only the key to the `Party` ACC.

After having the reference/aggregation association linked to the target object, the reference ACC can have regular associations to properties of the target object. The regular association indicates the property that the reference ACC repeats from the target object (the target ACC). In Figure 23, the `PartyReference` ACC has regular associations with the `Contact` and `LocationReference` ASCCPs. Notice that the cardinalities of the two properties in the `Party` ACC are zero-to-many (multiple), while the cardinalities in the `PartyReference` ACC are zero-to-one. This conveys that an instance of the `PartyReference` ACC needs to pick one contact and location information from a referenced instance of the master Party data.

In Figure 24, the ShipToPartyReference ACC reuses the PartyReference ACC through the generalization relationship. The single cardinality of Contact and the LocationReference ASCCPs in the PartyReference ACC reflect the need to know specifically where to ship and which contact information to use in a particular business arrangement. Note that a structure for a ShipFromPartyReference ACC can be created in the same way as the ShipToPartyReference ACC. Note that the structural effect of an aggregation association from an ACC class to an object role constraint kind of ASCCP class (an ASCCP with the aggregation association) is that the ACC gets the identity of the ACC class at the other end of the ASCCP class. Therefore, the ShipToPartyReference ACC association to the ShipToParty ASCCP in Figure 24 structurally only gives the ShipToPartyReference ACC the identity of the Party ACC. Because the ShipToPartyReference ACC already inherits the identity of the Party ACC from the PartyReference ACC, this association serves only as a logical indication of the relationship between the ShipToPartyReference ACC and the ShipToParty ASCCP (a syntax-specific data specification should consider this and not generate the identity data structure of the Party ACC two times within the structure of the ShipToPartyReference ACC).

The Contact and the LocationReference ASCCPs may be used by several ACCs. Therefore, the two regular associations to the two ASCCPs also need to indicate the owner (or owners) of the information they are referencing. The number of owners specified must be sufficient to provide a unique path from the top-level owner (in this case the Party ACC) to the property being referenced. For example, for referencing the PersonName ASCCP, the Person ACC is indicated as the owner. Indicating only the Person ACC is sufficient, because the Person ACC is used only once under the Party ACC.

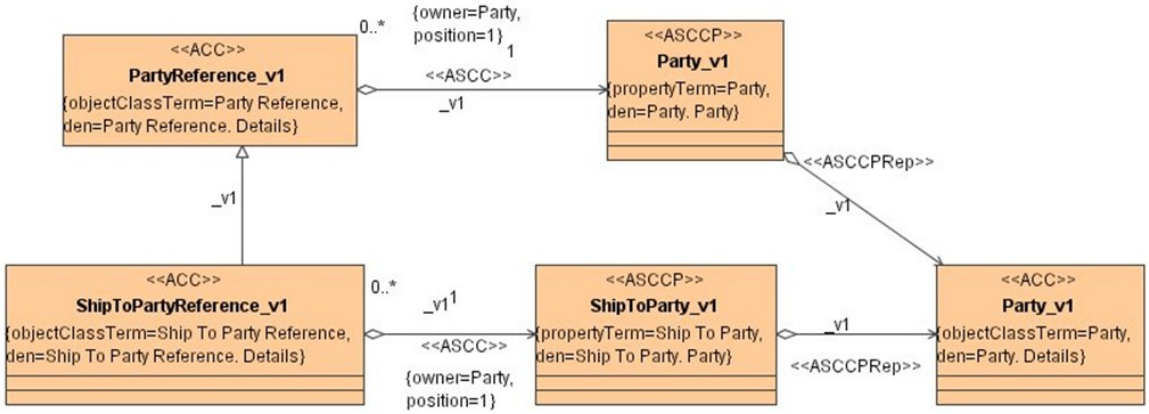

Figure 24: An example of reference object reuse

An ACC may also directly use the object role constraint ASCCP. Figure 25 shows two different effects when different association types are used. In the first case, the Delivery ACC, which has an aggregation association with the ShipToParty ASCCP, will have only the identity of the Party ACC. In the other case, the Shipment ACC, which has a regular association with the ShipToParty ASCCP, will have the full content of the Party ACC. This is useful when a business requirement requires the reference object to have a flexibility to repeat the whole content of the object (the ACC) being referenced.

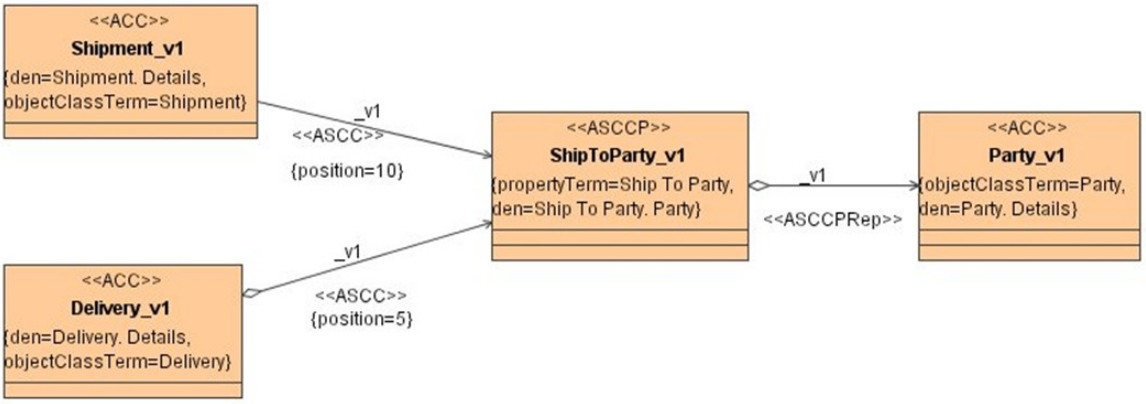

Figure 25: Examples of direct usages of the object role constraint ASCCP

Creating Business Information Entities

This section provides examples of deriving Business Information Entities (BIEs) from CCs. Figure 26 shows an example ABIE that is based on the `Identification. Details` ACC described earlier in Typical Core Component Pattern. The ["`Siebel`"] system context value is specified in the `SiebelContext` object instance. This indicates that the ABIE is applicable for combination of ["`Siebel`"] and any other business contexts. Note that the ABIE restricts the cardinalities of the original BCCs from zero-to-one to single (required). However, it also indicates that the `Alternate. Object Key` and `Revision. Revision` ASCCPs are not supported, while the `Application. Object Key` ASCCP is supported as the `Siebel_ Application. Siebel_ Object Key` ASBIEP.

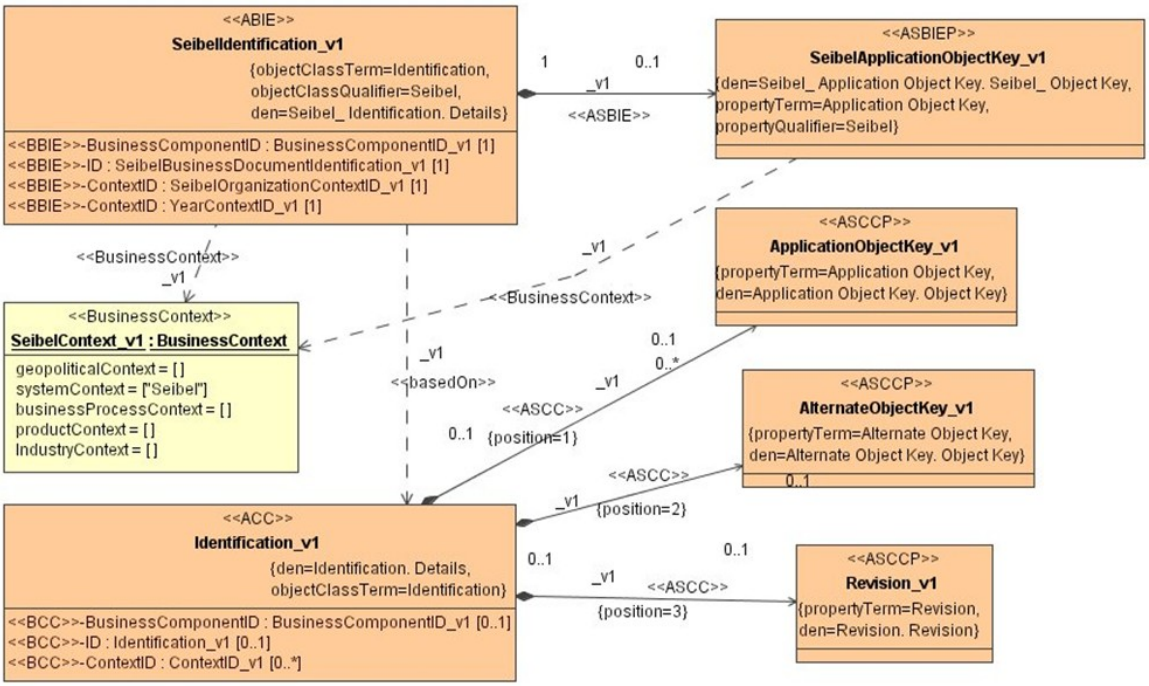

Figure 26: An example ABIE - `Siebel_ Identification. Details`

Figure 27 shows details of BBIE customizations. These are BBIEs that are used by the `Siebel_ Identification. Details` ABIE.

The original type of the BusinessComponentID attribute is the Business Component. Identifier BCCP. The corresponding BBIEP that is used in the ABIE specializes it by specializing its data type. The Fusion Middleware_ Object_ Identifier. Type BDT represents that specialization. The BDT indicates that this is a 64-character long string generated by the Oracle Fusion Middleware as indicated by the fixed schemeID and schemeAgencyID supplementary components. The BBIEP is applicable to the global context in the integration environment (which occurs when a BusinessContext instance has no context values), because this is an infrastructure-generated value.

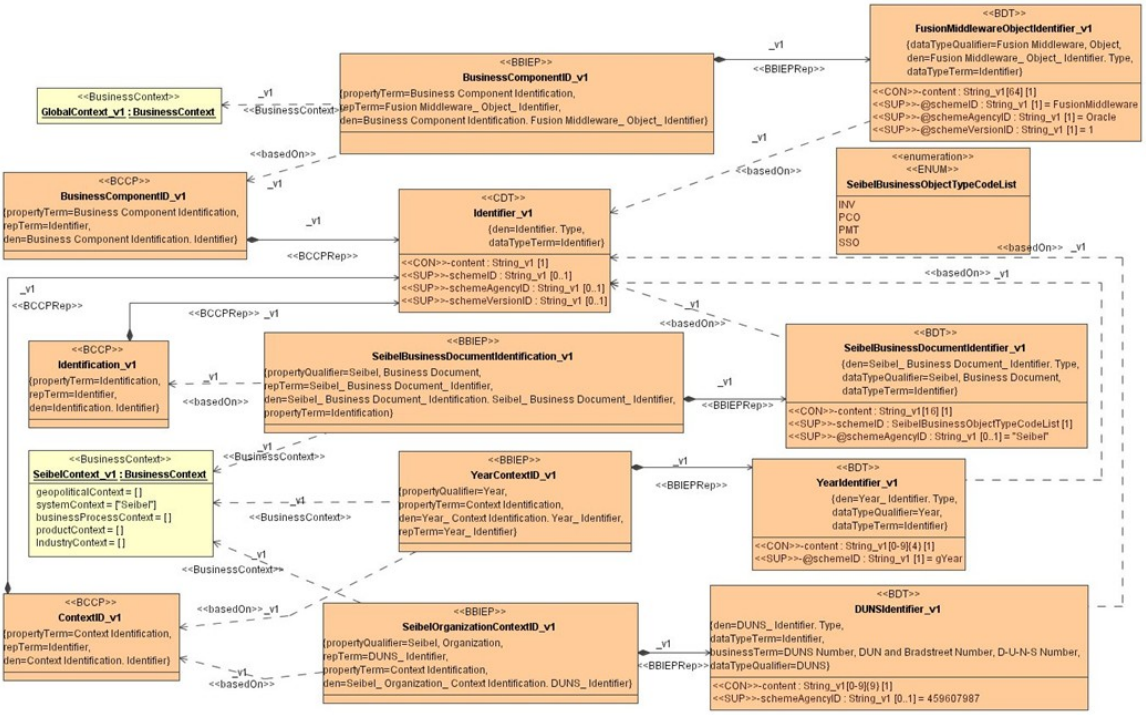

Figure 27: Examples of BBIEs used by the Siebel_ Identification. Details ABIE

The original type of the ID attribute is the Identification. Identifier BCCP. According to the dictionary definition, this is a business document identification, which is used in conjunction with the following sibling ContextID attribute (for which BCCP is Context. Identifier). The combination of (1) Siebel_ BusinessDocument_ Identification. Siebel_ Business Document_ Identifier, (2) Siebel_ Organization_ Context Identification. DUNS_ Identifier, and (3) Year_ Context. Year_ Identifier BBIEPs indicates the representation of business document identification in the Siebel system (that is, business document identification is unique only when it is combined with an issuing organization and the year it is issued). The BDT associated with (1) indicates that it is a 16-character long identifier. The format also depends on the type of document identified by the SiebelBusinessObjectTypeCodeList. The BDT associated with (2) is a corporation's DUN and Bradstreet number (DUNS number), which is a 9-digit long number. Its schemaAgencyID is fixed to the DUNS number of the DUN & Bradstreet Corporation that is the issuer of the identifier. The BDT associated with (3) conveys the 4-digit Gregorian year value and format. The three BBIEPs share the same BusinessContext instance as its parent ABIE, meaning that they apply to all other context values in combination with the "Siebel" context value.

Figure 28 illustrates an ASCCP specialization of the `Application Object. Object Key` ASCCP. The corresponding `Siebel_ Application Object. Siebel_ Object Key` ASBIEP is created by specializing its ACC representation—the `Object Key. Details`. Consequently, the `Siebel_ Object Key. Details` ABIE is defined. The `Object Key. Details` ACC reuses the `Context Identification. Identifier` and `Identification. Identifier` BCCPs (which are also used by the `Identification. Details` ACC). However, in this case they are used as a key to an object instance rather than to a document (the one usually shown to humans). The two are specialized into different BBIEPs to convey the representations used for that purpose. Their cardinalities are also restricted to single, indicating that they are required to be used in combination. The `Identification. Identifier` BCCP is specialized as (1) `Siebel_ Object Identification. Siebel_ Object Identifier` BBIEP. The `Context. Identifier` BCCP is specialized as (2) `Siebel_ Object_ Context Identification. Siebel_ Business Object Type_ Identifier` BBIEP. In (1), its BDT indicates that the `Identifier` is a 16-character long string that is issued by the Siebel system itself (as indicated by the fixed value in the `schemeAgencyID` attribute). In (2), its BDT indicates that the identifier is a code list indicating the type of business object; for example, `INV` for invoice and `PCO` for purchase order (the code list is shown in Figure 27). The `schemaAgencyID` attribute also indicates that this code list is maintained by the Siebel system itself.

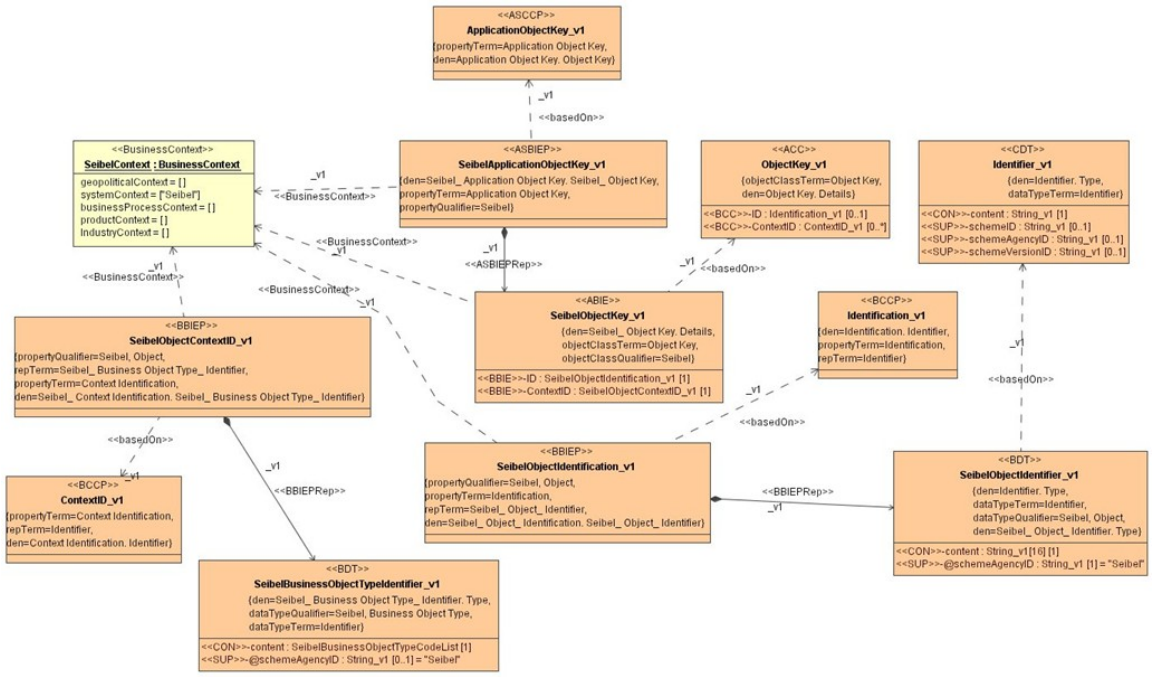

Figure 28: An example ASBIEP - `Siebel_ Application Object Key. Siebel_ Object Key`

Chapter 5: Oracle UML Packaging Structure (Modularity Model)

In this section, Oracle UML packaging structure is described. This structure provides a modularity model of business components. That is, either a package is self-contained and does not rely on classes in other packages or dependencies between any two packages are one way—unidirectional.

UML package is also used for grouping related kinds of components and versioning of model artifacts. Versioning will be described in more detail later in the versioning section (Versioning). Packages and their structures from the top level are described first in the following subsections.

Top-Level Packages

Figure 29 shows the top-level packaging structure. The root package of the Oracle UML CCTS model for EBO is the `OracleEnterpriseObjectLibrary`. This package is subdivided into the `Core` and the `Industry` packages. The `Core` package contains all Oracle-developed objects and components. It includes both families of Core Components (CCs) and Business Information Entities (BIEs). The `Industry` package contains vertical extensions to the core model organized according to domains. As shown in Figure 29, the `Telco` and the `HR` packages are examples of vertical extensions for the telecommunication and human resource industries, respectively.

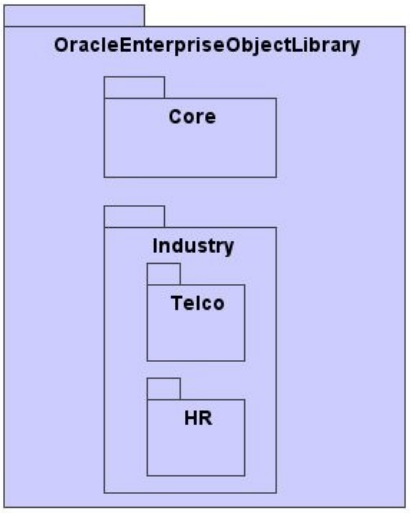

Figure 29: Top-level Oracle Enterprise Object Library packaging structure

Figure 30 shows detailed packaging structures of the `Core` package and a `Telco` industry package. Each of the bottom-level packages in the figure have more children packages including the `AGG` (aggregate), `ASP` (association property), and `BSP` (basic property) for storing ACC/ABIE, ASCCP/ASBIEP, and BCCP/BBIEP, respectively. In addition, ABIEs, which are based on a particular ACC, are kept under that particular individual ACC package as described later in Individual Business Component Packages. Figure 31 shows examples of these packages.

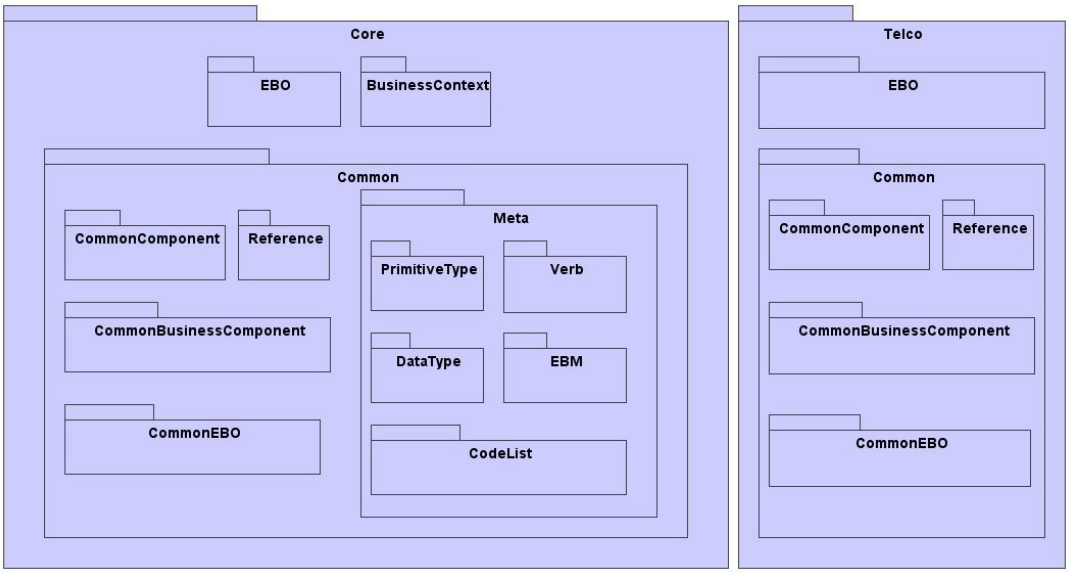

Figure 30: The `Core` and an `Industry` package structure

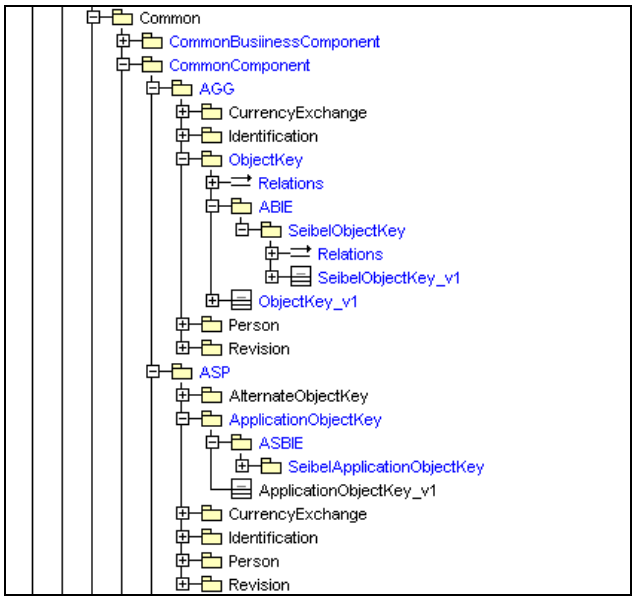

Figure 31: Business component packages

Figure 30 shows that the `Core` and `Industry` packages share a common modularity model with an exception that an industry package does not have a `Meta` package. That is, industries will not likely need to use `Meta`'s kinds of objects outside of those provided by Oracle.

The following subsections describe the kinds of objects and components that go into each package shown in Figure 30 under the `Core` and `industry` packages.

BusinessContext Package

The `Businesscontext` package exclusively stores object instances of the `BusinessContext` class (defined in the profile). Directly under the `Businesscontext` package are individual packages of each `BusinessContext` object instance. The individual package also store related context values. Figure 32 illustrates the portion of the `BusinessContext` package.

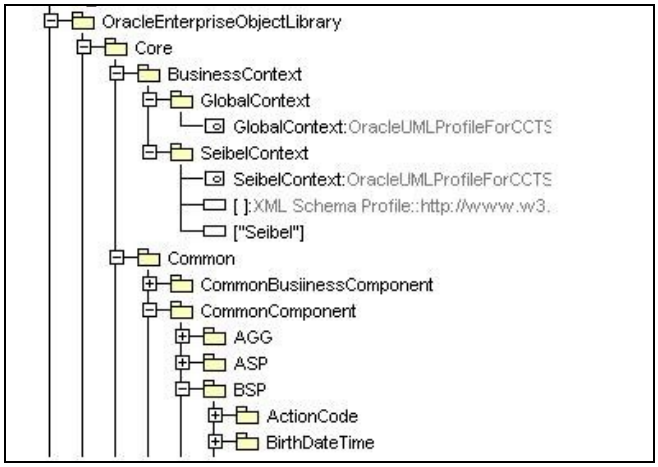

Figure 32: Illustration of the `BusinessContext` Package

EBO Package

The `EBO` package contains the kind of objects called Enterprise Business Objects (EBOs). It also contains components that are specific to a particular EBO; that is, those components are not used outside of that particular EBO. An EBO is an enterprise-level object. The condition for an object to be an EBO is that it can live by itself without any other parent object. That is, an object cannot be an EBO object if it must be created as part of another object. With this characteristic, enterprise-level objects typically have identifiers that function much like a driver license number that is globally unique in particular states or in this case, in particular application contexts. The enterprise-level objects are subdivided into the common ones and the non-common ones. The common ones are defined in the `CommonEBO` package, while the rest are defined in this `EBO` package. Further details about common enterprise-level objects are described in the `CommonEBO` Package section subsequently. Examples of objects in the `EBO` package are `PurchaseOrder`, `SalesOrder`, and so on. Components that are specific to the `SalesOrder` are, for example, `SalesOrderLine` and `SalesOrderCharge`.

Common Package

The `Common` package includes business components that are reused within other components within the `Common` package itself, within enterprise-level objects and components in the `EBO` (and the `CommonEBO`) package, or both.

Meta

The Meta package includes components that are reused across enterprise-level objects and messages. The components in the Meta package include those that are used in the message header and those that are primitive in support of defining other business components within the Common package but outside of the Meta. The Meta package is self-contained; that is, components within the Meta package do not depend on any component outside of the Meta package.

[RMeta] Model elements in the Meta package MUST NOT have any relationship with any other model elements outside of the Meta package.

CommonEBO Package

The CommonEBO package contains enterprise-level objects that are reused within other enterprise-level objects and components that are specific to that enterprise-level object. As described in BusinessContext Package, an enterprise-level object is an object that can live by itself. However, some common enterprise-level objects live by themselves and also are referred to by content as part of another object. Examples of objects in the CommonEBO package are Item, ItemStructure, ShipmentUnit, and so on. Components that are specific to Item are, for example, ItemManufacturer and ItemAttachment.

CommonBusinessComponent Package

The CommonBusinessComponent package contains large-grain reusable components. The distinguishing characteristic of components in this package from enterprise-level objects is that they are uniquely identifiable (that is, they have object identification) only within other business components. An example component in this package is the Distribution component. The component is identifiable only under another object such as a SalesOrder.

CommonComponent Package

As opposed to the CommonBusinessComponent, the CommonComponent package contains reusable components that do not have object identification. In some cases, components in this package may be viewed as logical groupings of related information for convenience of reuse, for indication of closely related information (for example, fields that are used together for most of the time), for data structure simplicity, or a combination of these). Examples of components in this package are Tax, GeographicalCoordinate, PhoneCommunication, and so on. Grouping of information in these components allows for reuse with one reference point instead of referencing each of its fields. For example, a Tax component groups the fields Amount and Percentage together. Consequently, the Tax can be reused as a single point of reference as opposing to having to refer to two separate fields as TaxAmount and TaxPercentage. Because of identification-less nature of a CommonComponent, the cardinality of this component in a direct usage is limited to the maximum of one. For example, having a multiple cardinality associated with the Tax component is impractical, because it has no identity when information reference is necessary.

Reference Package

Components in the `Reference` package are used for referring to particular business objects or components by identity. Examples of components in this package are `InvoiceReference`, `InvoiceLineReference`, `ShipToPartyReference`, and so on.

PrimitiveType Package

The `PrimitiveType` package contains components (classes) that represent primitive types as defined in the Core Components Data Types Catalogue 2.01 as well as any other additions by Oracle.

DataType Package

The `DataType` package contains components that represent the Core Date Type defined in the Core Components Data Types Catalogue 2.01 (the CDT classes) as well as derived business data types (the BDT classes) as defined by Oracle. Under the `DataType` package, individual packages exist for each core data type. Each of the individual core data type packages then has a BDT package, in which packages for each individual BDT reside. This is a similar pattern to the `AGG` and `ABIE` package structure shown in Figure 31.

Verb Package

The `Verb` package includes all transactional verbs (also known as operational verbs) that can be used with the enterprise-level objects (objects in the `EBO` and `CommonEBO` package) to create definitions for business messages.

EBM Package

Oracle business message is called Enterprise Business Message (EBM). An EBM is constructed by pairing a verb and an enterprise-level object plus header information. Header information is used by middleware to route, track, correlate messages, and more. The `EBM` package contains components that capture such header information. These components are used across all Oracle EBM definitions.

CodeList Package

The `CodeList` package is used exclusively to store UML enumerations (those with the <<`ENUM`>> stereotype).

Individual Business Component Packages

Each modularity package described from BusinessContext Package through CodeList Package (referred to as modular packages) has CCTS component type packages as necessary. CCTS component type packages include AGG, ASP, BSP, CCC, and SCC packages. Under each CCTS component type package is where each individual business component package resides. The name of individual business component package must be the same as the class name of the business component it contains. In addition, each enterprise-level business object in the EBO and CommonEBO package also has its own package. Each package contains a package for the enterprise-level object itself and other packages for its components.

Figure 33 shows example packages of individual components, as well as the enterprise-level objects. ObjectKey is an example of an individual component package of an ACC. It also contains packages for ABIEs that are based on the ObjectKey ACC such as the SiebelObjectKey package. Item and SalesOrder are examples of the CommonEBO and EBO packages, respectively. These packages group all business components that are related to each enterprise-level object. As seen in the figure, it has CCTS component type packages (the AGG, ASP, and BSP packages; the CCC and SCC packages may also be there) and then individual business component packages for CCs family of business components. Notice that each EBO package again contains a package with the same name under the AGG package.

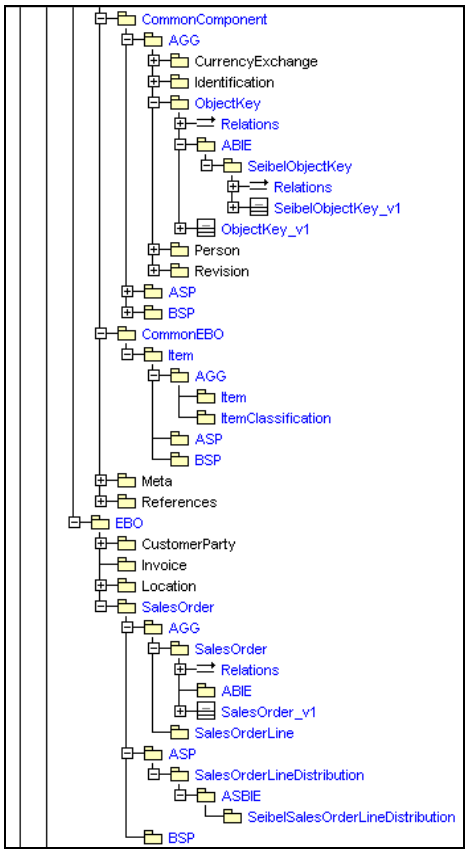

Figure 33: Examples of individual business component packages

Industry Extensions and Their Packages

This section describes the procedure to extend the Oracle core data model (the Enterprise Object Library - EOL) for industry-specific requirements and the packaging structure to maintain extensions. The core data model refers to those business objects and components under the `Core` package.

Industry extension is accomplished using the UML generalization (that is, specialization) relationship. Industry's specific business components may extend/specialize those in the core (the core is short for the core data model). Classes in the industry extension cannot be a generalization of classes in the core.

For example, a telecommunication industry (telco) transaction may require an extension to the core `SalesOrder` enterprise-level object. Specifically, telco needs to add one property to the core's `ItemInstance`. Oracle adopts the localized extension approach in the logical model, which makes the model more concise, and leaves it to the syntax-specific data structure specification generation to create necessary message structure for the extension. Figure 34 illustrates how this works.

In the example, a `Telco:ItemInstance` is defined as a subclass of the `Core:ItemInstance` and adds the `PriorSerialNumber` property as necessary (note that for illustration, classes in the core and in the telco are differentiated by namespace-like prefixes, while in fact they reside in different packages). Notice that making another `Telco:ItemInstance` ASCCP is not necessary, and creating any other extension on the ancestry where no real extension exists is also not necessary. On the other hand, a `Telco:SalesOrder` enterprise-level object is made as an extension to the `Core:SalesOrder` in the EBO package although a property is not directed added there. This extension serves as a logical indicator that a specialized sales order enterprise-level object is being made and that a syntax specific specification generation must take into account extensions related to children of the `Core:SalesOrder`. How this extension appears in a syntax-specific specification is left for its respective naming and design rules (NDR), for example, whether the extension will be a `substitutionGroup` or `xsi:type` in an XML schema syntax-specific specification.

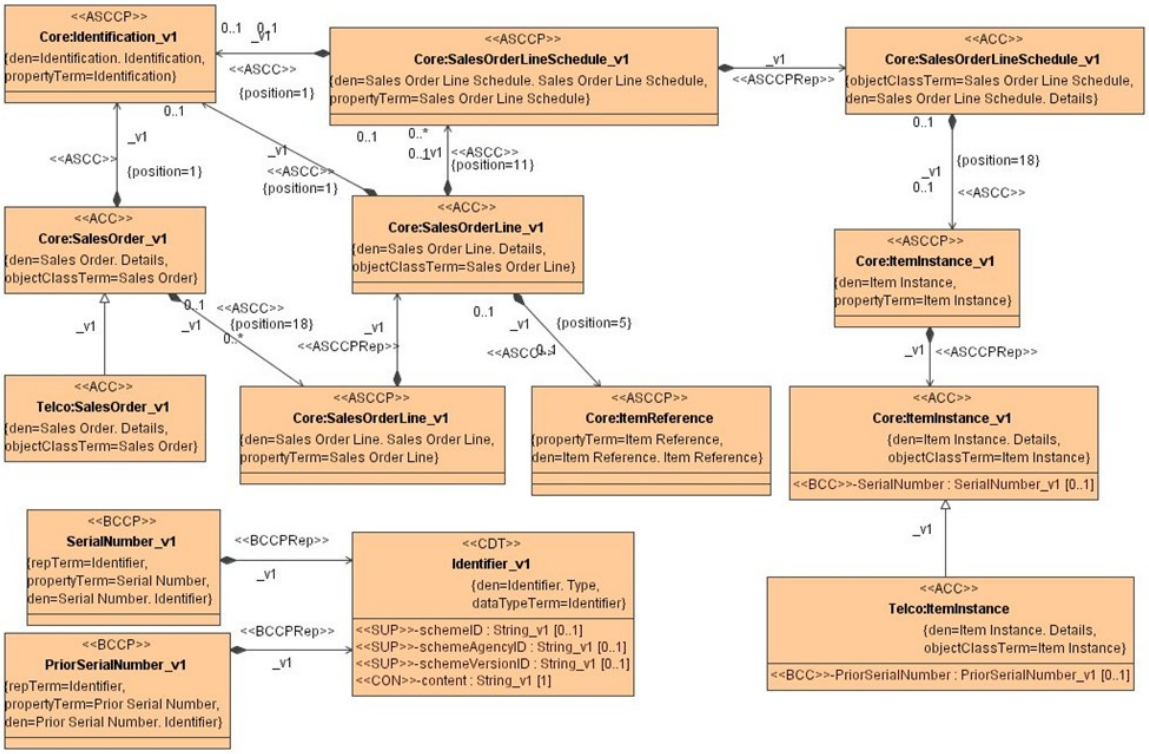

Figure 34: Telco industry extension example

The packages and their structures for keeping industry extension have been outlined in Top-Level Packages and delineated in Figure 29 and Figure 30. The industry package in Figure 30 is repeated here again in Figure 35. Specifically, the figure shows that the package for a particular industry vertical extension such as that of the telecommunication industry, the `Telco` package, contains all of the packages that have been described for the `Core` package except the `Meta` package and its children packages. The definition of each package is the same as that of the `Core` with the exception that they store only industry-specific objects and components. Each industry vertical extension, such as human resource and automotive will contain the same packaging structure. Note that in addition to these packages in Figure 35, the individual object and component packaging structure as described in Individual Business Component Packages must also be followed.

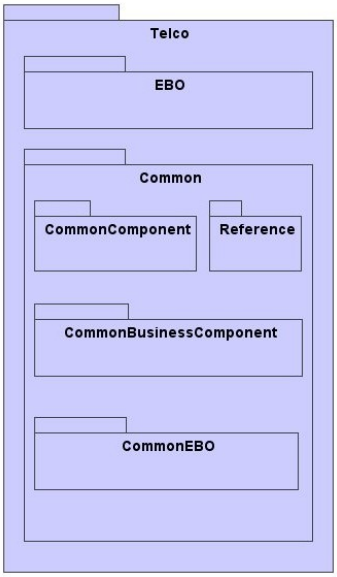

Figure 35: Industry extension packaging structure

Rules and Guidelines

[REXPCK] Extensions to the core data model MUST be put in an industry-specific package within the `Industry` package.

[RIndClass] Classes in the industry extension MUST NOT be generalization of classes in the core. On the contrary, they may extend classes in core.

[RIndvPacName] The name of an individual business component package MUST be the same as the class name of the business component that it contains.

Versioning

This section provides guidance as well as requirements if different versions of individual components need to be maintained within a CCTS UML data model.

Due to the lack of a UML tool with a CVS-like repository to support individualistic versioning requirements, suffixes to the class and relationship names are used as a primary vehicle to keep versions of CCTS model elements. The following conventions can be used as an interim approach to version CCTS model elements.

[RClassVer] The UML class name of all classes MUST be suffixes with `_v<number>`, where the `<number>` indicates the version of the model element.

[RClassLoc] Different versions of a class (classes that have the same "`uniqueIdentifier`" tag) SHALL stay in the same individual business component package until their names change.

For example, a `Contact` version 1 would be named `Contact_v1`. This convention, as well as others subsequently, is chosen because generally UML tools do not allow classes with the same name in the same package. Hence, a meta-attribute is not best for keeping the version number. Using the package to keep each version has proved to be more onerous. Moreover, neither of them allows the version of the model element to be easily seen in the class diagram.

[RAssocVer] All UML associations MUST be named `_v<number>`, where the `<number>` indicates the version of the model element.

[RDepVer] All UML dependencies MUST be named `_v<number>`, where the `<number>` indicates the version of the model element.

[RGenVer] All UML generalizations MUST be named `_v<number>`, where the `<number>` indicates the version of the model element.

[GRelLoc] Although not necessary, Oracle recommends that each UML association, dependency, and generalization link be kept in the same package as the client class (the class from which the association is directed).

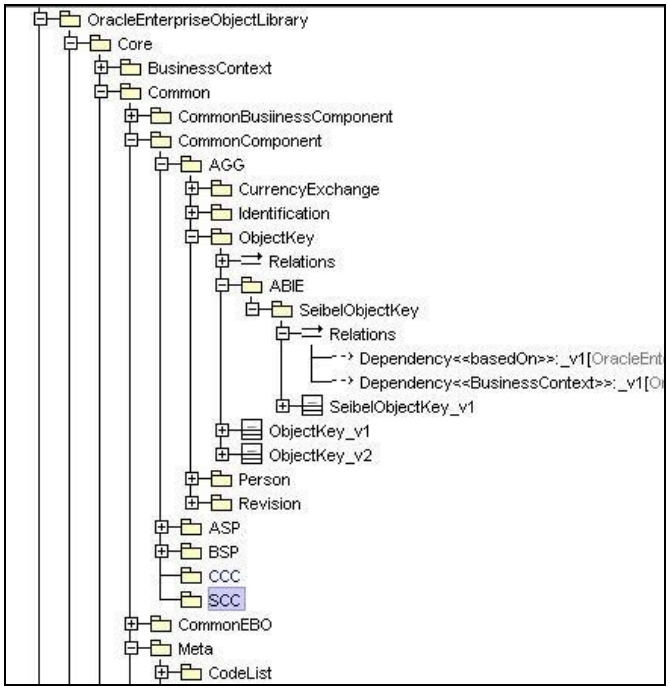

Figure 36: An example model with versioning information

Figure 36 shows a simple example of model with version information along with packages. Notice that each component has its own package including the `CurrencyExchange`, `Identification`, `ObjectKey`, and more. In particular, two versions of the `ObjectKey` ACC present. The figure also illustrates how ABIEs are stored along with their based ACCs using the `SiebelObjectKey` ABIE as an example. Furthermore, the figure illustrates that the two dependency relationships are kept with the client class, the `SiebelObjectKey` ABIE, because it is based on the `ObjectKey` ACC and also depends on a `BusinessContext` object instance. Both dependencies have their name labeled as `_v1` to indicate their version number.

The following rules must be followed when changes are made to the CCTS UML data model where maintaining traces of previous versions is necessary. The version number assignment approach provided next is not a means to directly indicate if a component has changed. Because the version number assignment is localized, a component which has the same version number in different releases may have one of its descendants changed with a new version. In the integration implementation, a component is considered change even if only one of its descendants changes.

Deleting a Class

The new version of class is created and its "`isDeleted`" tag is set to `True` (this means that the class has been marked Deleted). All existing relations to and from this class that are not marked Deleted need a new version that is connected to this new version of the class and their "`isDeleted`" tag set to `True`. This means that these new links are marked Deleted.) That is, the class is not physically deleted but is logically deleted with the "`isDeleted`" tag, and all relations to and from the class are also logically deleted as a consequence.

Note that rules under Changing the Name of a Class should be used to change the name of a class. Changing the name by deleting and adding a class will not result in a good trace, because the old and the new class will not have the same identity.

Making Changes to a Class

Changes in this category include add/remove attributes (BCC or BBIE) of a class, the type associated with the attribute, the cardinality of the attribute, the name of the attribute, tags related to the CCTS data model (tags not related to CCTS data model are those relating to the registry), and any UML documentation. For these changes, a new class must be created to reflect the content of the new version of the class. All previously existing relations (association, generalization, and dependency) to/from this class that are not marked Deleted need to have a new version, and they are connected to the existing versions of the classes at the other ends that do not change. The attribute for which name has been changed must have the original name assigned to the "`previousName`" tag (note that the value of its "`uniqueIdentifier`" tag must remain the same).

Changing to the stereotype of a class shall be considered as deleting and making a new class. That is, a new "`uniqueIdentifier`" is required in this case.

Changing the Name of a Class

Changing the name of a class requires creation of a new class with the new name and a new package with the corresponding name. The version of the new class starts with the version incremented from the previous class before the name change. The new class must have the same "`uniqueIdentifier`" as the previous class. Its "`previousName`" must also be assigned with the name of the previous class. New versions must also be created for all relations that are retained (not marked Deleted) from the previous version and connected to the new class.

Adding a Relation

Adding a relation to a class (the client class) requires a new version of the class. Then, all previously existing relations to/from the class that are not marked Deleted need to have a new version, and they are connected to the existing version of the classes at other ends that do not change. The new relation has version 1 and must stay in the right package as described at the beginning of Versioning. Note that the class, which the new relation is directed toward, remains the same version.

Deleting a Relation

Removing a relation (including association, dependency, or generalization) from a class (the client class) requires a new version of that class. All previously existing relations to/from the class that are not marked Deleted need to have a new version, and they are connected to the existing version of the classes at the other ends that do not change. The new version of the relation intended for deletion must then be marked Deleted.

Changing Attributes of an Association relation

Changes in this category include cardinalities, tags related to the CCTS data model, and UML documentation. These changes need to be applied to a new version of the association. The new version of the association is still connected to the same versions of the client class and supplier class of the previous version of the association.

Changes to the Generalization or Dependency

Change to the generalization or dependency is not possible. They can only be deleted. Deleting one of these relations required creation of a new version and then the new version is marked Deleted. The new version is connected to the same classes as the previous version.

Moving a Class from One Package to Another

Follow the rules under Changing the Name of a Class to move a class from one package to another. This is only about moving from one of the modular packages to another (modular packages are those described in BusinessContext Package through CodeList Package). Moving between one of the CCTS component type packages means changing stereotype, which requires deleting and creating a new class. See Deleting a Class and Changing the Name of a Class.

Making a Release

A release is a collective grouping of all related model elements of particular versions into a single, recognizable, and deliverable unit. When a release of the Oracle Enterprise Object Library is issued, a copy or snapshot of the entire CCTS UML data model is made and stored as a static entity so that references to release packages can be maintained. The following practice can be used to simulate a release snapshot, while a specific versioning tool is not available.

Each model element in the Oracle CCTS UML data model must have its "`Release`" tag added for each release of a particular data structure specification. The tag value must correspond to the release number. While the version control repository for the Oracle CCTS UML data model is not available, Oracle recommends the package structure and UML class diagrams to help manage releases as follows.

A `Diagram` package is created under the root `OracleEnterpriseObjectLibrary` package. This package contains release packages the names of which follow this pattern `Release<release_number>`. Refer to the "Enterprise Business Objects/Messages XML Naming and Design Rules" document for how the release number should be managed. Under each of these release packages is a `Core` package and `Industry` (this follows the practice that each industry package is released based upon a particular release of the `Core`). The `Core` package contains model elements for the Oracle Foundation Pack release indicated in the parent release package, while the `Industry` package contains individual industry packages and releases for individual industry extensions (which are based on the core release). The `Industry` package is optional. It does not have to be included, if no industry extension exists, which is based on the release of the foundation pack. See Figure 37 for illustration.

The `Core` package MUST be populated with packages as described from BusinessContext Package to Individual Business Component Packages as shown in Figure 30, with the exception that no need exists for individual business component packages for the BIE family of CCTS component types including ABIE, ASBIEP, BBIEP, SBIE, CBIE, and BDT. This is because only one version of each component in a particular release must exist. Figure 37 illustrates that each CCTS component type package and individual business component package contains a class diagram for each component.

Each diagram must be named after the corresponding name and version that it is intended to model. This means that in a particular release, a diagram must exist for each UML class intended to be included in that release (all UML classes included must not be marked Deleted).

In a particular release, only one version of a particular UML model element must exist. That is, two model elements must not exist with the same "`uniqueIdentifier`" residing in the single release package. Figure 30 shows that only one version of the `ObjectKey_v2 exists`.

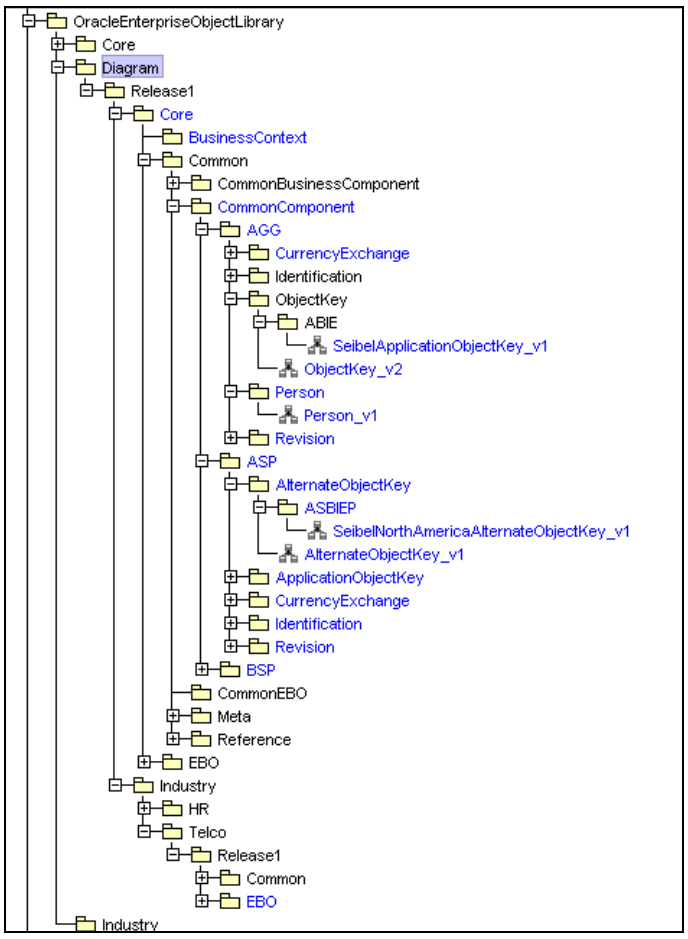

Figure 37: Diagram packaging structure to help manage releases

Under the `Industry` package are packages containing releases for each industry vertical, such as telecommunication and human resources. The package name is the same as the name of the corresponding physical package under the `Core` package. Under each industry vertical package are packages representing industry releases that are based on the parent core release. The package name MUST be of the form `Release<release_number>`. Under the release package are packages that contain extensions for each industry that are the same as those described in Industry Extensions and Their Packages. individual business component packages are not needed for the BIE family of CCTS component types including ABIE, ASBIEP, BBIEP, SBIE, CBIE, and BDT (similar to that of the `Core` release). Conventions about package name and containment must also follow those defined for the `Core` release.

Note that packages under the `Diagram` package are solely for logical viewing/grouping because a class that is physically in one package is commonly present in several packages under the `Diagram` package (note that the physical packages refer to those packages under the `OracleEnterpriseObjectLibrary/Core` package). For example, an `Address` component, which is physically located in the `OracleEnterpriseObjectLibrary/Core/Common/CommonBusinessComponent` package, will be present in several diagrams in several packages under the `Diagram` package where the `Address` component used.

Index

association types 65
  aggregation 65
  composition 66
  regular 65
  summary of 67
Business Information Entities
  creating 85
BusinessContext package 91
CCTS meta-model 11
CodeList package 93
Common package 19, 91
CommonBusinessComponent package 92
CommonComponent package 92
CommonEBO package 92
DataType package 93
datatypes package 56
EBM package 93
EBO package 91
enterprise business objects
  UML profile 11
extensions
  industry 95
individual business component packages 94
industry extensions 95
management package 17
Meta package 92
modeling patterns 69
  choice 79
  object role constraint 82
  reference object 82
  typical core component 69
modularity model 89
notations 4
packages
  BusinessContext 91
  CodeList 93
  Common 91
  CommonBusinessComponent 92
  CommonComponent 92
  CommonEBO 92
  DataType 93
  EBM 93
  EBO 91
  individual business component 94
  Meta 92
  PrimitiveType 93
  Reference 93
  top-level 89
  Verb 93
partitioning 16
prerequisites 5
PrimitiveType package 93
Reference package 93
terminology 7
UML data model
  general usage rules 64
UML generalization 63
UML packaging structure 89
UML profile
  enterprise business objects 11

Verb package....93
versioning....97
adding a relation....99
changing a dependency....100
changing a generalization....100
changing attributes of an association relation....100
changing the name of a class....99
deleting a class....99
deleting a relation....100
making a release....100
making changes to a class....99
moving a class....100