Product Data Management Enablers Specification

Version 1.3
November 2000

PATENT

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

NOTICE

The information contained in this document is subject to change without notice. The material in this document details an Object Management Group specification in accordance with the license and notices set forth on this page. This document does not represent a commitment to implement any portion of this specification in any company's products.

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR PARTICULAR PURPOSE OR USE. In no event shall The Object Management Group or any of the companies listed above be liable for errors contained herein or for indirect, incidental, special, consequential, reliance or cover damages, including loss of profits, revenue, data or use, incurred by any user or any third party. The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

ISSUE REPORTING

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the issue reporting form at *http://www.omg.org/library/issuerpt.htm*.

Contents

Preface 1

About the Object Management Group 1

What is CORBA? 1

Associated OMG Documents 2

Acknowledgments 3

1. PDM Overview 1-1

1.1 Introduction 1-1

1.1.1 Scope — PDM Enablers 1-1

1.1.2 Specification Overview 1-2

1.2 Compliance 1-3

1.2.1 Summary of Optional vs. Mandatory Interfaces 1-3

1.2.2 Compliance Points 1-4

2. PdmResponsibility Module 2-1

2.1 Overview 2-1

2.2 PdmResponsibility Model 2-2

2.3 PdmResponsibility Description 2-3

2.3.1 Exceptions 2-3

2.3.2 Actor 2-5

2.3.3 Program 2-5

2.3.4 Party 2-5

2.3.5 Person 2-6

2.3.6 Organization 2-6

2.3.7 PersonOrganization 2-6

2.3.8 ProgramOwner 2-7

Contents

2.4 PdmResponsibility IDL 2-8

3. PdmFoundation Module 3-1

3.1 Overview 3-1

3.2 PdmFoundation Model 3-2

3.3 PdmFoundation Description 3-3

3.3.1 Exceptions 3-3

3.3.2 Measurement 3-6

3.3.3 IdentifierSeq 3-7

3.3.4 Identifiable 3-7

3.3.5 IdentificationContext 3-8

3.3.6 Factory Operations 3-10

3.3.7 Additional Identification Description 3-10

3.3.8 Lockable 3-13

3.3.9 Manageable 3-14

3.3.10 SecurityClassifiable 3-14

3.3.11 SecurityClassification 3-14

3.3.12 SecurityClassification Attributes 3-15

3.3.13 Factory Operations 3-15

3.3.14 Stateable 3-15

3.3.15 Identification 3-16

3.3.16 LockOwner 3-17

3.3.17 ObjectCreator 3-18

3.3.18 ObjectOwner 3-18

3.3.19 ObjectSecurityClassification 3-19

3.4 PdmFoundation IDL 3-20

4. PdmFramework Module 4-1

4.1 Overview 4-1

4.2 PdmFramework Model 4-2

4.2.1 PdmFramework Entity Model 4-2

4.2.2 PdmFramework Relationship Model 4-3

4.3 PdmFramework Description 4-4

4.3.1 PdmSystem 4-4

4.3.2 Attributable 4-4

4.3.3 Baselineable 4-6

4.3.4 Qualifiable 4-6

4.3.5 Changeable 4-6

4.3.6 Documentable 4-6

4.3.7 ManagedEntity 4-6

4.3.8 ItemMaster 4-6

4.3.9 ItemRevision 4-7
4.3.10 ItemIteration 4-7
4.3.11 PdmContainmentRelationship 4-8
4.3.12 PdmReferenceRelationship 4-9
4.3.13 PdmTypedRelationship 4-9
4.3.14 MasterRelationship 4-10
4.3.15 RevisionRelationship 4-11
4.3.16 IterationRelationship 4-11
4.3.17 Derive 4-12
4.3.18 Dependency 4-13
4.3.19 RevisionMasterRelationship 4-14
4.3.20 Supersedes 4-14
4.4 PdmFramework IDL 4-15

5. PdmBaseline Module 5-1

5.1 Overview 5-1
5.2 PdmBaseline Model 5-2
5.3 PdmBaseline Description 5-2
5.3.1 Baselineable 5-2
5.3.2 BaselineMaster 5-3
5.3.3 BaselineRevision 5-3
5.3.4 BaselineIteration 5-4
5.3.5 BaselineMasterComposition 5-4
5.3.6 BaselineRevisionComposition 5-4
5.3.7 Baselined 5-5
5.4 PdmBaseline IDL 5-6

6. PdmViews Module 6-1

6.1 Overview 6-1
6.2 PdmViews Model 6-2
6.3 PdmViews Description 6-2
6.3.1 Qualification 6-2
6.3.2 Qualifiable 6-3
6.3.3 Qualifies Relationship 6-3
6.3.4 PdmContext 6-3
6.3.5 PdmTraversalCriteriaFactory 6-4
6.3.6 ViewQualification 6-5
6.3.7 ViewContext 6-5
6.3.8 LifeCycleQualification 6-5
6.3.9 LifeCycleContext 6-6
6.3.10 LocationQualification 6-6

Contents

6.3.11 LocationContext 6-6
6.3.12 DisciplineQualification. 6-6
6.3.13 DisciplineContext. 6-7
6.3.14 CompoundQualification 6-7
6.3.15 CompoundContext 6-7
6.3.16 StateContext 6-8
6.3.17 Using Qualifications and PdmContexts 6-8
6.4 PdmViews IDL. 6-9

7. PdmDocument Management Module. 7-1

7.1 Overview 7-1
7.2 PdmDocumentManagement Model 7-2
7.3 PdmDocumentManagement Description 7-3
7.3.1 DocumentMaster 7-3
7.3.2 DocumentRevision 7-3
7.3.3 DocumentMasterComposition Relationship . . . 7-4
7.3.4 DocumentIteration 7-4
7.3.5 DocumentRevisionComposition Relationship . 7-5
7.3.6 Documentable. 7-5
7.3.7 Documentation Relationship 7-6
7.3.8 File 7-6
7.3.9 DocumentFileRelationship Relationship 7-7
7.3.10 UnsecuredFile. 7-8
7.3.11 SecuredFile. 7-8
7.3.12 Vaults 7-11
7.3.13 FileStorage Relationship 7-12
7.3.14 Data Transfer 7-12
7.4 Document Management IDL 7-14

8. PdmProductStructureDefinition Module 8-1

8.1 Overview 8-1
8.2 PdmProductStructureDefinition Model 8-3
8.3 PdmProductStructureDefinition Description 8-4
8.3.1 Alternate. 8-4
8.3.2 AssemblyComponentUsage 8-5
8.3.3 DesignSupplierRelationship. 8-6
8.3.4 PartDocumentRelationship. 8-6
8.3.5 NextAssemblyUsageOccurrence 8-7
8.3.6 PartData 8-8
8.3.7 PartDataIteration 8-8

8.3.8 PartDataIterationRelationship 8-8
8.3.9 PartDataRelationship 8-9
8.3.10 PartMaster 8-9
8.3.11 PartMasterComposition 8-10
8.3.12 PartRevision 8-11
8.3.13 PartStructure 8-11
8.3.14 PartStructureIteration 8-11
8.3.15 PartStructureIterationRelationship 8-12
8.3.16 PartStructureRelationship 8-12
8.3.17 PartSupplierRelationship 8-13
8.3.18 PromissoryUsageOccurrence 8-13
8.3.19 Substitute 8-14
8.3.20 Usage 8-15
8.3.21 Make From Usage 8-16
8.4 PdmProductStructureDefinition IDL 8-17

9. PdmEffectivity Module 9-1

9.1 Overview 9-1
9.2 PdmEffectivity Model 9-2
9.3 PdmEffectivity Description 9-2
9.3.1 ConfigurationItem 9-2
9.3.2 ConfigurationDesign Relationship 9-3
9.3.3 Effectivity 9-3
9.3.4 EffectivityItem Relationship 9-4
9.3.5 DatedEffectivity 9-4
9.3.6 LotEffectivity 9-5
9.3.7 SerialNumberedEffectivity 9-6
9.3.8 EffectivityContext 9-6
9.3.9 DatedContext 9-6
9.3.10 SerialNumberedContext 9-7
9.4 PdmEffectivity IDL 9-8

10. PdmChangeManagement Module 10-1

10.1 Overview 10-1
10.2 PdmChangeManagement Model 10-3
10.3 PdmChangeManagement Description 10-3
10.3.1 EngChangeItem 10-3
10.3.2 EcoDeliverable 10-4
10.3.3 EngChangeIssue 10-5
10.3.4 EngChangeNotice 10-5
10.3.5 EngChangeOrder 10-5

10.3.6 EngChangeRequest. 10-6
10.3.7 Addressing 10-7
10.3.8 ChangeDescription 10-7
10.3.9 Deliverable 10-8
10.3.10 EciInitiation 10-8
10.3.11 EcnCoRequirement. 10-9
10.3.12 EcnPreRequirement 10-10
10.3.13 EcoCoRequirement. 10-10
10.3.14 EcoPreRequirement 10-11
10.3.15 EcrInitiation 10-11
10.3.16 EngChangeAffectedData 10-12
10.3.17 EngChangeAffectedParty 10-13
10.3.18 ObjectChange 10-14
10.3.19 ObjectChangeNotification 10-14
10.3.20 Responsibility. 10-15
10.4 PdmChangeManagement IDL 10-15

11. PdmManufacturingImplementation Module 11-1

11.1 Overview 11-1
11.2 PdmManufacturingImplementation Model. 11-2
11.3 PdmManufacturingImplementation Description 11-2
11.3.1 ProcessMaster. 11-2
11.3.2 ProcessMasterControlledByOrganizationRelationship 11-3
11.3.3 ProcessMasterComposition Relationship 11-3
11.3.4 ProcessMasterUsedAtOrganization Relationship 11-4
11.3.5 ProcessRevision 11-5
11.3.6 ProcessRevisionComposition Relationship . . . 11-5
11.3.7 ProcessRevisionDocumentation Relationship. . 11-5
11.3.8 PartProducedByProcessRevision Relationship . 11-6
11.3.9 ProcessOperation 11-7
11.3.10 ProcessOperationDocumentation Relationship. 11-7
11.3.11 ProcessOperationSequence Relationship 11-8
11.3.12 ProcessStep. 11-9
11.3.13 ProcessStepUsesTool 11-9
11.3.14 ProcessStepConsumesPart 11-10
11.3.15 ControlledProcessStep 11-11
11.3.16 ImportedProcessStep 11-11
11.3.17 ImportedProcessStepUsedAtOrganizationRelationship 11-11

11.3.18 ManufacturingOrganization 11-12
11.4 PdmManufacturingImplementation IDL 11-12

12. PdmConfigurationManagement Module 12-1
12.1 Overview 12-1
12.2 PdmConfigurationManagement Model 12-3
12.3 PdmConfigurationManagement Description 12-5
12.3.1 PartDiscipline 12-5
12.3.2 ProductClass 12-5
12.3.3 ProductClassHierarchy 12-6
12.3.4 ProductComponent 12-6
12.3.5 ProductClassToComponent 12-7
12.3.6 ComponentHierarchy 12-8
12.3.7 ProductFunction 12-9
12.3.8 FunctionHierarchy 12-10
12.3.9 ComponentFunction 12-10
12.3.10 ProductClassFunction 12-11
12.3.11 ItemSolution 12-12
12.3.12 ComponentSolution 12-13
12.3.13 SolutionPartMaster 12-14
12.3.14 SpecificationCategory 12-15
12.3.15 SpecificationExpression 12-16
12.3.16 SpecificationInclusion 12-17
12.3.17 ProductClassCategory 12-18
12.3.18 ProductClassExpression 12-19
12.3.19 ProductClassInclusion 12-20
12.3.20 ProductClassSpecification 12-21
12.3.21 InclusionIfCondition 12-22
12.3.22 IncludedSpecification 12-22
12.3.23 SpecificationCategoryComposition 12-23
12.3.24 SpecificationExpressionOperands 12-24
12.3.25 Configuration 12-25
12.3.26 ConfiguredItemUsage 12-25
12.3.27 ConfiguredSpecificationSolution 12-26
12.3.28 ProductComponentSatisfiesSpecification 12-27
12.4 PdmConfigurationManagement IDL 12-28

13. PdmStepModule 13-1
13.1 Overview 13-1
13.2 PdmStep Model 13-2

Contents

13.3 PdmStep Description 13-2
13.3.1 StepTranslator. 13-2
13.3.2 StepContext 13-4
13.3.3 StepQualification 13-4
13.4 PdmStep IDL 13-5

Appendix A - Mapping the Product Development Process to the PDM Enablers A-1

Appendix B - OMG IDL B-1

Appendix C - References. C-1

Appendix D - Relationship to Other Services D-1

Preface

About the Object Management Group

The Object Management Group, Inc. (OMG) is an international organization supported by over 800 members, including information system vendors, software developers and users. Founded in 1989, the OMG promotes the theory and practice of object-oriented technology in software development. The organization's charter includes the establishment of industry guidelines and object management specifications to provide a common framework for application development. Primary goals are the reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments. Conformance to these specifications will make it possible to develop a heterogeneous applications environment across all major hardware platforms and operating systems.

OMG's objectives are to foster the growth of object technology and influence its direction by establishing the Object Management Architecture (OMA). The OMA provides the conceptual infrastructure upon which all OMG specifications are based.

What is CORBA?

The Common Object Request Broker Architecture (CORBA), is the Object Management Group's answer to the need for interoperability among the rapidly proliferating number of hardware and software products available today. Simply stated, CORBA allows applications to communicate with one another no matter where they are located or who has designed them. CORBA 1.1 was introduced in 1991 by Object Management Group (OMG) and defined the Interface Definition Language (IDL) and the Application Programming Interfaces (API) that enable client/server object interaction within a specific implementation of an Object Request Broker (ORB). CORBA 2.0, adopted in December of 1994, defines true interoperability by specifying how ORBs from different vendors can interoperate.

Associated OMG Documents

The CORBA documentation is organized as follows:

- *Object Management Architecture Guide* defines the OMG's technical objectives and terminology and describes the conceptual models upon which OMG standards are based. It defines the umbrella architecture for the OMG standards. It also provides information about the policies and procedures of OMG, such as how standards are proposed, evaluated, and accepted.
- *CORBA: Common Object Request Broker Architecture and Specification* contains the architecture and specifications for the Object Request Broker.
- *CORBA Languages*, a collection of language mapping specifications. See the individual languagee mapping specifications.
- *CORBA Services: Common Object Services Specification* contains specifications for OMG's Object Services.
- *CORBA Facilities: Common Facilities Specification* includes OMG's Common Facility specifications.
- *CORBA Manufacturing*: Contains specifications that relate to the manufacturing industry. This group of specifications defines standardized object-oriented interfaces between related services and functions.
- *CORBA Med*: Comprised of specifications that relate to the healthcare industry and represents vendors, healthcare providers, payers, and end users.
- *CORBA Finance*: Targets a vitally important vertical market: financial services and accounting. These important application areas are present in virtually all organizations: including all forms of monetary transactions, payroll, billing, and so forth.
- *CORBA Telecoms*: Comprised of specifications that relate to the OMG-compliant interfaces for telecommunication systems.

The OMG collects information for each specification by issuing Requests for Information, Requests for Proposals, and Requests for Comment and, with its membership, evaluating the responses. Specifications are adopted as standards only when representatives of the OMG membership accept them as such by vote. (The policies and procedures of the OMG are described in detail in the *Object Management Architecture Guide*.)

OMG formal documents are available from our web site in PostScript and PDF format. To obtain print-on-demand books in the documentation set or other OMG publications, contact the Object Management Group, Inc. at:

OMG Headquarters
250 First Avenue, Suite 201
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
pubs@omg.org
http://www.omg.org

Acknowledgments

The following companies submitted and/or supported parts of this specification:

- Digital Equipment Corporation
- FUJITSU LIMITED
- International Business Machines Corporation
- Matrix One, Inc.
- Metaphase Technology Division, Structural Dynamics Research Corp.
- Sherpa Corporation
- National Centers of Manufacturing Sciences (NCMS) through the Rapid Response Manufacturing (RRM) Consortium represented by The MacNeal-Schwendler Corporation
- National Industrial Information Infrastructure Protocols (NIIIP) Consortium represented by General Dynamics Electric Boat

PDM Overview 1

Contents

This chapter contains the following sections.

Section Title	Page
“Introduction”	1-1
“Compliance”	1-3

1.1 Introduction

1.1.1 Scope — PDM Enablers

A Product Data Management system (PDM) is a software tool that manages engineering information, supports management of product configurations, and supports management of the product engineering process. The engineering information includes both database objects and “document” objects – sets of information stored in files that are opaque to the PDM system. This information may be associated with specific products or specific product designs, or more generally with product families, production processes or the engineering process itself. The engineering process support usually includes workflow management and concepts of engineering change and notification. In many manufacturing organizations, the PDM is the central engineering information repository for product development activities.

This specification[1] is intended to provide standard interfaces to Product Data Management systems, or other systems providing similar services, from other manufacturing software systems, primarily those supporting various aspects of product and process engineering, and those supporting manufacturing planning.

This specification is organized as twelve IDL modules:

1. PdmResponsibility
2. PdmFoundation
3. PdmFramework
4. PdmBaseline
5. PdmViews
6. PdmDocumentManagement
7. PdmProductStructureDefinition
8. PdmEffectivity
9. PdmChangeManagement
10. PdmManufacturingImplementation
11. PdmConfigurationManagement
12. PdmSTEP

The first three modules – **PdmResponsibility**, **PdmFoundation**, and **PdmFramework** – provide a “replaceable” foundation for the conceptual PDM services that maps basic concepts to available OMG technologies. It is expected that these foundation services may be modified as the common facilities and services mature. The other nine modules define somewhat separable groups of PDM services. The modularity allows diverse software systems to comply, according to the PDM services they are able to offer.

1.1.2 Specification Overview

1.1.2.1 Characteristics

The primary PDM constructs are first-class CORBA objects, which have their own object references for each object instance. The specification uses the following approved CORBA Specifications: Life Cycle Service, Object Property Service, Relationship Service, Time Service, and Currency Service. The specification is compatible with, but does not depend on, other CORBA Services and Facilities, including: Security, Persistent Object, Query, and Transactions.

---

1. PDM Enablers Specification V1.2 is based on the revised, approved submission, (http://www.omg.org/cgi-bin/doc?)mfg/1998-01-01, as modified by the Errata mfg/1998-02-01, the PDM Enablers V1.1 RTF Report, dtc/1999-02-01, and the PDM Enablers V1.2 RTF Report, dtc/1999-10-02. The original Request for Proposal can be found in mfg/1996-08-01.

The specification avoids interfaces that provide services or facilities that meet common needs of all application frameworks and that are not specific to the PDM domain, such as Presentation, Session Management, Rules.

The specification includes frameworks and interfaces for interoperability with other systems and clients that use or provide managed product information. It does not necessarily provide a complete set of interfaces for the total set of services that are necessary for implementing all the user functionality of a fully functional PDM system. It avoids interfaces for administering information that is not managed product information and is known by most PDM systems, such as hosts, rules, valid values lists. These capabilities are to be provided by each PDM system in an implementation-dependent fashion.

1.1.2.2 Structure

Elementary behavior has been organized into separate coherent interfaces, generally named with the -able suffix, such as Identifiable and Manageable. These interfaces do not represent whole abstract or instantiable persistent entities, but rather define separable interfaces and behavior that may be supported by entities. Some of these interfaces may be established in the Common Business Objects effort, while others are PDM-specific behaviors. Many of these behavioral interfaces are defined in the **PdmFoundation** and **PdmFramework** modules.

A group of entities related to persons, organizations, and parties has been separated into the **PdmResponsibility** module. It is expected that most of this module will be replaced by future standard Common Business Objects definitions.

The **PdmFramework** module brings together appropriate behavioral interfaces to define a set of useful abstract whole entities and relationships that are commonly used in PDM related applications.

The other modules extend the **PdmFramework** classes to provide specific concrete instantiable entities and relationships with functionality supporting product data management.

1.2 Compliance

1.2.1 Summary of Optional vs. Mandatory Interfaces

All interfaces defined in each module are mandatory, unless indicated as optional in this section.

1.2.1.1 Modules

A PDM system implementation of these PDM enabler modules is not required to provide all modules. In a sense, each module as a whole is optional. However, some modules depend on other modules.

1.2.1.2 Qualifications

The Qualification interface defined in the PdmViews module is mandatory. However, the inheritance of the Qualifiable interface by each of the relationships and interfaces it affects is optional. If a Qualifiable inheritance is implemented, the implementation is not required to implement all subtypes of Qualification.

If they are implemented, they must be implemented as described.

For example, a PartMaster implementation might support DisciplineQualification but not support EffectivityQualification.

1.2.1.3 Transactions

In this specification all persistent objects optionally inherit the **CosTransactions::TransactionalObject** interface.

1.2.1.4 Documentations

The Documentation relationship, defined in the PdmDocumentManagement module, between a Documentable object and a DocumentMaster object is optional. If a Documentation relationship is implemented, the implementation is not required to implement this relationship for all ManagedEntity objects. For example, a PartMaster object implementation might support Documentation relationship but a File object may not support this relatioship.

1.2.1.5 PdmSTEP

The **StepTranslator::export_baseline** operation is optional.

1.2.1.6 CORBAservices

This specification specifies interfaces inherited from CORBAservices specifications. Compliant PDM enablers implementations shall provide all operations and attributes inherited by PDM enabler interfaces.

It is not required that a PDM Enablers implementation provide the services of CORBAservices interfaces which are not inherited by PDM Enablers interfaces.

1.2.2 Compliance Points

1.2.2.1 IDL Specifications

Compliance to this specification is to be judged against the IDL definitions of the interfaces and their attributes and operations.

The UML object model diagrams are not to be considered as part of the specification for the purposes of judging compliance. They are provided to illustrate and describe the behavior of the IDL interfaces.

1.2.2.2 Minimal Implementation and Extensions

Each module describes a minimal interface definition and a framework. A compliant implementation is expected to provide at least the specified mandatory interfaces, attributes, and operations, but may provide more. However, a compliant implementation must not require that clients of PDM services use extensions to this standard specification in order to use the PDM service in an error-free manner.

1.2.2.3 Module by Module Compliance

It is not required that an implementation of a PDM system provide all the modules specified in this specification. Compliance is to be judged on a module-by-module basis.

1.2.2.4 Module Substitutability

The intent of this specification is that it enables PDM services to be substituted as a whole. A client written to interoperate with one PDM Enablers implementation of a module should also interoperate with other implementations of that module.

This specification does not require module by module substitutability.

That is, it is not required that a module supplied by one PDM implementation interoperate with a different module supplied by another PDM implementation. For example, it is not required that a PdmEffectivity module supplied by vendor A interoperate as an extension to the PdmProductStructure module supplied by vendor B.

1.2.3 Module Interdependencies

The following diagram indicates dependencies among PDM Enabler Modules. For clarity, commonly referenced basic modules (**PdmResponsibility**, **PdmFoundation**, and **PdmFramework**) are shown grouped as are the balance of the domain modules. A single reference is shown between the two groups to remove the many relations that each domain module has on the basic classes.

In the domain classes, a solid arrow indicates a strong dependency on the class pointed to. The strong dependency indicates that the module cannot operate meaningfully without the class pointed to. A dashed arrow indicates a weak dependency meaning that the modules have relationships between them, but can operate meaningfully without the class pointed to.

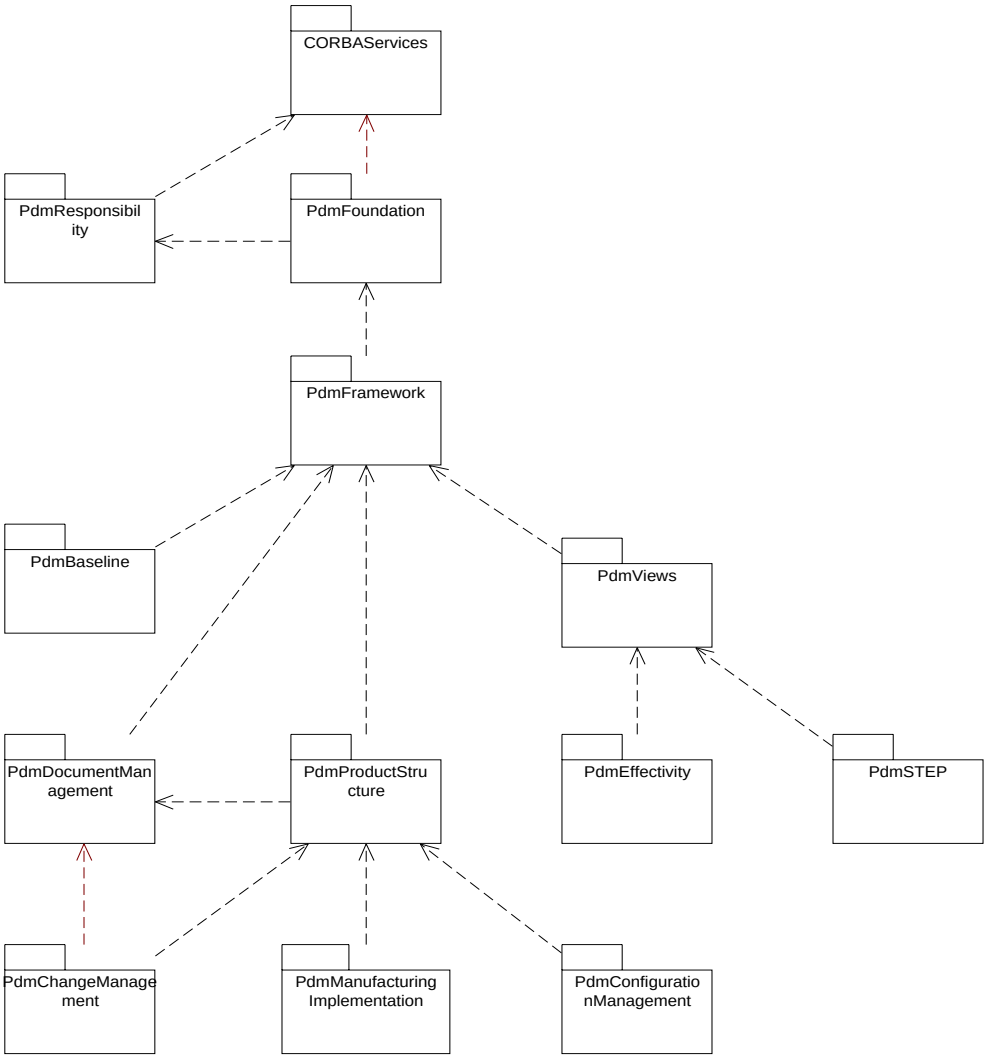

Figure 1-1 Module Interdependencies Diagram

PdmResponsibility Module 2

Contents

This chapter contains the following sections.

Section Title	Page
“Overview”	2-1
“PdmResponsibility Model”	2-2
“PdmResponsibility Description”	2-3
“PdmResponsibility IDL”	2-8

2.1 Overview

The following interfaces represent behaviors that have been identified as required by multiple PDM enablers and which we expect to be required by other interfaces, external to the PDM enablers. These classes represent fundamental elementary units of behavior. We expect several of these classes to become part of Common Business Objects or other domains, but until such time it is necessary to define the behavior that we require of the objects today.

This module defines the minimal interfaces required by the PDM enablers for references to people and organizations as owners, contact, or some other responsibility. It does not attempt to address the entire requirement for identifying and manage people and organizations.

2.2 PdmResponsibility Model

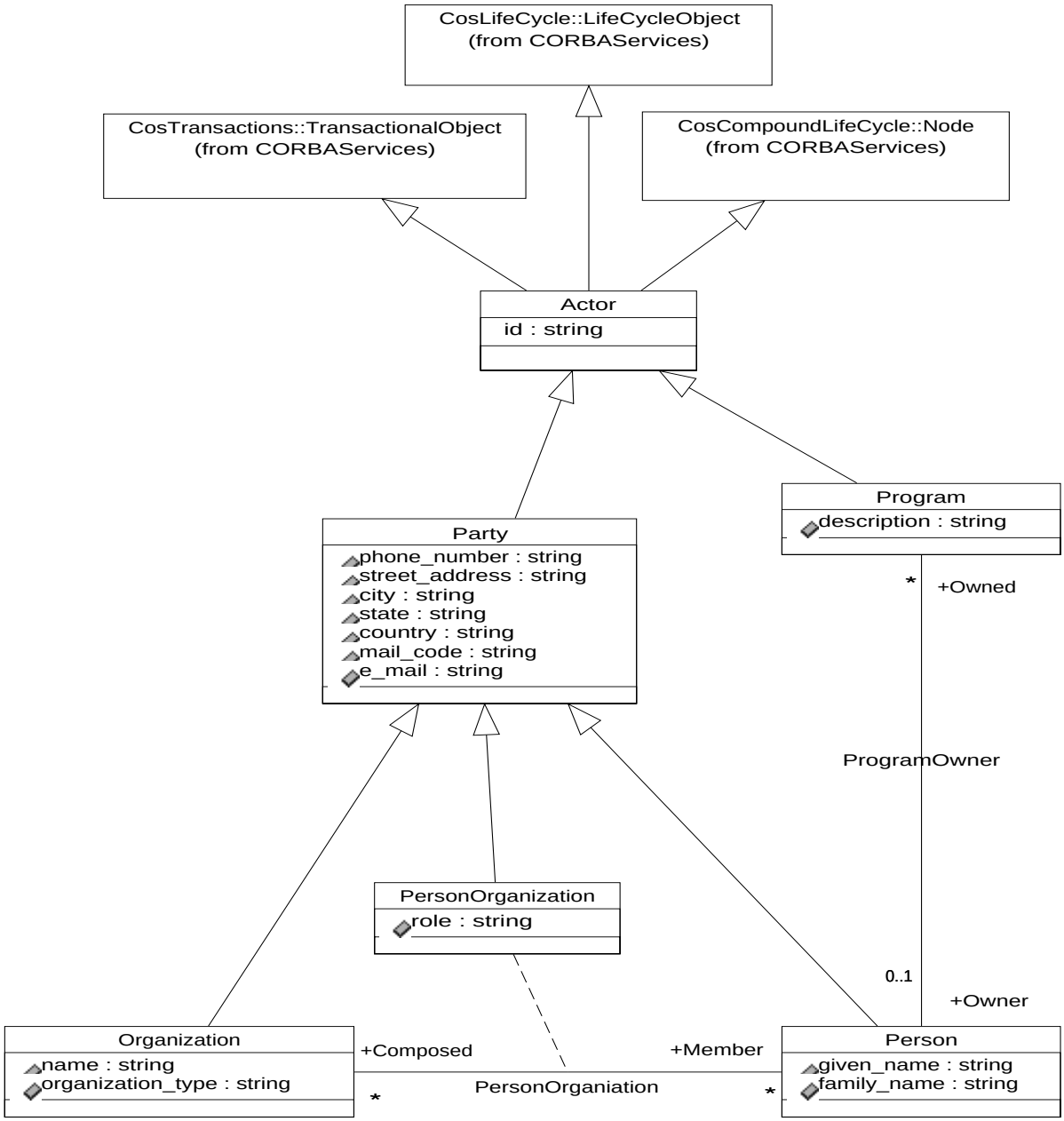

Figure 2-1 PDMResponsibility Model Diagram

2.3 PdmResponsibility Description

2.3.1 Exceptions

The following exceptions describe the various error conditions that are specified in the operation definitions of the **PdmResponsibility** module. All the exceptions include a numeric error code and an error text string. The value of the error code and the error text will be defined by each implementation and is used to provide greater detail into the cause of the exception. The InvalidProperties exception also includes a **validation_errors** parameter that is used to indicate the value or values within the property set which were considered invalid.

```
struct PdmPropertyValidationError
{
  CosPropertyService::PropertyName property_name;
  unsigned long error_code;
  string error_text;
};

typedef sequence<PdmPropertyValidationError>
PdmPropertyValidationErrors;

exception GeneralError
  {unsigned long error_code; string error_text;};
exception PermissionDenied
  {unsigned long error_code; string error_text;};
exception ValidationError
  {unsigned long error_code; string error_text;};
exception NotUnique
  {unsigned long error_code; string error_text;};
exception InvalidProperties
{
  unsigned long error_code; string error_text;
  PdmPropertyValidationErrors validation_errors;
};
exception CardinalityExceeded
{
  unsigned long error_code; string error_text;
  string role_name;
};
```

GeneralError

The GeneralError exception is raised when the **PdmResponsibility** module detects an error that is not described by one of the other standard exceptions.

PermissionDenied

The PermissionDenied exception is raised when the **PdmResponsibility** module does not allow the current user to perform the attempted operation on the object.

ValidationError

The ValidationError exception is raised when the **PdmResponsibility** module detects either an invalid value or another condition that would result in invalid information in the **PdmResponsibility** module.

NotUnique

The NotUnique exception is raised when an operation attempts to create a new object, but an object with a conflicting key or uniqueness constraint already exists in the **PdmResponsibility** module.

NotFound

The NotFound exception is raised by find operations when the **PdmResponsibility** module cannot find an object that matches the search criteria specified by the operation.

InvalidProperties

The InvalidProperties exception is raised by operations with a property set in its signature when a property in the set cannot be processed or a required property is not included in the set. The additional **validation_errors** parameter will indicate the property or properties causing the exception.

CardinalityExceeded

The CardinalityExceeded exception is raised by operations that attempt to create a relationship that would cause the cardinality of an objects role to exceed its maximum value.

2.3.1.1 Exception Lists

The following **#define** statements are used to improve the readability of the operation definitions by group the list of standard exceptions for a class of operations, allowing additional exceptions to be easily seen.

```
#define RESPONSIBILITY_ITEM_CREATE_EXCEPTIONS \
  PdmResponsibility::GeneralError, \
  PdmResponsibility::PermissionDenied, \
  PdmResponsibility::ValidationError, \
  PdmResponsibility::InvalidProperties, \
  PdmResponsibility::NotUnique

#define RESPONSIBILITY_RELATIONSHIP_CREATE_EXCEPTIONS \
  PdmResponsibility::GeneralError, \
  PdmResponsibility::PermissionDenied, \
  PdmResponsibility::ValidationError, \
  PdmResponsibility::InvalidProperties, \
  PdmResponsibility::NotUnique, \
  PdmResponsibility::CardinalityExceeded
```

2.3.2 Actor

An **Actor** represents any entity that has the ability to initiate actions on other objects. This is abstract object and is used where a person, organization, or program needs to be associated with other object.

```
interface Actor : CosLifeCycle::LifeCycleObject,
  CosTransactions::TransactionalObject,
  CosCompoundLifeCycle::Node
{
  attribute string id;
};
```

2.3.3 Program

A **Program** represents non-human agents that invoke actions on other objects, usually based on specific events within the system. This may represent either a single run of a program or simply the program itself.

```
interface Program : Actor
{
  attribute string description;
};

interface ProgramFactory
{
  Program create( in CosPropertyService::PropertySet property_set)
    raises(RESPONSIBILITY_ITEM_CREATE_EXCEPTIONS);
};
```

The **property_set** parameter on the **create** operation is used to specify initial attribute values.

2.3.4 Party

A **Party** is an abstract interface and is used where a person, organization, or groups of people need to be referenced.

The following attributes are based on similar attributes used to describe people in ISO 10303-203 (STEP AP203).

```
interface Party : Actor
{
  attribute string phone_number;
  attribute string street_address;
  attribute string city;
  attribute string state;
  attribute string country;
```

```
  attribute string mail_code;
  attribute string e_mail;
};
```

2.3.5 Person

A **Person** is used to represent an individual. Like other interfaces in this module, **Person** should be defined as a common business object and therefore only a minimal set of attributes have been defined in this specification.

```
interface Person : Party
{
  attribute string given_name;
  attribute string family_name;
};

interface PersonFactory
{
  Person create( in CosPropertyService::PropertySet property_set)
    raises(RESPONSIBILITY_ITEM_CREATE_EXCEPTIONS);
};
```

2.3.6 Organization

The **Organization** object represents a group of people that operate as a unit (that is, departments, teams, and suppliers).

```
interface Organization : Party
{
  attribute string name;
  attribute string organization_type;
};
interface OrganizationFactory
{
  Organization create( in CosPropertyService::PropertySet property_set)
    raises(RESPONSIBILITY_ITEM_CREATE_EXCEPTIONS);
};
```

2.3.7 PersonOrganization

The **PersonOrganization** relationship identifies that a **Person** belongs to an **Organization**. The **PersonOrganization** may be referenced in objects like approvals when it is important to know not only the **Person**, but the specific organization that person is representing during a particular event.

```
// PersonOrganization Relationship
//    role:  Member
//       name:  'Member'
//       entity:  Person
```

```
//      cardinality:  0..unbounded
//    role:  Composed
//      name:  'Composed'
//      cardinality:  0..unbounded

interface PersonOrganization : CosLifeCycleReference::Relationship,
  Party
{
  attribute string role;
};

interface PersonOrganizationFactory
{
  PersonOrganization create(
    in CosPropertyService::PropertySet property_set,
    in Person the_member,
    in Organization the_composed)
    raises(RESPONSIBILITY_RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface Member : CosLifeCycleReference::ReferencesRole { };

interface Composed : CosLifeCycleReference::ReferencedByRole { };
```

2.3.7.1 *role*

The role or responsibility that the person has within the specified organization.

2.3.8 *ProgramOwner*

The **ProgramOwner** relationship object relates a program to the person that is responsible for the program.

```
// ProgramOwner Relationship
//    role:  Owned
//      name:  'Owned'
//      entity:  Program
//      cardinality:  0..1
//    role:  Owner
//      name:  'Owner'
//      entity:  Person
//      cardinality:  0..unbounded

interface ProgramOwner : CosLifeCycleReference::Relationship { };

interface ProgramOwnerFactory
{
  ProgramOwner create(in CosPropertyService::PropertySet property_set,
    in Person the_owner,
    in Program the_owned)
```

```
    raises(RESPONSIBILITY_RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface Owned : CosLifeCycleReference::ReferencesRole { };

interface Owner : CosLifeCycleReference::ReferencedByRole { };
```

2.4 PdmResponsibility IDL

```
// PdmResponsibility.idl

#ifndef  PDMRESPONSIBILITY
#define  PDMRESPONSIBILITY

#ifdef   SOM_COMPILE
#include <somobj.idl>                  // SOM COMPILE
#endif

#ifdef  ORBIX_COMPILE
#ifndef  IFR
#define  IFR
#include <ifr.idl>
#endif
#endif

#include <CosLifeCycle.idl>
#include <CosLifeCycleReference.idl>
#include <CosTransactions.idl>
#include <CosPropertyService.idl>
#include <CosCompoundLifeCycle.idl>

module PdmResponsibility {

// Exceptions

  struct PdmPropertyValidationError
  {
    CosPropertyService::PropertyName property_name;
    unsigned long error_code;
    string error_text;
  };

  typedef sequence<PdmPropertyValidationError>
    PdmPropertyValidationErrors;

  exception GeneralError
    {unsigned long error_code; string error_text;};
  exception PermissionDenied
    {unsigned long error_code; string error_text;};
  exception ValidationError
```

```
  {unsigned long error_code; string error_text;};
exception NotUnique
  {unsigned long error_code; string error_text;};
exception InvalidProperties
{
  unsigned long error_code; string error_text;
  PdmPropertyValidationErrors validation_errors;
};
exception CardinalityExceeded
{
  unsigned long error_code; string error_text;
  string role_name;
};

#define RESPONSIBILITY_ITEM_CREATE_EXCEPTIONS \
  PdmResponsibility::GeneralError, \
  PdmResponsibility::PermissionDenied, \
  PdmResponsibility::ValidationError, \
  PdmResponsibility::InvalidProperties, \
  PdmResponsibility::NotUnique

#define RESPONSIBILITY_RELATIONSHIP_CREATE_EXCEPTIONS \
  PdmResponsibility::GeneralError, \
  PdmResponsibility::PermissionDenied, \
  PdmResponsibility::ValidationError, \
  PdmResponsibility::InvalidProperties, \
  PdmResponsibility::NotUnique, \
  PdmResponsibility::CardinalityExceeded

// Entities

interface Actor : CosLifeCycle::LifeCycleObject,
     CosTransactions::TransactionalObject,
     CosCompoundLifeCycle::Node
{
  attribute string id;
};

interface Party : Actor
{
  attribute string phone_number;
  attribute string street_address;
  attribute string city;
  attribute string state;
  attribute string country;
  attribute string mail_code;
  attribute string e_mail;
};

interface Organization : Party
{
```

```
    attribute string name;
    attribute string organization_type;
};

interface OrganizationFactory
{
    Organization create( in CosPropertyService::PropertySet
        property_set)
        raises(RESPONSIBILITY_ITEM_CREATE_EXCEPTIONS);
 };

interface Person : Party
{
    attribute string given_name;
    attribute string family_name;
};

interface PersonFactory
{
    Person create( in CosPropertyService::PropertySet property_set)
        raises(RESPONSIBILITY_ITEM_CREATE_EXCEPTIONS);
};

interface Program : Actor
{
    attribute string description;
};

interface ProgramFactory
{
    Program create( in CosPropertyService::PropertySet property_set)
        raises(RESPONSIBILITY_ITEM_CREATE_EXCEPTIONS);
};

// Relationships

// PersonOrganization Relationship
//    role:  Member
//       name:  'Member'
//       entity:  Person
//       cardinality:  0..unbounded
//    role:  Composed
//       name:  'Composed'
//       cardinality:  0..unbounded

interface PersonOrganization : CosLifeCycleReference::Relationship,
        Party
{
    attribute string role;
};
```

```
  interface PersonOrganizationFactory
  {
    PersonOrganization create(
      in CosPropertyService::PropertySet property_set,
      in Person the_member,
      in Organization the_composed)
      raises(RESPONSIBILITY_RELATIONSHIP_CREATE_EXCEPTIONS);
  };

  interface Member : CosLifeCycleReference::ReferencesRole { };

  interface Composed : CosLifeCycleReference::ReferencedByRole { };

  // ProgramOwner Relationship
  //   role:  Owned
  //     name:  'Owned'
  //     entity:  Program
  //     cardinality:  0..1
  //   role:  Owner
  //     name:  'Owner'
  //     entity:  Person
  //     cardinality:  0..unbounded

  interface ProgramOwner : CosLifeCycleReference::Relationship { };

  interface ProgramOwnerFactory
  {
    ProgramOwner create(in CosPropertyService::PropertySet
      property_set,
      in Person the_owner,
      in Program the_owned)
      raises(RESPONSIBILITY_RELATIONSHIP_CREATE_EXCEPTIONS);
  };

  interface Owned : CosLifeCycleReference::ReferencesRole { };

  interface Owner : CosLifeCycleReference::ReferencedByRole { };

  };

#endif
```

PdmFoundation Module 3

Contents

This chapter contains the following sections.

Section Title	Page
“Overview”	3-1
“PdmFoundation Model”	3-2
“PdmFoundation Description”	3-3
“PdmFoundation IDL”	3-20

3.1 Overview

The following classes represent behaviors that have been identified as required by multiple PDM enablers and which we expect to be required by other interfaces, external to the PDM enablers. These classes represent fundamental elementary units of behavior. We expect several of these classes to become part of Common Business Objects or other domains, but until such time it is necessary to define the behavior that we require of the objects today.

In most of the initial submissions, a single “PDM object” class was defined, which described several generic characteristics required of most of the objects defined within the enablers. In the follow-on work, we have tried to separate out the different types of behaviors into separate foundation classes, allowing each object to have greater control to pick and choose the behaviors it will support. This should also make it easier to apply these requirements to a generic business object framework that can be defined outside a specific domain area.

Currently we have defined six basic foundation classes that many of the enabler objects will inherit from. These are *Manageable, Lockable, Revisionable, Iteratable, Stateable,* and *SecurityClassifable*. The following UML diagram describes these classes along with several other related classes.

3.2 PdmFoundation Model

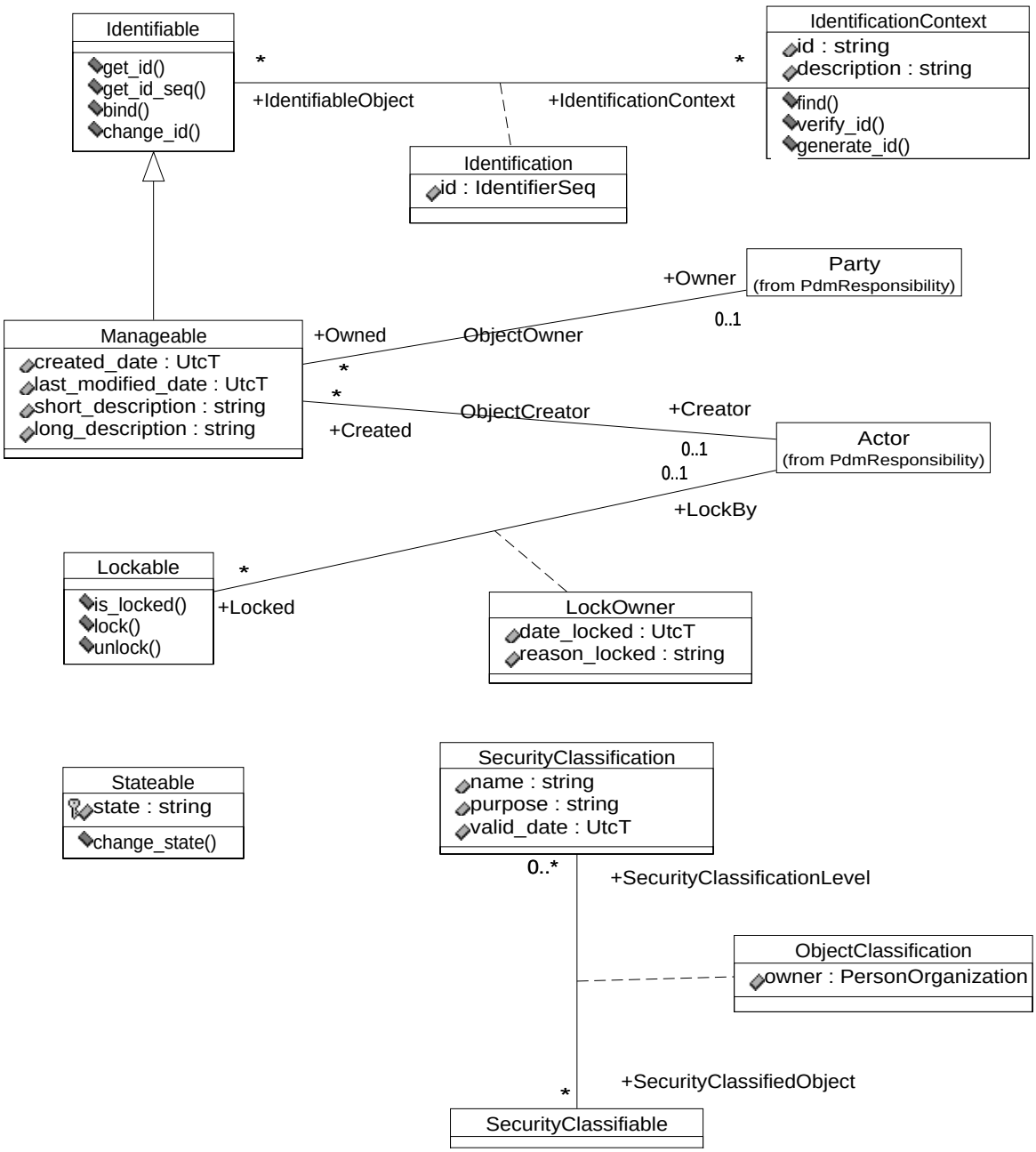

Figure 3-1 PdmFoundation Model Diagram

Note – IdentificationContext inherits from **CosTransactions::TransactionalObject**, **CosLifeCycle::LifeCycleObject**, and **CosCompoundLifeCycle::Node**.

3.3 PdmFoundation Description

3.3.1 Exceptions

3.3.1.1 General Exceptions

The following exceptions are common to all the following PDM modules. Defined in the **PdmFoundation** module and included by other modules, these exceptions describe the various error conditions that each operation definition will indicate as potential being raised by its implementation. Exceptions not shared by all modules shall be defined in the module where used. All the exceptions defined in the PDM Enablers specification include a numeric error code and an error text string. The value of the error code and the error text will be defined by each implementation and is used to provide greater detail into the cause of the exception. The InvalidProperties exception also includes a **validation_errors** parameter that is used to indicate the value or values within the property set that were considered invalid.

```
struct PdmPropertyValidationError
{
  CosPropertyService::PropertyName property_name;
  unsigned long error_code;
  string error_text;
};

typedef sequence<PdmPropertyValidationError>
  PdmPropertyValidationErrors;

exception PdmError
  {unsigned long error_code; string error_text;};
exception PermissionDenied
  {unsigned long error_code; string error_text;};
exception ValidationError
  {unsigned long error_code; string error_text;};
exception NotUnique
  {unsigned long error_code; string error_text;};
exception NotFound
  {unsigned long error_code; string error_text;};
exception InvalidProperties
{
  unsigned long error_code; string error_text;
  PdmPropertyValidationErrors validation_errors;
};
exception CardinalityExceeded
```

```
{
  unsigned long error_code; string error_text;
  string role_name;
};
```

PdmError

The PdmError exception is raised when the PDM system detects an error that is not described by one of the other standard exceptions.

GeneralError

The GeneralError exception is raised when the **PdmResponsibility** module detects an error that is not described by one of the other standard exceptions.

PermissionDenied

The PermissionDenied exception is raised when the **PdmResponsibility** module does not allow the current user to perform the attempted operation on the object.

ValidationError

The ValidationError exception is raised when the **PdmResponsibility** module detects either an invalid value or another condition that that would result in invalid information in the **PdmResponsibility** module.

NotUnique

The NotUnique exception is raised when an operation attempts to create a new object, but an object with a conflicting key or uniqueness constraint already exists in the **PdmResponsibility** module.

NotFound

The NotFound exception is raised by find operations when the **PdmResponsibility** module cannot find an object that matches the search criteria specified by the operation.

InvalidProperties

The InvalidProperties exception is raised by operations with a property set in its signature when a property in the set can not be process or a required property is not included in the set. The additional **validation_errors** parameter will indicate the property or properties causing the exception.

CardinalityExceeded

The CardinalityExceeded exception is raised by operations that attempt to create a relationship that would cause the cardinality of an objects role to exceed its maximum value.

3.3.1.2 PdmFoundation Exceptions

The following exceptions are used only by operations defined in the **PdmFoundation** module.

```
exception IdentifierNotDefined
  {unsigned long error_code; string error_text;};
exception GenerateNotAvailable
  {unsigned long error_code; string error_text;};
exception AlreadyLocked
  {unsigned long error_code; string error_text;};
exception UnlockFailed
  {unsigned long error_code; string error_text;};
exception InvalidTransition
  {unsigned long error_code; string error_text;};
```

IdentifierNotDefined

The IdentifierNotDefined exception is raised by operations that are unable to find an Identifiable object or its identification.

GenerateNotAvailable

The GenerateNotAvailable exception is raised by operations which attempt to automatically generate or define an Identifiable objects id within an **IdentificationContext** which does not support this capability.

AlreadyLocked

The AlreadyLocked exception is raised by operations that attempt to lock a Lockable object that is already locked by another user.

UnlockFailed

The UnlockFailed exception is raised when an attempt to release a lock on a Lockable object fails.

InvalidTransition

The InvalidTransition exception is raised when an attempt to change the state of a Stateable object fails.

3.3.1.3 Exception Lists

The following **#define** statements are used to improve the readability of the operation definitions by group the list of standard exceptions for a class of operations, allowing additional exceptions to be easily seen.

```
#define PDM_EXCEPTIONS \
  PdmFoundation::PdmError, \
  PdmFoundation::PermissionDenied, \
```

```
    PdmFoundation::ValidationError

#define ITEM_CREATE_EXCEPTIONS \
    PdmFoundation::PdmError, \
    PdmFoundation::PermissionDenied, \
    PdmFoundation::ValidationError, \
    PdmFoundation::InvalidProperties, \
    PdmFoundation::NotUnique

#define RELATIONSHIP_CREATE_EXCEPTIONS \
    PdmFoundation::PdmError, \
    PdmFoundation::PermissionDenied, \
    PdmFoundation::ValidationError, \
    PdmFoundation::InvalidProperties, \
    PdmFoundation::NotUnique, \
    PdmFoundation::CardinalityExceeded
```

3.3.2 Measurement

A measurement is only useful if both the quantity and the unit of measure are known. The Measurement structure is a common typedef used in other modules to record both the quantity and the measurement unit. Additionally, an approximate flag allows the user to indicate if the measurement is exact or represents only an average or as-required quantity.

```
struct Measurement
{
    double quantity;
    string uom;
    boolean approximate;
};
```

quantity

A measurement quantity expressed in a specific unit of measure.

uom

The units in which the measurement quantity is specified. The attribute is typed as a string instead of an enum because an accepted standard for units of measure does not exist.

approximate

If false, the quantity represents an exact amount. If true, the quantity represents an average or estimated quantity. May be used for planning when the quantity is as required.

3.3.3 IdentifierSeq

The **IdentifierSeq** type is used to record the identification or name of an object. In many cases, an objects' identification is a single value, but there are instances where the identification is made up of several components. An example may be a bin name that includes room, row, and column. The components of the id are limited to string values, allowing the id to be stored in a sequence of named string pairs instead of named value pairs.

```
struct NSPair {string name; string value;};
typedef sequence<NSPair> IdentifierSeq;
```

3.3.4 Identifiable

Identifiable is an abstract object that describes objects that may have different identifications within different contexts. For instance, a part object may have an internal part number, a government part number, and a catalog part number. The actual identifier is a property of the **IdentificationRelation**. An identifiable instance will have a separate **IdentificationRelation** for each external id it has. The id must be unique within the related **IdentificationContext**. An identifiable instance should have only one id per **IdentificationContext**.

Identifiable also allows an enterprise to define multiple numbering schemes for the same class of objects since the numbering scheme will be a property of the **IdentificationContext**.

```
interface Identifiable
{
  string get_id(in IdentificationContext id_context)
    raises(IdentifierNotDefined, PDM_EXCEPTIONS);
  IdentifierSeq get_id_seq(in IdentificationContext id_context)
    raises(IdentifierNotDefined, PDM_EXCEPTIONS);

  IdentifierSeq bind(
      in CosPropertyService::PropertySet property_set,
      in IdentificationContext the_context)
    raises(RELATIONSHIP_CREATE_EXCEPTIONS);

  void change_id( in CosPropertyService::PropertySet property_set,
      in IdentificationContext the_context)
    raises(PDM_EXCEPTIONS, PdmFoundation::NotUnique);

};
```

Because the objects' identification is actually stored on a relationship with context, two convenience operations are specified for fetching the identification.

get_id

Returns a string with components of the identification concatenated together for the context specified.

bind

Creates a new **Identification** relation between the current **Identifiable** object and the **IdentificationContext** specified. The values passed in via the **property_set** are used to generate the id, which is stored in the new **IdentificationRelation** and returned as a result of the interface.

change_id

Changes the identification within the specified identification context.

get_id_seq

Returns the sequence of name-string pairs containing each of the components of the identification for the context specified.

3.3.5 *IdentificationContext*

The **IdentificationContext** object describes the context or scope in which object identifications apply and the identification formats, and rules that apply in the context. For example, an identification context may define Navy Part numbers and include the format and rules that the Navy requires for their naming parts. Context objects can be defined for different classes of objects and different scopes, such as government agencies, vendors, different internal divisions.

```
typedef CosNaming::NameComponent  IdentificationContextName;
typedef sequence <IdentificationContextName>
  IdentificationContextNames;

interface IdentificationContext : CosLifeCycle::LifeCycleObject,
  CosTransactions::TransactionalObject, CosCompoundLifeCycle::Node
{
  attribute IdentificationContextName name;
  attribute string description;

  Identifiable find(in IdentifierSeq the_id)
    raises(NotFound, PDM_EXCEPTIONS);
  boolean verify_id(in IdentifierSeq the_id)
    raises(ValidationError, PDM_EXCEPTIONS);
  IdentifierSeq generate_id(
    in CosPropertyService::PropertySet property_set,
    in Identifiable the_object)
    raises(GenerateNotAvailable, PDM_EXCEPTIONS);
};

typedef sequence <IdentificationContext> IdentificationContexts;
```

```
interface IdentificationContextFactory
{
  IdentificationContext create(
    in CosPropertyService::PropertySet property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
  IdentificationContext find_id_context(
    in IdentificationContextName the_context_name)
    raises(NotFound, PDM_EXCEPTIONS);
  IdentificationContexts find_all_id_contexts()
    raises (PDM_EXCEPTIONS);
  IdentificationContexts find_id_contexts( in string identifiableType )
    raises(NotFound, PDM_EXCEPTIONS);
}
```

Implementations may provide a separate **IdentificationContext** for each **Identifiable** type, or may provide one or more **IdentificationContext** objects that support multiple types of Identifiable object.

Each **IdentificationContext** is identified by a **CosNaming::NameComponent**, defined by **CosNaming** as:

```
struct NameComponent {
  Istring id;
  Istring kind;
}
```

The **id** field represents the business-related name. If the implementation supports a default identification context or only one identification context for a kind, the default **id** is represented by the empty string (that is, a string of length 0).

The **kind** field indicates the type of the **Identifiable** objects managed by the **IdentificationContext**. If the **IdentificationContext** supports a single type of **Identifiable** item, the **kind** field has the form of the interface repository **RepositoryId** for the **InterfaceDef** of the **Identifiable** object. If the **IdentificationContext** supports more than one type of **Identifiable** item, then the kind field may contain the **RepositoryId** for the common supertype of the items, or the content of the **kind** field may be implementation dependent.

find

Returns the **Identifiable** object associated with the specified **IdentificationSeq** within the current context.

verify_id

Reports whether the specified **id** confirms to the naming rules for this context.

generate_id

Generates an **Identification** relation between the specified **Identifiable** and **IdentificationContext** objects and returns the assigned **IdentifierSeq**. The Identification's **IdentifierSeq** is generated using the rules defined within the Id Context.

3.3.6 Factory Operations

create

Creates and returns a new **IdentificationContext** instance. The **propery_set** parameter may be used to specify initial attribute values.

find_id_context

Returns the **IdentificationContext** instance with the context name specified.

find_all_id_contexts

Returns all identification contexts supported by the implementation.

find_id_contexts

Returns all identification contexts that identify objects of a given type. The **identifiableType** parameter is in the form of the interface repository **RepositoryId** for the **InterfaceDef** of the **Identifiable** object.

3.3.7 Additional Identification Description

3.3.7.1 Background

Consider the following:

- A company's part numbering system may be different from its document numbering system, and its ECO numbering system will likely be different from those of parts and documents.
- A part may have more than one part number. Company part number as well as the suppliers' part numbers may identify purchased parts. The Navy may insist that parts submitted for approval be identified using the Navys own numbering system.
- A company uses dash numbers for document revisions. The DoD insists that documents submitted for review and approval use MIL-STD-100 alphabetic coding system.
- A document may go through several formal revisions, but the Army insists that the first revision submitted for its approval be identified as revision A.

- Different kinds of items and documents may have different identification nomenclature, similarly for revision identification. It is also not uncommon for a company that has acquired other companies to have multiple numbering systems for items and documents.

3.3.7.2 Design

The characteristic of an **Identifiable** item is that its identification is defined within the scope of an **IdentificationContext**. The actual identifier is a property of the **IdentificationRelation** between the **Identifiable** item and the **IdentificationContext**. The **Identifiable** item has two methods:

1. **get_id()** returns a string representing the identifier within a specified context (supplied as an argument).
2. **get_id_seq()** returns a sequence of strings representing the components of the identifier.

For example, an intelligent part number may comprise multiple components, and the **get_id_seq()** method will return the individual nomenclature components, which the **get_id()** method will return a single string combining the nomenclature components.

Examples of Identifiable are Part Master, Document Master, Document Revision, ECO, Person, and Organization. For example, company name, cage code, Dun & Bradstreet number, or all of the above depending on the context may identify a company.

Is **IdentificationContext** a sub-type of **Identifiable**? The answer is no. An **IdentificationContext** will have a name that must be unique among all **IdentificationContext**s, including subtypes.

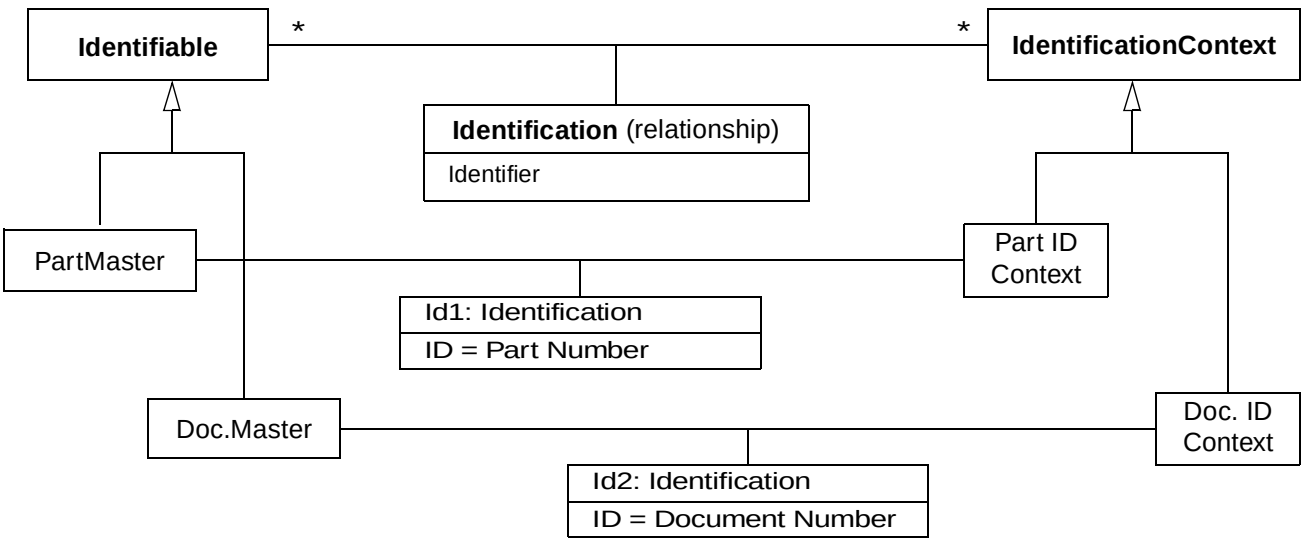

Figure 3-2 Example of how the identification scheme might work

Figure 3-3 shows how different descendents of Identifiable may have ids associated with different **IdentificationContext**s. The unique rules and logic required to ensure valid ids for Parts and Documents are specified in the corresponding descendent objects of **IdentificationContext**.

Figure 3-4 shows how the Identification relationship can be applied to a purchased part having different part numbers in different contexts. Even if the part is not a purchased part, external customers, such as the military, may demand that the part be identified to them using their own numbering nomenclature.

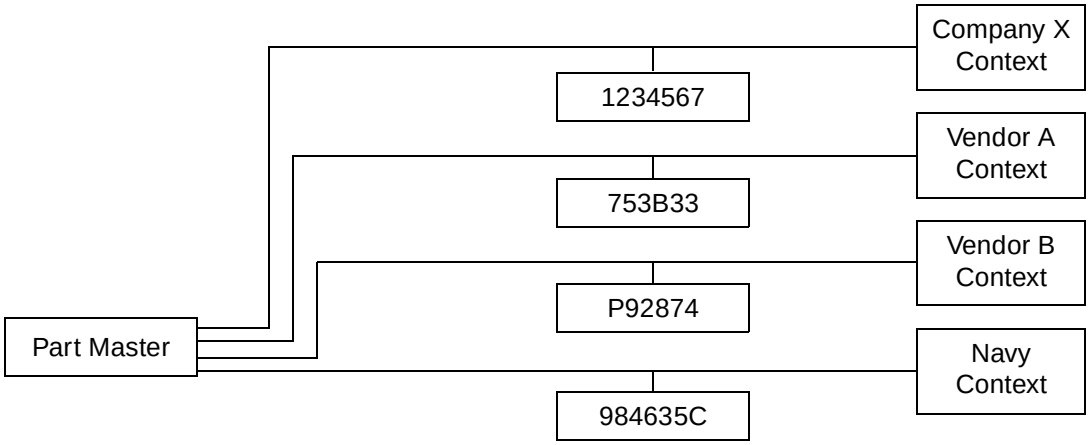

Figure 3-3 A purchased part has different part numbers in different contexts.

PartRevisions and **DocumentRevision**s may also have multiple **IdentificationContext**s. The format of part revision ids may differ from document revision ids or an external customer may insist on a part revision nomenclature that is different than the revision ids used internally.

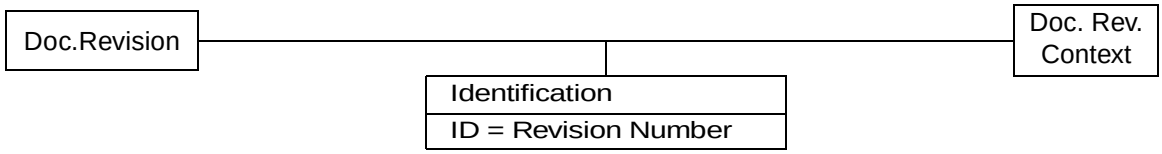

Figure 3-4 Document Revision identification in the context of a Document Revision Context.

One difference between revision identification and master identification is that the ID of a master is required to be unique within a given identification context, while the uniqueness of a revision identifier is a function of the identification context and the master of the revision. For example, document 123456 cannot have two revisions named A, but another document, 987654 can have a revision named A.

3.3.7.3 *Implementation Issues*

There are many ways of implementing this object model and the performance of the implementation would be one way for PDM suppliers to differentiate their products.

How does one obtain an identifier predicated on an identification context? Certainly, one can navigate the identification relationships, but it would be more convenient to use the **get_id()** and **get_id_seq()** methods on the **Identifiable** object.

This will allow an implementation more optimization opportunities. For example, an implementation may choose to redundantly copy the identifier for the most commonly used context onto the **Identifiable** object itself to avoid having to traverse a relationship.

Note that there is no single default context. Different clients may want to use different identification contexts.

3.3.8 *Lockable*

Lockable is an abstract object that defines the characteristics that allow an object to be locked by an actor, preventing others from successfully attaining a lock. The lock is persistent and will remain in place until explicitly released. An object may be locked to prevent simultaneous updates by different users or to freeze an object at points in its lifecycle.

Note – It is up to the inheriting classes to include the locking or checks for locks on dependent objects. For example, it is up to the Document classes to define what happens to DocumentRevisions when the Document object is locked.

```
interface Lockable
{
  boolean is_locked();

  void lock(in PdmResponsibility::Actor lock_owner, in string reason)
    raises( AlreadyLocked, PDM_EXCEPTIONS);
  void unlock(in PdmResponsibility::Actor lock_owner)
    raises( UnlockFailed, PDM_EXCEPTIONS);
};
```

is_locked

False is the object is not locked. True is a lock exist.

lock

This method checks to insure that a lock does not exist for the object and if none, adds a lock for the actor specified. If another actor already has a lock on this object, the AlreadyLocked exception is raised.

unlock

This method will release a lock previously granted to the specified user.

3.3.9 Manageable

Manageable is an abstract object that represents the minimum set of behavior required by an object in order to be manipulated by the PDM enablers. All optional behaviors have been split off into other abstract classes.

```
interface Manageable : Identifiable
{
  attribute TimeBase::UtcT created_date;
  attribute TimeBase::UtcT last_modified_date;
  attribute string short_description;
  attribute string long_description;
};
```

created_date

The date and time that the instance was created.

last_modified_date

The date and time that the instance was last updated.

short_description

Brief description of the object.

long_description

Detail description of the object.

3.3.10 SecurityClassifiable

SecurityClassifiable is an abstract object that describes objects that can be assigned a Security Classification. The Security Classification represents the level to which the data in this object must be protected. This does not represent access authority, which is controlled by the Security Service, but may be used by that service to determine access rights.

```
interface SecurityClassifiable { };
```

3.3.11 SecurityClassification

The **SecurityClassification** object is used to define the valid security classification level used within the system. This object is only to identify the security classification level of other objects and does not manage access control.

```
interface SecurityClassification : CosLifeCycle::LifeCycleObject,
  CosTransactions::TransactionalObject
{
  attribute string name;
  attribute string purpose;
  attribute TimeBase::UtcT valid_date;
};

interface SecurityClassificationFactory
{
  SecurityClassification create(
    in CosPropertyService::PropertySet property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
  SecurityClassification find_sec_class(in string in_name)
    raises(NotFound, PDM_EXCEPTIONS);
};
```

3.3.12 SecurityClassification Attributes

name

Name of a valid security classification.

purpose

Purpose of this security classification.

valid_date

The date when this security classification is valid for use.

3.3.13 Factory Operations

create

Creates and returns a new **SecurityClassification** instance. The **propery_set** parameter may be used to specify initial attribute values.

find_sec_class

Returns the **SecurityClassification** instance with the name specified.

3.3.14 Stateable

Stateable is an abstract object that defines the behavior required to indicate the current state of objects in the PDM context (for example, approval status) and the method to change the state.

The **Stateable** object does not attempt to define the valid states, the valid transitions between states or any of the affects associated with changing state. This is left to specific implementation or a workflow interface.

Our design objective is to define the PDM enablers such that a system can be implemented using either an automated or manual workflow control process. To avoid overlap with the workflow specification, our focus is on the minimum behavior required to manually manipulate an objects state and assuming that forthcoming workflow controls can build on this minimum set in order to automate the workflow process.

```
interface Stateable
{
  readonly attribute string state;

  void change_state(in string in_state)
    raises( InvalidTransition, PDM_EXCEPTIONS );
};
```

state

Identifies the current state of the instance. Descendents may define allowed values.

change_state

Causes the state to be changed to the state passed as a parameter. Descendents may add additional validation checks and/or processing required in order to change the state.

3.3.15 Identification

The **Identification** relationship object contains the identifier for a **Identifiable** instance within a specific **IdentificationContext**. An **Identifiable** instance may have several identifiers, but it should not have multiple identifiers for the same **IdentificationContext**. Additionally, the identifier shall be unique within a given **IdentificationContext**.

Note that most of the operations on **Identification** are encapsulated by methods on **Identifiable** and **IdentificationContext**. The exception is retrieving a set of all IDs of an **Identifiable** object, which is attained using the **Relationship** service on the relation.

```
// Identification Relationship
//  role : IdentifiableObjectRole
//    name : 'IdentifiablebjectRole'
//    entity : Identifiable
//    cardinality : 0..unbounded
//  role : IdentificationContextRole
//    name : 'IdentificationContextRole'
//    entity : IdentificationContext
//    cardinality : 0..unbounded
```

```
interface Identification : CosLifeCycleReference::Relationship
{
  readonly attribute IdentifierSeq id;

};

interface IdentifiableObjectRole :
  CosLifeCycleReference::ReferencesRole {};

interface IdentificationContextRole :
  CosLifeCycleReference::ReferencedByRole {};
```

id

A unique external name for the **Identifiable** instance within an **IdentificationContext**. Using the sequence of NS-pairs, the identifier may consist of more than one field.

3.3.16 LockOwner

The **LockOwner** relationship object identifies the Actor that currently has a specific instance locked. When locked, the actor holding the lock can only update the instance. An object should have at most one **LockOwner** relationship at any point in time.

The **LockOwner** relation is created and deleted using the **Lockable:Lock** and **Lockable:Unlock**. The **Relationship** service on the **LockOwner** relation can be used to retrieve a set of all the locks active on a **Lockable** object.

```
// LockOwner Relationship
//  role : Locked
//    name : 'Locked'
//    entity : Lockable
//    cardinality : 0..1
//  role : LockBy
//    name : 'LockBy'
//    entity : PdmResponsibility::Actor
//    cardinality : 0..unbounded

interface LockOwner : CosLifeCycleReference::Relationship
{
  readonly attribute TimeBase::UtcT date_locked;
  readonly attribute string reason_locked;
};

interface Locked : CosLifeCycleReference::ReferencesRole { };

interface LockBy : CosLifeCycleReference::ReferencedByRole { };
```

date_locked

The date and time that the current lock was granted. Returns NULL if no lock exists.

reason_locked

The reason the object was locked. Possible Values: Review, CheckedOut, Reserved, Released, etc.

3.3.17 ObjectCreator

The **ObjectCreator** relationship object identifies the actor who originally created a **Manageable** instance. Each manageable instance should have only one creator.

```
// ObjectCreator Relationship
//  role : Created
//    name : 'Created'
//    entity : Manageable
//    cardinality : 0..1
//  role : Creator
//    name : 'Creator'
//    entity : PdmResponsibility::Actor
//    cardinality : 0..unbounded

interface ObjectCreator : CosLifeCycleReference::Relationship { };

interface ObjectCreatorFactory
{
  ObjectCreator create(
     in CosPropertyService::PropertySet property_set,
     in Manageable the_createe,
     in PdmResponsibility::Actor the_creator)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface Created : CosLifeCycleReference::ReferencesRole { };

interface Creator : CosLifeCycleReference::ReferencedByRole { };
```

3.3.18 ObjectOwner

The **ObjectOwner** relationship object identifies the **Party** that currently has ownership of a **Manageable** instance. Other parties, based on access authority, may modify the instance, but typically the objects owner to insure that all updates are appropriate.

```
// ObjectOwner Relationship
//  role : Owned
//    name : 'Owned'
//    entity : Manageable
//    cardinality : 0..1
//  role : Owner
//    name : 'Owner'
//    entity : PdmResponsibility::Party
```

```
//    cardinality : 0..unbounded

interface ObjectOwner : CosLifeCycleReference::Relationship { };

interface ObjectOwnerFactory
{
  ObjectOwner create(
     in CosPropertyService::PropertySet property_set,
     in Manageable the_ownee,
     in PdmResponsibility::Party the_owner)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface Owned : CosLifeCycleReference::ReferencesRole { };

interface Owner : CosLifeCycleReference::ReferencedByRole { };
```

3.3.19 ObjectSecurityClassification

The **ObjectSecurityClassification** relationship object is used to identify the security level for a specified object. This determines the degree to which the data must be protected.

```
// ObjectSecurityClassification Relationship
//  role : SecurityClassificationLevel
//    name : 'SecurityClassificationLevel'
//    entity : SecurityClassification
//    cardinality : 0..unbounded
//  role : SecurityClassifiedObject
//    name : 'SecuirtyClassifiedObject'
//    entity : SecurityClassifiable
//    cardinality : 0..unbounded

interface ObjectSecurityClassification :
  CosLifeCycleReference::Relationship
{
  attribute PdmResponsibility::Person owner;
};

interface ObjectSecurityClassificationFactory
{
  ObjectSecurityClassification create(
     in CosPropertyService::PropertySet property_set,
     in SecurityClassification sec_classification_level,
     in SecurityClassifiable sec_classified_object )
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface SecurityClassificationLevel :
  CosLifeCycleReference::ReferencesRole
```

```
{ };

interface SecurityClassifiedObject :
  CosLifeCycleReference::ReferencedByRole
{ };
```

3.4 PdmFoundation IDL

```
// PdmFoundation.idl

#ifndef  PDMFOUNDATION
#define  PDMFOUNDATION

#ifdef   SOM_COMPILE
#include <somobj.idl>                  // SOM COMPILE
#endif

#ifdef  ORBIX_COMPILE
#ifndef  IFR
#define  IFR
#include <ifr.idl>
#endif
#endif

#include <CosLifeCycle.idl>
#include <CosLifeCycleReference.idl>
#include <CosTime.idl>
#include <CosTransactions.idl>
#include <CosPropertyService.idl>
#include <CosNaming.idl>
#include <PdmResponsibility.idl>

module PdmFoundation
{

  interface Identification;
  interface IdentificationContext;

  struct Measurement
  {
    double quantity;
    string uom;
    boolean approximate;
  };

  struct NSPair {string name; string value;};
  typedef sequence<NSPair> IdentifierSeq;

  // Exceptions
  struct PdmPropertyValidationError
```

```
{
   CosPropertyService::PropertyName property_name;
   unsigned long error_code;
   string error_text;
};

typedef sequence<PdmPropertyValidationError>
   PdmPropertyValidationErrors;

exception PdmError
   {unsigned long error_code; string error_text;};
exception PermissionDenied
   {unsigned long error_code; string error_text;};
exception ValidationError
   {unsigned long error_code; string error_text;};
exception NotUnique
   {unsigned long error_code; string error_text;};
exception NotFound
   {unsigned long error_code; string error_text;};
exception InvalidProperties
{
   unsigned long error_code; string error_text;
   PdmPropertyValidationErrors validation_errors;
};
exception CardinalityExceeded
{
   unsigned long error_code; string error_text;
   string role_name;
};
exception IdentifierNotDefined
   {unsigned long error_code; string error_text;};
exception GenerateNotAvailable
   {unsigned long error_code; string error_text;};
exception AlreadyLocked
   {unsigned long error_code; string error_text;};
exception UnlockFailed
   {unsigned long error_code; string error_text;};
exception InvalidTransition
   {unsigned long error_code; string error_text;};

#define PDM_EXCEPTIONS \
   PdmFoundation::PdmError, \
   PdmFoundation::PermissionDenied, \
   PdmFoundation::ValidationError

#define ITEM_CREATE_EXCEPTIONS \
   PdmFoundation::PdmError, \
   PdmFoundation::PermissionDenied, \
   PdmFoundation::ValidationError, \
   PdmFoundation::InvalidProperties, \
   PdmFoundation::NotUnique
```

```
#define RELATIONSHIP_CREATE_EXCEPTIONS \
  PdmFoundation::PdmError, \
  PdmFoundation::PermissionDenied, \
  PdmFoundation::ValidationError, \
  PdmFoundation::InvalidProperties, \
  PdmFoundation::NotUnique, \
  PdmFoundation::CardinalityExceeded

// Entities

typedef CosNaming::NameComponent  IdentificationContextName;
typedef sequence <IdentificationContextName>
  IdentificationContextNames;

interface Identifiable
{
  string get_id(in IdentificationContext id_context)
    raises(IdentifierNotDefined, PDM_EXCEPTIONS);
  IdentifierSeq get_id_seq(in IdentificationContext id_context)
    raises(IdentifierNotDefined, PDM_EXCEPTIONS);
  IdentifierSeq bind(
      in CosPropertyService::PropertySet property_set,
      in IdentificationContext the_context)
    raises(RELATIONSHIP_CREATE_EXCEPTIONS);
  void change_id( in CosPropertyService::PropertySet property_set,
      in IdentificationContext the_context)
    raises(PDM_EXCEPTIONS, PdmFoundation::NotUnique);

};

interface IdentificationContext : CosLifeCycle::LifeCycleObject,
  CosTransactions::TransactionalObject, CosCompoundLifeCycle::Node
{
  attribute IdentificationContextName name;
  attribute string description;

  Identifiable find(in IdentifierSeq the_id)
    raises(NotFound, PDM_EXCEPTIONS);
  boolean verify_id(in IdentifierSeq the_id)
    raises(ValidationError, PDM_EXCEPTIONS);
IdentifierSeq generate_id(
    in CosPropertyService::PropertySet property_set,
    in Identifiable the_object)
  raises(GenerateNotAvailable, PDM_EXCEPTIONS);
};

typedef sequence <IdentificationContext> IdentificationContexts;

interface IdentificationContextFactory
{
```

```
    IdentificationContext create(
          in CosPropertyService::PropertySet property_set)
      raises(ITEM_CREATE_EXCEPTIONS);
    IdentificationContext find_id_context(
          in IdentificationContextName the_context_name)
      raises(NotFound, PDM_EXCEPTIONS);
    IdentificationContexts find_all_id_contexts()
      raises (PDM_EXCEPTIONS);
    IdentificationContexts find_id_contexts( in string identifiableType )
      raises(NotFound, PDM_EXCEPTIONS);
}

interface Lockable
{
    boolean is_locked();

    void lock(in PdmResponsibility::Actor lock_owner, in string reason)
      raises( AlreadyLocked, PDM_EXCEPTIONS);
    void unlock(in PdmResponsibility::Actor lock_owner)
      raises( UnlockFailed, PDM_EXCEPTIONS);
};

interface Manageable : Identifiable
{
    attribute TimeBase::UtcT created_date;
    attribute TimeBase::UtcT last_modified_date;
    attribute string short_description;
    attribute string long_description;
};

interface SecurityClassifiable
{ };

interface SecurityClassification : CosLifeCycle::LifeCycleObject,
    CosTransactions::TransactionalObject
{
    attribute string name;
    attribute string purpose;
    attribute TimeBase::UtcT valid_date;
};

interface SecurityClassificationFactory
{
    SecurityClassification create(
      in CosPropertyService::PropertySet property_set)
      raises(ITEM_CREATE_EXCEPTIONS);
    SecurityClassification find_sec_class(in string in_name)
      raises(NotFound, PDM_EXCEPTIONS);
};

interface Stateable
```

```
{
   readonly attribute string state;

   void change_state(in string in_state)
      raises( InvalidTransition, PDM_EXCEPTIONS );
};

// Relationships

// Identification Relationship
//  role : IdentifiableObjectRole
//    name : 'IdentifiablebjectRole'
//    entity : Identifiable
//    cardinality : 0..unbounded
//  role : IdentificationContextRole
//    name : 'IdentificationContextRole'
//    entity : IdentificationContext
//    cardinality : 0..unbounded

interface Identification : CosLifeCycleReference::Relationship
{
   readonly attribute IdentifierSeq id;


};

interface IdentifiableObjectRole :
   CosLifeCycleReference::ReferencesRole {};

interface IdentificationContextRole :
   CosLifeCycleReference::ReferencedByRole {};

// LockOwner Relationship
//  role : Locked
//    name : 'Locked'
//    entity : Lockable
//    cardinality : 0..1
//  role : LockBy
//    name : 'LockBy'
//    entity : PdmResponsibility::Actor
//    cardinality : 0..unbounded

interface LockOwner : CosLifeCycleReference::Relationship
{
   readonly attribute TimeBase::UtcT date_locked;
   readonly attribute string reason_locked;
};


interface Locked : CosLifeCycleReference::ReferencesRole { };
```

```
interface LockBy : CosLifeCycleReference::ReferencedByRole { };

// ObjectCreator Relationship
//  role : Created
//    name : 'Created'
//    entity : Manageable
//    cardinality : 0..1
//  role : Creator
//    name : 'Creator'
//    entity : PdmResponsibility::Actor
//    cardinality : 0..unbounded

interface ObjectCreator : CosLifeCycleReference::Relationship { };

interface ObjectCreatorFactory
{
   ObjectCreator create(
      in CosPropertyService::PropertySet property_set,
      in Manageable the_createe,
      in PdmResponsibility::Actor the_creator)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface Created : CosLifeCycleReference::ReferencesRole { };

interface Creator : CosLifeCycleReference::ReferencedByRole { };

// ObjectOwner Relationship
//  role : Owned
//    name : 'Owned'
//    entity : Manageable
//    cardinality : 0..1
//  role : Owner
//    name : 'Owner'
//    entity : PdmResponsibility::Party
//    cardinality : 0..unbounded

interface ObjectOwner : CosLifeCycleReference::Relationship { };

interface ObjectOwnerFactory
{
   ObjectOwner create(
      in CosPropertyService::PropertySet property_set,
      in Manageable the_ownee,
      in PdmResponsibility::Party the_owner)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface Owned : CosLifeCycleReference::ReferencesRole { };

interface Owner : CosLifeCycleReference::ReferencedByRole { };
```

```
  // ObjectSecurityClassification Relationship
  //  role : SecurityClassificationLevel
  //    name : 'SecurityClassificationLevel'
  //    entity : SecurityClassification
  //    cardinality : 0..unbounded
  //  role : SecurityClassifiedObject
  //    name : 'SecuirtyClassifiedObject'
  //    entity : SecurityClassifiable
  //    cardinality : 0..unbounded

  interface ObjectSecurityClassification :
    CosLifeCycleReference::Relationship
    {
      attribute PdmResponsibility::Person owner;
    };

  interface ObjectSecurityClassificationFactory
  {
    ObjectSecurityClassification create(
      in CosPropertyService::PropertySet property_set,
      in SecurityClassification sec_classification_level,
      in SecurityClassifiable sec_classified_object )
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
  };

  interface SecurityClassificationLevel :
    CosLifeCycleReference::ReferencesRole
    { };

  interface SecurityClassifiedObject :
    CosLifeCycleReference::ReferencedByRole
    { };

};

#endif
```

PdmFramework Module 4

Contents

This chapter contains the following sections.

Section Title	Page
“Overview”	4-1
“PdmFramework Model”	4-2
“PdmFramework Description”	4-4
“PdmFramework IDL”	4-15

4.1 Overview

The **PDMFramework** module provides a set of PDM-specific framework object and relationship classes.

This module uses multiple inheritance to collect elementary units of behavior provided by CORBAservices and the **PDMFoundation** module into high level PDM domain specific abstract objects (**ItemMaster**, **ItemRevision**, **ItemIteration**) that are used as the basis for extension in the other modules for specific applications.

4.2 PdmFramework Model

4.2.1 PdmFramework Entity Model

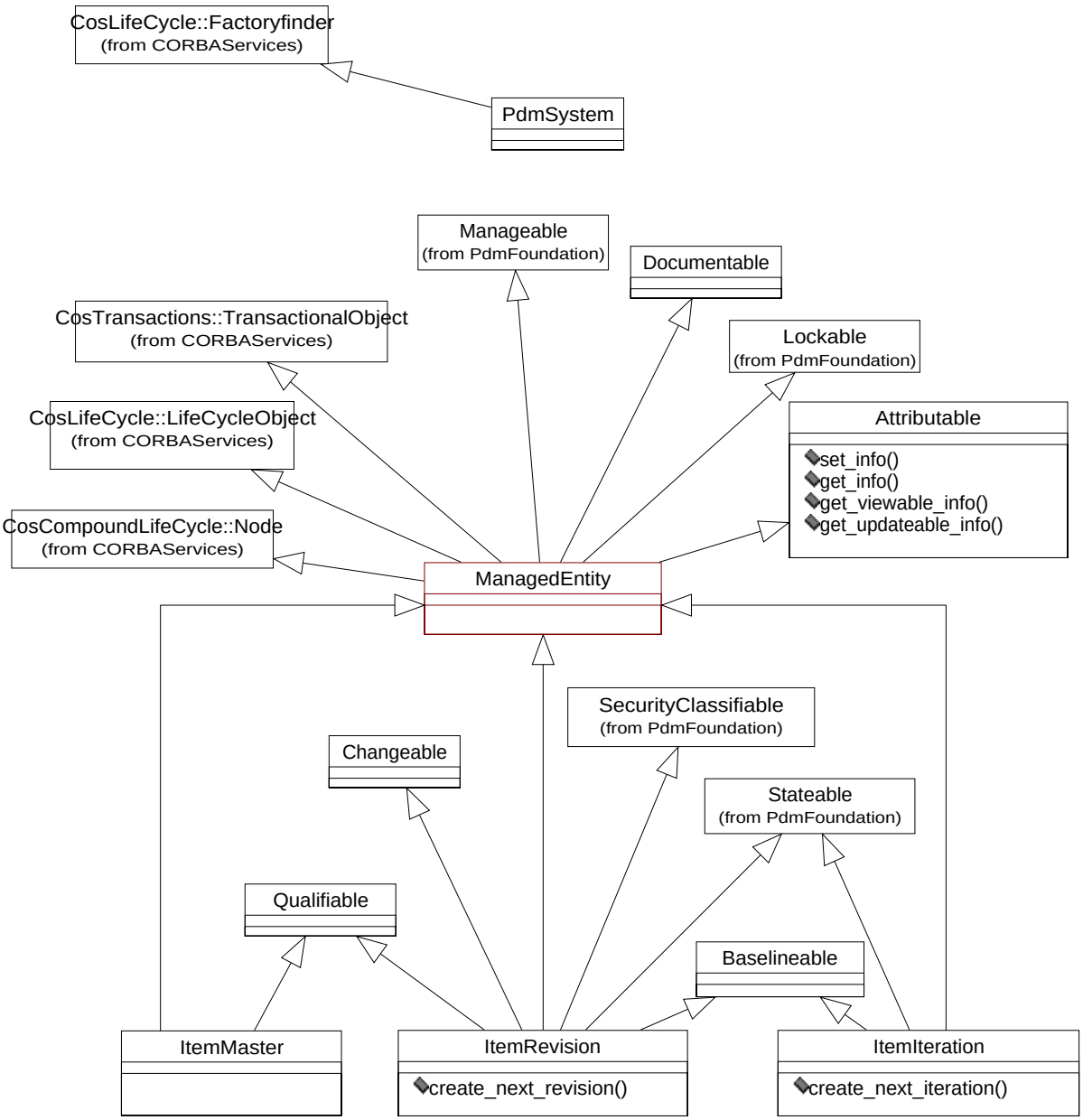

Figure 4-1 PdmFramework Entity Model Diagram

4.2.2 PdmFramework Relationship Model

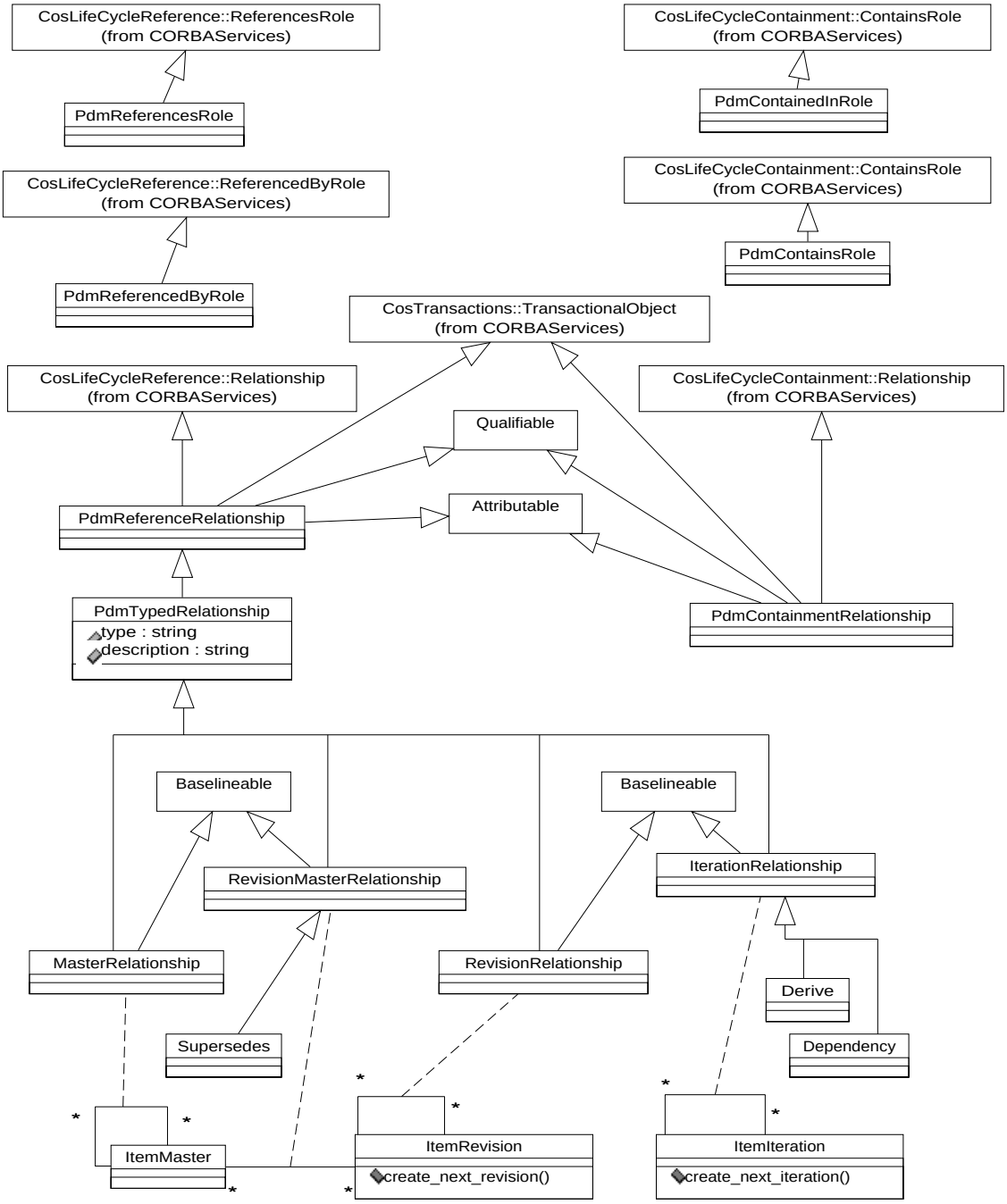

Figure 4-2 PdmFramework Relationship Model Diagram

4.3 PdmFramework Description

4.3.1 PdmSystem

```
interface PdmSystem : CosLifeCycle::FactoryFinder { };
```

The **PdmSystem** object represents the services of a single PDM system. As a **LifeCycle FactoryFinder**, it provides a destination location for the LifeCycle services copy and move operations. Furthermore, it can find all the other factories that are provided by the PDM Enablers server via the **find_factories** operation.

The **find_factories** operation takes a **CosNaming** style composite name (**NameComponent**) as a parameter to identify the factory. The composite name comprises an **id** field and a **kind** field. Following the naming conventions from Table 6-1 of omg/formal/98-12-09, interoperable clients and servers shall use the interface repository name of the factory interface for the **id** field and the string "factory interface" for the **kind** field. Furthermore, it is suggested that the **PdmSystem** support one factory of each type.

4.3.2 Attributable

The **Attributable** class is an abstract behavioral class that provides the ability to set and update attributes for an item.

```
interface Attributable
{
  void get_info(
       inout CosPropertyService::PropertySet property_set)
    raises (PDM_EXCEPTIONS, PdmFoundation::InvalidProperties);
  void set_info(
       in CosPropertyService::PropertySet property_set)
    raises (PDM_EXCEPTIONS, PdmFoundation::InvalidProperties);
  void get_viewable_info(
       out CosPropertyService::PropertySet property_set);
  void get_updatable_info(
       out CosPropertyService::PropertySet property_set)
    raises (PDM_EXCEPTIONS, PdmFoundation::InvalidProperties);
};
```

The operations of the **Attributable** interface use **CosPropertyService PropertySet** interfaces to specify the names of the attributes and their values.

This technique allows access to all legal attributes for the managed item in the PDM system, even those that are not exposed to clients through the IDL definitions. **PropertySet**s support the manipulation of attributes that are known to a customized PDM system but are not specifically exposed by IDL. And **PropertySet**s defined and created by the client support the manipulation of ad hoc or loose attributes that are not known to the PDM system's schema (if the PDM system supports ad hoc attributes).

Different types of **PropertySet**s can be established depending on the types of ManagedItems in the PDM system and the types of operations to be performed. For example, the set of attributes returned by a **get_info** operation are usually different from the set of attributes changed by a **set_info** operation; an Electrical Part item may have different updatable attributes than a Design Specification; and a CAD interface may update different attributes than an ERP interface. Different PropertySets can be defined for each of these purposes using the **CosPropertyDef** interface, if a **CosPropertyDef** service is available.

Clients that are oriented to performing specific business tasks should use the facilities of the **CosPropertyService** to define and manipulate specific **PropertySet**s tailored for their use. Generic clients may use the **get_viewable_properties** and **get_updatable_properties** operations to provide the client with **PropertySet**s containing all permitted attribute names and values.

get_info

The **get_info** operation accepts a **PropertySet**, which defines the names of the attributes that the client wishes to obtain. The operation returns the **PropertySet** with the current values of the attributes. If the **PropertySet** names an attribute that the user is not authorized to view, the PermissionDenied exception is raised. If the **PropertySet** contains the name of an attribute that is not recognized by the PDM system, an InvalidProperties exception is raised. In either case, even if an exception is raised, all recognized and permitted attributes are returned.

The implementation of the **get_info** operation is responsible for identifying the desired attributes, validating that the client may see them, and moving their values from the **Attributable** item to the **property_set**.

set_info

The **set_info** operation accepts a **PropertySet** with the names and values of the attributes that the client wishes to set. If the **PropertySet** names an attribute that the user is not authorized to update, the PermissionDenied exception is raised. If the **PropertySet** names an attribute that is not recognized by the PDM system, the InvalidProperties exception is raised.

The implementation of the **set_info** operation is responsible for obtaining the attributes and their values from the **property_set**, validating them, and changing their persistent values in the PDM system.

get_viewable_info

The **get_viewable_info** returns a **PropertySet** with the names and values of the attributes that the user is permitted to view.

get_updatable_info

The **get_updatable_info** operation returns a **PropertySet** with the names and values of the attributes that the user is permitted to update.

4.3.3 Baselineable

This behaviorable interface class allows the item to be recorded in a Baseline. It is more fully explained in the **PdmBaseline** module, Chapter 5.

```
interface Baselineable {};
```

4.3.4 Qualifiable

This **Qualifiable** interface class allows the item to take effect under certain conditions. It is more fully explained in the **PdmViews** module, Chapter 6.

```
interface Qualifiable {};
```

4.3.5 Changeable

The **Changeable** interface allows the item to be used in engineering change processes. A **Changeable** entity participates in relationships described in the **PdmChangeManagement** module, Chapter 10.

```
interface Changeable {};
```

4.3.6 Documentable

The **Documentable** interface allows documents to be attached to a **ManagedEntity**. This interface is explained further in the **PdmDocumentManagement** module, Chapter 7.

```
interface Documentable {};
```

4.3.7 ManagedEntity

The **ManagedEntity** is an abstract framework class that encapsulates a variety of attributes and behavior for items that are managed by the PDM System.

```
interface ManagedEntity :
  PdmFoundation::Manageable,
  PdmFoundation::Lockable,
  Attributable,
  CosLifeCycle::LifeCycleObject,
  CosCompoundLifeCycle::Node,
  CosTransactions::TransactionalObject,
  Documentable {};
```

4.3.8 ItemMaster

The **ItemMaster** is an abstract framework class that is specialized for particular functions in other enablers.

The **ItemMaster** represents an item managed by the PDM system throughout its existence regardless of formal revisions or informal changes. It encapsulates attributes and behaviors that do not change. Usually, the primary item identifier (part number, document id) is associated to the **ItemMaster** as an **IdentifiableItem**.

```
interface ItemMaster : ManagedEntity, Qualifiable {};
```

4.3.9 ItemRevision

The **ItemRevision** is an abstract Framework object that is specialized for particular functions in other enablers.

The **ItemRevision** represents a formal revision or change level of an item managed by the PDM system. It encapsulates attributes and behavior that changes as the item is formally revised.

Every **ItemRevision** is associated to a single corresponding **ItemMaster**. This association is not defined in this module, but is defined for specific applications in the other modules. As an **Identifiable** object, usually, a revision code and/or change letter is associated to the **ItemRevision**. The primary item identifier is derived through its associated **ItemMaster**.

The **create_next_revision** operation creates a new revision to follow the target revision. It is like the target revision, but with attributes as specified by the **property_set** parameter. None of the **ItemIterations** from the previous **ItemRevision** are copied forward. Specific Revision Factory interfaces allow the creation of the first revision for an **ItemMaster**.

```
interface ItemRevision :
  ManagedEntity,
  PdmFoundation::Stateable,
  PdmFoundation::SecurityClassifiable,
  Baselineable,
  Changeable,
  Qualifiable
{
  ItemRevision create_next_revision(
       in CosPropertyService::PropertySet property_set)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

4.3.10 ItemIteration

The **ItemIteration** is an abstract Framework object that is specialized for particular functions in other enablers.

Iterations represent individual aspects of the data that jointly defines a revision. Iterations may be updated independently, resulting in different numbers of iterations for the various aspects of the revisions.

The **ItemIteration** represents the smallest set of changes for an item that are separately managed in the PDM system. It encapsulates attributes and behavior that change as the item is modified by work-in-process. Typically, depending on implementation or site specific rules, an item undergoing active sequential and/or alternative change has more than one iteration. When a particular iteration is complete, proven, and released for the current active revision, the other iterations might be discarded.

Every **ItemIteration** is associated to a single corresponding **ItemRevision**. This association is not shown at the abstract level in this enabler diagram, but is shown for specific applications in the other enablers. As an Identifiable object, usually, an iteration number or timestamp is associated to the **ItemIteration**. The primary item identifier and revision / change level code is derived through its associated **ItemRevision** and **ItemMaster**.

The **create_next_revision** operation creates a new revision to follow the target revision. It is like the target revision, except with attributes as specified by the **property_set** parameter. Specific Revision Factory interfaces allow the creation of the first revision for an **ItemMaster**.

```
interface ItemIteration : ManagedEntity, Baselineable,
  PdmFoundation::Stateable
{
  ItemIteration create_next_iteration(
       in CosPropertyService::PropertySet property_set)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

4.3.11 PdmContainmentRelationship

The **PdmContainmentRelationship** is an abstract class that can be extended for specific uses in the other modules. It is transactional and has the behavior of the **CosLifeCycleContainment** relationship, including the behavior of processing related graphs of objects when a node is copied, moved, or removed. PDM Enablers interfaces that create new items based on an existing item have the behavior of a compound copy operation with respect to these relationships.

```
// PdmContainmentRelationship
//   role:  PdmContainsRole
//     name: 'PdmContainsRole'
//     entity:  ManagedEntity
//     cardinality:  0..unbounded
//   role: PdmContainedInRole
//     name:  'PdmContainedInRole'
//     entity: ManagedEntity
//     cardinality:  1..1

interface PdmContainmentRelationship :
  CosTransactions::TransactionalObject,
  CosLifeCycleContainment::Relationship,
```

```
  Qualifiable, Attributable {};

interface PdmContainsRole :
  CosLifeCycleContainment::ContainsRole {};

interface PdmContainedInRole:
  CosLifeCycleContainment::ContainedInRole {};
```

4.3.12 PdmReferenceRelationship

The **PdmReferenceRelationship** is an abstract class that can be extended for specific uses in the other modules. It is transactional and has the behavior of the **CosLifeCycleReference** relationship, including the behavior of processing related graphs of objects when a node is copied, moved, or removed. PDM Enablers interfaces that create new items based on an existing item have the behavior of a compound copy operation with respect to these relationships.

```
// PdmReferenceRelationship
//   role:  PdmReferencesRole
//     name:  'PdmReferencesRole'
//     entity:  ManagedEntity
//     cardinality:  0..unbounded
//   role: PdmReferencedByRole
//     name:  'PdmReferencedBy'
//     entity: ManagedEntity
//     cardinality:  0..unbounded

interface PdmReferenceRelationship :
  CosTransactions::TransactionalObject,
  CosLifeCycleReference::Relationship,
  Qualifiable, Attributable {};

interface PdmReferencesRole :
  CosLifeCycleReference::ReferencesRole{};

interface PdmReferencedByRole:
  CosLifeCycleReference::ReferencedByRole{};
```

4.3.13 PdmTypedRelationship

A **PdmTypedRelationship** is an abstract extendible relationship that can be used for many purposes. This relationship can be extended in IDL to provide specific behavior. In addition, through the use of the type and description attributes, relationships that inherit the **PdmTypedRelationship** interface can be used for different specific purposes without creating additional subclasses in IDL.

For example, if the PDM system at a site supports a relationship known as **AnalysisResultsRelationship** between a CAD document and a structural analysis results document, a client might be able to define the relationship by creating a **PdmTypedRelationship** with type=**AnalysisResultsRelationship**.

```
// PdmTypedRelationship
//   role:  PdmReferencesRole
//     name:  'PdmReferencesRole'
//     entity:  ManagedEntity
//     cardinality:  0..unbounded
//   role: PdmReferencedByRole
//     name:  'PdmReferencedByRole'
//     entity: ManagedEntity
//     cardinality:  0..unbounded

interface PdmTypedRelationship :
      PdmReferenceRelationship {
   readonly attribute string type;
   attribute string description;
};

interface PdmTypedRelationshipFactory
{
   PdmTypedRelationship create(
      in CosPropertyService::PropertySet property_set,
      in ManagedEntity references,
      in ManagedEntity referenced_by,
      in string relationship_type)
      raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};
```

4.3.14 MasterRelationship

A **MasterRelationship** can be used for any kind of predefined or ad-hoc relationship between two **ItemMasters**.

```
// MasterRelationship
//   role:  ReferencesMaster
//     name:  'ReferencesMaster'
//     entity:  ItemMaster
//     cardinality:  0..unbounded
//   role: ReferencedByMaster
//     name:  'ReferencedByMaster'
//     entity: ItemMaster
//     cardinality:  0..unbounded

interface MasterRelationship :
   Baselineable, PdmTypedRelationship {};

interface MasterRelationshipFactory
```

```
{
  MasterRelationship create(
    in CosPropertyService::PropertySet property_set,
    in ItemMaster references_master,
    in ItemMaster referenced_by_master)
    raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};
```

4.3.15 RevisionRelationship

A **RevisionRelationship** can be used for any kind of predefined or ad-hoc relationship between two **ItemRevisions**.

```
// RevisionRelationship
//   role: ReferencesRevision
//     name:  'ReferencesRevision'
//     entity:  ItemRevision
//     cardinality:  0..unbounded
//   role: ReferencedByRevision
//     name:  'ReferencedByRevision'
//     entity: ItemRevision
//     cardinality:  0..unbounded

interface RevisionRelationship :
  Baselineable, PdmTypedRelationship {};

interface RevisionRelationshipFactory
{
  RevisionRelationship create(
    in CosPropertyService::PropertySet property_set,
    in ItemRevision references_revision,
    in ItemRevision referenced_by_revision)
    raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ReferencesRevision : PdmReferencesRole{};

interface ReferencedByRevision : PdmReferencedByRole{};
```

4.3.16 IterationRelationship

An **IterationRelationship** can be used for any kind of predefined or ad-hoc relationship between two **ItemIterations**.

```
// IterationRelationship
//   role: ReferencesIteration
//     name:  'ReferencesIteration'
//     entity:  ItemIteration
//     cardinality:  0..unbounded
```

```
//   role: ReferencedByIteration
//     name:  'ReferencedByIteration'
//     entity: ItemIteration
//     cardinality:  0..unbounded

interface IterationRelationship :
   Baselineable, PdmTypedRelationship {};

interface IterationRelationshipFactory
{
   IterationRelationship create(
      in CosPropertyService::PropertySet property_set,
      in ItemIteration references_iteration,
      in ItemIteration referenced_by_iteration)
      raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};
```

4.3.17 Derive

The **Derive** relationship specifies that one **ItemIteration** was derived in whole or part from another **ItemIteration**. If the source item changes, someone may need to analyze the impact on the derivative item. The derivative item may need to be regenerated.

Three possible types of **Derive** relationships are:

1. Successor - the item iteration is a new version or revision of a previous item iteration.

2. Copied - the item iteration is a copy of another item iteration of the same type and has a new identity.

3. Translated - the item iteration originated by being translated or transformed from another item iteration of a different type.

An implementation, extension, or customization can further classify derive relationship types as an extension to the standard framework.

```
// Derive Relationship
//   role: Derivative
//     name:  'Derivative'
//     entity: ItemIteration
//     cardinality:  0..unbounded
//   role: DeriveSource
//     name:  'DeriveSource'
//     entity:  ItemIteration
//     cardinality:  0..unbounded

interface Derive : IterationRelationship {};

interface DeriveFactory
{
```

```
  Derive create(
    in CosPropertyService::PropertySet property_set,
    in ItemIteration derivative,
    in ItemIteration derive_source)
    raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface Derivative : PdmReferencedByRole{};

interface DeriveSource : ReferencesIteration{};
```

4.3.18 Dependency

A **Dependency** relationship specifies that one **ItemIteration** depends on another **ItemIteration** to be complete. If the independent item is changed, the dependent item is implicitly changed too. If the independent item is removed, the dependent item is incomplete or invalid.

Two possible types of dependency relationships are:

1. ad-hoc - specified manually by the user.

2. application - an application requires the dependency.

An implementation, extension, or customization can further classify dependency types as an extension to the standard framework.

```
// Dependency Relationship
//   role:  Dependent
//     name:  'Dependent'
//     entity:  ItemIteration
//     cardinality:  0..unbounded
//   role: Independent
//     name:  'Independent'
//     entity: ItemIteration
//     cardinality:  0..unbounded

interface Dependency : IterationRelationship{};

interface DependencyFactory
{
  Dependency create(
    in CosPropertyService::PropertySet property_set,
    in ItemIteration dependent,
    in ItemIteration independent)
    raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface Dependent : ReferencesIteration{};

interface Independent  : ReferencedByIteration{};
```

4.3.19 RevisionMasterRelationship

The **RevisionMaster** relationship is an abstract extendible relationship that allows an **ItemRevision** to reference or use **ItemMasters** of other items, without specifically defining which revision of the reference should be used. By using the concepts defined in the **PdmViews** module, a specific qualified relationship can be determined for a particular context.

```
// RevisionMaster Relationship
//   role: RevisionReferencesMaster
//     name:  'RevisionReferencesMaster'
//     entity:  ItemRevision
//     cardinality:  0..unbounded
//   role: MasterReferencedByRevision
//     name:  'MasterReferencedByRevision'
//     entity: ItemMaster
//     cardinality:  0..unbounded

interface RevisionMasterRelationship :
  Baselineable, PdmTypedRelationship {};

interface RevisionMasterRelationshipFactory
{
  RevisionMasterRelationship create(
        in CosPropertyService::PropertySet property_set,
        in ItemRevision referencing_revision,
        in ItemMaster referenced_master )
     raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface RevisionReferencesMaster : PdmReferencesRole{};

interface MasterReferencedByRevision : PdmReferencedByRole{};
```

4.3.20 Supersedes

The **Supersedes** relationship is used to define that a particular **ItemRevision** has been superseded by another item. The particular revision of the succession item is unspecified, but can be determined for a particular context by using the **PdmViews** module.

```
// Supersedes Relationship
//   role:  Superseded
//     name:  'Superseded'
//     entity:  ItemRevision
//     cardinality:  0..unbounded
//   role: Successor
//     name:  'Successor'
//     entity: ItemMaster
//     cardinality:  0..unbounded
```

```
interface Supersedes : RevisionMasterRelationship {};

interface SupersedesFactory
{
  Supersedes create(
    in CosPropertyService::PropertySet property_set,
    in ItemRevision superseded_item,
    in ItemMaster successor_item)
    raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface Superseded : RevisionReferencesMaster {};

interface Successor : MasterReferencedByRevision {};
```

4.4 PdmFramework IDL

```
// PdmFramework.idl

#ifndef  PDMFRAMEWORK
#define  PDMFRAMEWORK

#include <CosLifeCycle.idl>
#include <CosPropertyService.idl>
#include <CosTransactions.idl>
#include <CosCompoundLifeCycle.idl>
#include <CosLifeCycleContainment.idl>
#include <CosLifeCycleReference.idl>
#include <PdmFoundation.idl>

module PdmFramework
{

  // Exceptions

  #define PDM_EXCEPTIONS \
    PdmFoundation::PdmError, \
    PdmFoundation::PermissionDenied, \
    PdmFoundation::ValidationError

  #define ITEM_CREATE_EXCEPTIONS \
    PdmFoundation::PdmError, \
    PdmFoundation::PermissionDenied, \
    PdmFoundation::ValidationError, \
    PdmFoundation::InvalidProperties, \
    PdmFoundation::NotUnique

  #define RELATIONSHIP_CREATE_EXCEPTIONS \
    PdmFoundation::PdmError, \
```

```
    PdmFoundation::PermissionDenied, \
    PdmFoundation::ValidationError, \
    PdmFoundation::InvalidProperties, \
    PdmFoundation::NotUnique, \
    PdmFoundation::CardinalityExceeded

// Entities

interface Attributable
{
    void get_info(
            inout CosPropertyService::PropertySet property_set)
        raises (PDM_EXCEPTIONS, PdmFoundation::InvalidProperties);
    void set_info(
            in CosPropertyService::PropertySet property_set)
        raises (PDM_EXCEPTIONS, PdmFoundation::InvalidProperties);
    void get_viewable_info(
            out CosPropertyService::PropertySet property_set);
    void get_updatable_info(
            out CosPropertyService::PropertySet property_set)
        raises (PDM_EXCEPTIONS, PdmFoundation::InvalidProperties);
};

interface Documentable {};

interface ManagedEntity :
        PdmFoundation::Manageable,
        PdmFoundation::Lockable,
        Attributable,
        CosLifeCycle::LifeCycleObject,
        CosCompoundLifeCycle::Node,
        CosTransactions::TransactionalObject,
        Documentable { };

interface Qualifiable {};

interface Baselineable {};

interface Changeable {};

interface ItemIteration : ManagedEntity, Baselineable,
        PdmFoundation::Stateable
{
    ItemIteration create_next_iteration(
            in CosPropertyService::PropertySet property_set)
        raises (ITEM_CREATE_EXCEPTIONS);
};

interface ItemMaster : ManagedEntity, Qualifiable {};
```

```
interface ItemRevision :
      ManagedEntity,
      PdmFoundation::Stateable,
      PdmFoundation::SecurityClassifiable,
      Baselineable,
      Changeable,
      Qualifiable
{
   ItemRevision create_next_revision(
         in CosPropertyService::PropertySet property_set)
      raises (ITEM_CREATE_EXCEPTIONS);
};

interface PdmSystem : CosLifeCycle::FactoryFinder {};

// Relationships

// PdmContainmentRelationship
//   role:  PdmContainsRole
//     name: 'PdmContainsRole'
//     entity:  ManagedEntity
//     cardinality:  0..unbounded
//   role: PdmContainedInRole
//     name:  'PdmContainedInRole'
//     entity: ManagedEntity
//     cardinality:  1..1

interface PdmContainmentRelationship :
      CosTransactions::TransactionalObject,
      CosLifeCycleContainment::Relationship,
      Qualifiable, Attributable {};

interface PdmContainsRole :
      CosLifeCycleContainment::ContainsRole {};

interface PdmContainedInRole:
      CosLifeCycleContainment::ContainedInRole {};

// PdmReferenceRelationship
//   role:  PdmReferencesRole
//     name:  'PdmReferencesRole'
//     entity:  ManagedEntity
//     cardinality:  0..unbounded
//   role: PdmReferencedByRole
//     name:  'PdmReferencedBy'
//     entity: ManagedEntity
//     cardinality:  0..unbounded

interface PdmReferenceRelationship :
      CosTransactions::TransactionalObject,
      CosLifeCycleReference::Relationship,
```

```
    Qualifiable, Attributable {};

interface PdmReferencesRole :
    CosLifeCycleReference::ReferencesRole{};

interface PdmReferencedByRole:
    CosLifeCycleReference::ReferencedByRole{};

// PdmTypedRelationship
//   role:  PdmReferencesRole
//     name:  'PdmReferencesRole'
//     entity:  ManagedEntity
//     cardinality:  0..unbounded
//   role: PdmReferencedByRole
//     name:  'PdmReferencedByRole'
//     entity: ManagedEntity
//     cardinality:  0..unbounded

interface PdmTypedRelationship :
    PdmReferenceRelationship {
  readonly attribute string type;
  attribute string description;
};

interface PdmTypedRelationshipFactory
{
  PdmTypedRelationship create(
    in CosPropertyService::PropertySet property_set,
    in ManagedEntity references,
    in ManagedEntity referenced_by,
    in string relationship_type)
    raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

// MasterRelationship
//   role:  ReferencesMaster
//     name:  'ReferencesMaster'
//     entity:  ItemMaster
//     cardinality:  0..unbounded
//   role: ReferencedByMaster
//     name:  'ReferencedByMaster'
//     entity: ItemMaster
//     cardinality:  0..unbounded

interface MasterRelationship :
    Baselineable, PdmTypedRelationship {};

interface MasterRelationshipFactory
{
  MasterRelationship create(
    in CosPropertyService::PropertySet property_set,
```

```
        in ItemMaster references_master,
        in ItemMaster referenced_by_master)
        raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ReferencesMaster : PdmReferencesRole{};

interface ReferencedByMaster : PdmReferencedByRole{};

// RevisionRelationship
//   role: ReferencesRevision
//     name:  'ReferencesRevision'
//     entity:  ItemRevision
//     cardinality:  0..unbounded
//   role: ReferencedByRevision
//     name:  'ReferencedByRevision'
//     entity: ItemRevision
//     cardinality:  0..unbounded

interface RevisionRelationship :
      Baselineable, PdmTypedRelationship {};

interface RevisionRelationshipFactory
{
   RevisionRelationship create(
      in CosPropertyService::PropertySet property_set,
      in ItemRevision references_revision,
      in ItemRevision referenced_by_revision)
      raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ReferencesRevision : PdmReferencesRole{};

interface ReferencedByRevision : PdmReferencedByRole{};

// IterationRelationship
//   role: ReferencesIteration
//     name:  'ReferencesIteration'
//     entity:  ItemIteration
//     cardinality:  0..unbounded
//   role: ReferencedByIteration
//     name:  'ReferencedByIteration'
//     entity: ItemIteration
//     cardinality:  0..unbounded

interface IterationRelationship :
      Baselineable, PdmTypedRelationship {};

interface IterationRelationshipFactory
{
   IterationRelationship create(
```

```
      in CosPropertyService::PropertySet property_set,
      in ItemIteration references_iteration,
      in ItemIteration referenced_by_iteration)
      raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ReferencesIteration : PdmReferencesRole{};

interface ReferencedByIteration : PdmReferencedByRole{};

// Derive Relationship
//   role: Derivative
//     name:  'Derivative'
//     entity: ItemIteration
//     cardinality:  0..unbounded
//   role: DeriveSource
//     name:  'DeriveSource'
//     entity:  ItemIteration
//     cardinality:  0..unbounded

interface Derive : IterationRelationship {};

interface DeriveFactory
{
   Derive create(
      in CosPropertyService::PropertySet property_set,
      in ItemIteration derivative,
      in ItemIteration derive_source)
      raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface Derivative : PdmReferencedByRole{};

interface DeriveSource : ReferencesIteration{};

// Dependency Relationship
//   role:  Dependent
//     name:  'Dependent'
//     entity:  ItemIteration
//     cardinality:  0..unbounded
//   role: Independent
//     name:  'Independent'
//     entity: ItemIteration
//     cardinality:  0..unbounded

interface Dependency : IterationRelationship{};

interface DependencyFactory
{
   Dependency create(
      in CosPropertyService::PropertySet property_set,
```

```
        in ItemIteration dependent,
        in ItemIteration independent)
        raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface Dependent : ReferencesIteration{};

interface Independent  : ReferencedByIteration{};

// RevisionMaster Relationship
//   role: RevisionReferencesMaster
//     name:  'RevisionReferencesMaster'
//     entity:  ItemRevision
//     cardinality:  0..unbounded
//   role: MasterReferencedByRevision
//     name:  'MasterReferencedByRevision'
//     entity: ItemMaster
//     cardinality:  0..unbounded

interface RevisionMasterRelationship :
      Baselineable, PdmTypedRelationship {};

interface RevisionMasterRelationshipFactory
{
   RevisionMasterRelationship create(
         in CosPropertyService::PropertySet property_set,
         in ItemRevision referencing_revision,
         in ItemMaster referenced_master )
      raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface RevisionReferencesMaster : PdmReferencesRole{};

interface MasterReferencedByRevision : PdmReferencedByRole{};

// Supersedes Relationship
//   role:  Superseded
//     name:  'Superseded'
//     entity:  ItemRevision
//     cardinality:  0..unbounded
//   role: Successor
//     name:  'Successor'
//     entity: ItemMaster
//     cardinality:  0..unbounded

interface Supersedes : RevisionMasterRelationship {};

interface SupersedesFactory
{
   Supersedes create(
      in CosPropertyService::PropertySet property_set,
```

```
            in ItemRevision superseded_item,
            in ItemMaster successor_item)
            raises (RELATIONSHIP_CREATE_EXCEPTIONS);
        };

        interface Superseded : RevisionReferencesMaster {};

        interface Successor : MasterReferencedByRevision {};

    };

    #endif
```

PdmBaseline Module 5

Contents

This chapter contains the following sections.

Section Title	Page
“Overview”	5-1
“PdmBaseline Model”	5-2
“PdmBaseline Description”	5-2
“PdmBaseline IDL”	5-6

5.1 Overview

A Baseline is a collection of items and the relationships between the items that is established to ensure their continued existence and to enable their configuration to be reconstructed and audited. Baselines are created for a particular purpose and they are constructed by automatic or manual means over time. Any Baselineable object may be added as a member of a baseline.

A Baseline is a captured configuration of items that is used for any business purpose. Those business purposes are not defined by this specification. Appropriate business or implementation specific rules should enforce appropriate constraints. For example: After an object is added to a baseline, the object should not be deleted. After a baseline is completed and has reached an appropriate state, it may be frozen to indicate that it can no longer be changed and is protected. But it should not be frozen until each of its members are also frozen.

Baselines may be used for many different purposes. They can be used to capture a configuration for audit, for later possible reconstruction, for change analysis, or simply to be able to group a set of data for a temporary purpose. Some Baselines may be formal, contractual or even legal requirements. Others may be strictly informal and temporary.

To create a baseline, a client would follow the following general procedure: Create a **BaselineMaster** and the first **BaselineRevision** and **BaselineIteration**. Identify each of the **Baselineable** items and relationships that are to be part of the baseline, and add them to the baseline by creating Baselined relationships between the BaselineIteration and the **Baselineable** object. As each is added, non-standard business-specific or implementation-specific consistency rules may automatically cause other objects to be added as well.

5.2 PdmBaseline Model

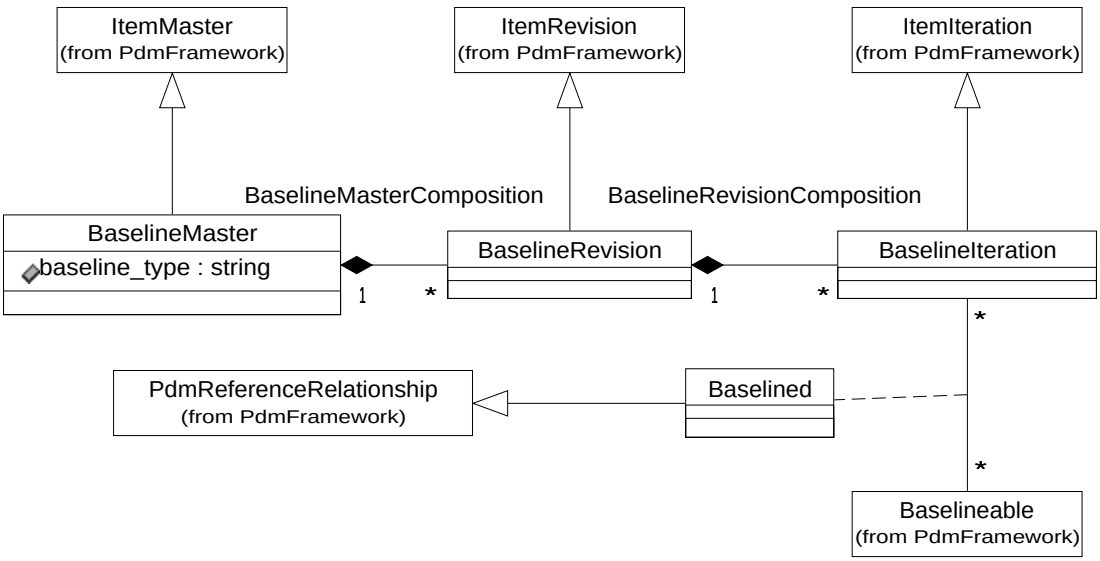

Figure 5-1 PdmBaseline Model Diagram

5.3 PdmBaseline Description

5.3.1 Baselineable

Baselineable is a behavioral interface class inherited by any object or relationship that can be placed into a baseline. It is defined in the **PdmFramework** module (see Chapter 4).

Any **ItemIteration** and **IterationRelationship** is **Baselineable**, to capture the specific iteration that is part of the Baseline.

Any **ItemRevision** and **RevisionRelationship** is **Baselineable**, to capture the formal revision or change level of an item that is part of the baseline.

In the View enabler, to capture the fact that a particular Qualification is associated to a Qualified Item for the purposes of a Baseline, the Qualification and the Qualifies relationship (between a **QualifiedItem** and a **Qualification**) are **Baselineable**.

Several other types of relationships and items are not formally **Baselineable**, but may be captured implicitly as part of a baseline because they make up the definition of a **Baselineable** object according to implementation or business requirements not specified here. For example, in the **PdmProductStructureDefinition** module, all Usage relationships for a **PartStructureIteration** should probably be implicitly captured when the **PartStructureIteration** is added to the Baseline.

5.3.2 *BaselineMaster*

The **BaselineMaster** identifies the baseline and represents the attributes and behavior that are not changed as the baseline undergoes formal revision and informal change.

```
interface BaselineMaster : PdmFramework::ItemMaster
{
  attribute string baseline_type;
};

interface BaselineMasterFactory
{
  BaselineMaster create(
       in CosPropertyService::PropertySet property_set)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

5.3.3 *BaselineRevision*

A **BaselineRevision** represents a formal revision of a **Baseline**. It encapsulates the attributes and behavior that change as a baseline is formally revised.

Each **BaselineRevision** is part of a single **BaselineMaster**.

```
interface BaselineRevision : PdmFramework::ItemRevision {};

interface BaselineRevisionFactory
{
  BaselineRevision create(
       in CosPropertyService::PropertySet property_set,
       in BaselineMaster baseline_master)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

5.3.4 BaselineIteration

A **BaselineIteration** encapsulates the attributes and behavior that change as a baseline is informally modified. It is associated to the particular **Baselineable** items that are protected by the Baseline.

Each **BaselineIteration** is part of a single **BaselineRevision**.

```
interface BaselineIteration : PdmFramework::ItemIteration {};

interface BaselineIterationFactory
{
  BaselineIteration create(
        in CosPropertyService::PropertySet property_set,
        in BaselineRevision baseline_revision)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

5.3.5 BaselineMasterComposition

The **BaselineMasterComposition** relationship relates a **BaselineMaster** to its **BaselineRevisions**.

```
// BaselineMasterComposition relationship
//   role: BaselineMasterForRevisions
//     name : 'BaselineMasterForRevisions'
//     entity: BaselineMaster
//     cardinality: 0..unbounded
//   role: BaselineRevisionForMaster
//     name: 'BaselineRevisionForMaster'
//     entity: BaselineRevision
//     cardinality: 1..1

interface BaselineMasterComposition :
  PdmFramework::PdmContainmentRelationship { };

interface BaselineMasterForRevisions :
  PdmFramework::PdmContainsRole { };

interface BaselineRevisionForMaster :
  PdmFramework::PdmContainedInRole { };
```

5.3.6 BaselineRevisionComposition

A **BaselineRevisionComposition** relationship relates a **BaselineRevision** to its **BaselineIterations**.

```
// BaselineRevisionComposition relationship
//   role: BaselineRevisionForIterations
//     name: 'BaselineRevisionForIterations'
```

```
//     entity: BaselineRevision
//     cardinality: 0..unbounded
//   role: BaselineIterationForRevision
//     name: 'BaselineIterationForRevision'
//     entity: BaselineIteration
//     cardinality: 1..1

interface BaselineRevisionComposition :
  PdmFramework::PdmContainmentRelationship {};

interface BaselineRevisionForIterations :
  PdmFramework::PdmContainsRole { };

interface BaselineIterationForRevision :
  PdmFramework::PdmContainedInRole { };
```

5.3.7 Baselined

The **Baselined** Relationship relates a **BaselineIteration** to the **Baselinable** items that are part of the baseline.

```
// Baselined relationship
//   role: BaseliningIteration
//     name: 'BaseliningIteration'
//     entity: BaselineIteration
//     cardinality: 0..unbounded
//   role: BaselinedItem
//     name: 'BaselinedItem'
//     entity: Baselinable
//     cardinality: 0..unbounded

interface Baselined :
  PdmFramework::PdmReferenceRelationship {};

interface BaselinedFactory
{
  Baselined create(
        in CosPropertyService::PropertySet property_set,
        in BaselineIteration baselining_iteration,
        in PdmFramework::Baselineable baselined_item)
      raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface BaseliningIteration :
     PdmFramework::PdmReferencesRole { };

interface BaselinedItem :
     PdmFramework::PdmReferencedByRole { };
```

5.4 PdmBaseline IDL

```
// PdmBaseline.idl

#ifndef  PDMBASELINE
#define  PDMBASELINE

#include <CosPropertyService.idl>
#include <PdmFramework.idl>

module PdmBaseline
{

interface BaselineRevision;

// Exceptions

#define PDM_EXCEPTIONS \
   PdmFoundation::PdmError, \
   PdmFoundation::PermissionDenied, \
   PdmFoundation::ValidationError

#define ITEM_CREATE_EXCEPTIONS \
   PdmFoundation::PdmError, \
   PdmFoundation::PermissionDenied, \
   PdmFoundation::ValidationError, \
   PdmFoundation::InvalidProperties, \
   PdmFoundation::NotUnique

#define RELATIONSHIP_CREATE_EXCEPTIONS \
   PdmFoundation::PdmError, \
   PdmFoundation::PermissionDenied, \
   PdmFoundation::ValidationError, \
   PdmFoundation::InvalidProperties, \
   PdmFoundation::NotUnique, \
   PdmFoundation::CardinalityExceeded

// Entities

interface BaselineIteration : PdmFramework::ItemIteration {};

interface BaselineIterationFactory
{
   BaselineIteration create(
         in CosPropertyService::PropertySet property_set,
         in BaselineRevision baseline_revision)
      raises (ITEM_CREATE_EXCEPTIONS);
};

interface BaselineMaster : PdmFramework::ItemMaster
```

```
{
  attribute string baseline_type;
};

interface BaselineMasterFactory
{
  BaselineMaster create(
        in CosPropertyService::PropertySet property_set)
    raises (ITEM_CREATE_EXCEPTIONS);
};

interface BaselineRevision : PdmFramework::ItemRevision {};

interface BaselineRevisionFactory
{
  BaselineRevision create(
        in CosPropertyService::PropertySet property_set,
        in BaselineMaster baseline_master)
    raises (ITEM_CREATE_EXCEPTIONS);
};

// Relationships

// Baselined relationship
//   role: BaseliningIteration
//     name: 'BaseliningIteration'
//     entity: BaselineIteration
//     cardinality: 0..unbounded
//   role: BaselinedItem
//     name: 'BaselinedItem'
//     entity: Baselinable
//     cardinality: 0..unbounded

interface Baselined :
     PdmFramework::PdmReferenceRelationship {};

interface BaselinedFactory
{
  Baselined create(
        in CosPropertyService::PropertySet property_set,
        in BaselineIteration baselining_iteration,
        in PdmFramework::Baselineable baselined_item)
    raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface BaseliningIteration :
     PdmFramework::PdmReferencesRole { };

interface BaselinedItem :
     PdmFramework::PdmReferencedByRole { };
```

```
// BaselineMasterComposition relationship
//   role: BaselineMasterForRevisions
//     name : 'BaselineMasterForRevisions'
//     entity: BaselineMaster
//     cardinality: 0..unbounded
//   role: BaselineRevisionForMaster
//     name: 'BaselineRevisionForMaster'
//     entity: BaselineRevision
//     cardinality: 1..1

interface BaselineMasterComposition :
     PdmFramework::PdmContainmentRelationship { };

interface BaselineMasterForRevisions :
     PdmFramework::PdmContainsRole { };

interface BaselineRevisionForMaster :
     PdmFramework::PdmContainedInRole { };

// BaselineRevisionComposition relationship
//   role: BaselineRevisionForIterations
//     name: 'BaselineRevisionForIterations'
//     entity: BaselineRevision
//     cardinality: 0..unbounded
//   role: BaselineIterationForRevision
//     name: 'BaselineIterationForRevision'
//     entity: BaselineIteration
//     cardinality: 1..1

interface BaselineRevisionComposition :
     PdmFramework::PdmContainmentRelationship {};

interface BaselineRevisionForIterations :
     PdmFramework::PdmContainsRole { };

interface BaselineIterationForRevision :
     PdmFramework::PdmContainedInRole { };

};

#endif
```

PdmViews Module 6

Contents

This chapter contains the following sections.

Section Title	Page
“Overview”	6-1
“PdmViews Model”	6-2
“PdmViews Description”	6-2
“PdmViews IDL”	6-10

6.1 Overview

The **PdmView** module provides a framework and explicit classes for indicating that items and relationships apply or qualify in particular contexts.

6.2 PdmViews Model

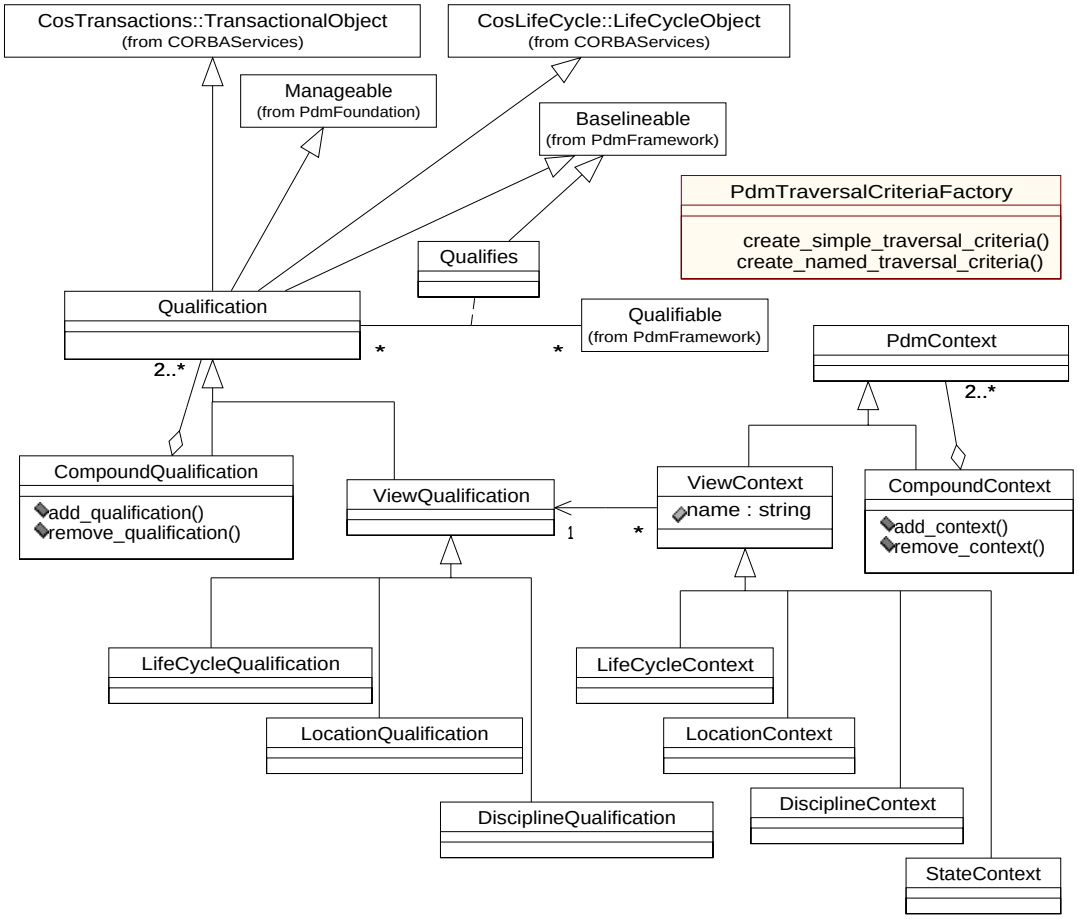

Figure 6-1 PdmViews Model Diagram

6.3 PdmViews Description

6.3.1 Qualification

A **Qualification** object indicates whether a relationship or object is applicable (or qualifies) under some set of constraints or for some purpose. This is an abstract object intended to be specialized by specific types of **Qualifications**. This module defines several types of **Qualifications**.

interface Qualification :
CosTransactions::TransactionalObject,

```
  CosLifeCycle::LifeCycleObject,
  PdmFramework::Baselineable,
  PdmFoundation::Manageable {};
```

6.3.2 Qualifiable

The **Qualifiable** interface introduces the concept that objects and relationships can be associated to qualifications. A **Qualifiable** object is visible in a **PdmContext** only when one of its associated qualifications matches the **PdmContext**.

In the PDM Enablers modules, several objects inherit the **Qualifiable** interface and thus participate in the View mechanisms.

The definition of the **Qualifiable** interface is found in the **PdmFramework** module (see Chapter 4).

6.3.3 Qualifies Relationship

Qualifies relationships associate **Qualifications** to the **Qualifiable** items to which they apply.

```
// Qualifies Relationship
//   role:  Qualified
//     name:  'Qualified'
//     entity:  Qualifiable
//     cardinality:  0..unbounded
//   role: Qualifier
//     name:  'Qualifier'
//     entity: Qualification
//     cardinality:  0..unbounded

interface Qualifies :
  PdmFramework::PdmReferenceRelationship,
  PdmFramework::Baselineable {};

interface QualifiesFactory
{
Qualifies create(
      in CosPropertyService::PropertySet property_set,
      in PdmFramework::Qualifiable qualified,
      in Qualification qualifier)
  raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface Qualified : PdmFramework::PdmReferencesRole{};

interface Qualifier : PdmFramework::PdmReferencedByRole{};
```

6.3.4 PdmContext

A **PdmContext** is an object that is established by the client and used to specify that a particular constraint, condition, or purpose holds for operations or relationship navigation.

A **PdmContext** is used to help create a **CosGraphs::TraversalCriteria**, which can in turn be used to create **CosGraphs::Traversal** objects, which navigate relationships. Only **Qualifiable** items with no **Qualification** or with at least one **Qualification** that matches the **PdmContext** are returned during the traversal. **Qualifiable** items without a matching **Qualification** are ignored.

The **PdmContext** is an abstract object intended to be specialized to create different types of **PdmContexts**.

```
interface PdmContext {};
```

6.3.5 PdmTraversalCriteriaFactory

The **PdmTraversalCriteriaFactory** creates a **CosGraphs::TraversalCriteria** that can be used to navigate relationship graphs in a PDM Enablers system.

```
exception UnsupportedTraversalName
{
  unsigned long error_code;
  string error_text;
};

interface PdmTraversalCriteriaFactory
{
  CosGraphs::TraversalCriteria create_simple_traversal_criteria(
    in PdmContext pdm_context, in CORBA::InterfaceDef role_type);
  CosGraphs::TraversalCriteria create_named_traversal_criteria(
    in PdmContext pdm_context, in string traversal_name)
    raises (UnsupportedTraversalName);
};
```

create_simple_traversal_criteria

The **create_simple_traversal_criteria** operation creates a simple **CosGraphs::TraversalCriteria** that facilitates navigating from a Node through a single Role, one level deep. The Role type and desired **PdmContext** are specified as parameters to the operation.

create_named_traversal_criteria

The **create_named_traversal_criteria** operation creates a **CosGraphs::TraversalCriteria** with more functionality. A Traversal may navigate and return **CosGraphs::Edges** for many Roles, and may continue to many levels of the relationship graph. This allows the implementation of a wide range of functionality to meet business and performance goals.

The names of specific traversal criteria and their functionality are not standardized by the PDM Enablers specification at this time, and must be documented by individual PDM Enablers service providers.

6.3.6 ViewQualification

A **ViewQualification** object is a subtype of **Qualification** that indicates something is applicable in a particular View.

Separate Views can be established for any purpose where a difference in part structure, revision, or other aspect is desired. Typically, Views are organized for separate disciplines (design, manufacturing), part life cycle (as-designed, as-planned), and/or locations (plant 1, plant 2). Separate subtypes are established for these purposes, and others may be constructed as extensions.

```
interface ViewQualification : Qualification
{
  attribute string name;
};

interface ViewQualificationFactory
{
  ViewQualification create(
    in CosPropertyService::PropertySet property_set)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

6.3.7 ViewContext

A **ViewContext** is an abstract object that indicates the view that the client wants to use to determine Qualified items.

```
interface ViewContext : PdmContext
{
  attribute string name;
  attribute ViewQualification view_qualification;
};
```

6.3.8 LifeCycleQualification

A **LifeCycleQualification** object expresses that an item is applicable to a particular life cycle stage, such as “as-designed” or “as-planned.” See ISO10303-44, “life cycle stage.”

```
interface LifeCycleQualification : ViewQualification {};

interface LifeCycleQualificationFactory
{
  LifeCycleQualification create(
```

```
    in CosPropertyService::PropertySet property_set)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

6.3.9 LifeCycleContext

A **LifeCycleContext** object indicates that the client is interested in items that qualify for a particular life cycle stage.

```
interface LifeCycleContext : ViewContext {};

interface LifeCycleContextFactory
{
  LifeCycleContext create(in string name)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

6.3.10 LocationQualification

A **LocationQualification** object expresses that an item is applicable to a particular location.

```
interface LocationQualification : ViewQualification {};

interface LocationQualificationFactory
{
  LocationQualification create(
    in CosPropertyService::PropertySet property_set)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

6.3.11 LocationContext

A **LocationContext** object indicates that the client is interested in items that qualify for a particular location.

```
interface LocationContext : ViewContext {};

interface LocationContextFactory
{
  LocationContext create( in string name)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

6.3.12 DisciplineQualification

A **DisciplineQualification** object expresses that an item is applicable to a particular discipline, such as “Electrical” or “Mechanical.” See ISO10303-41 “discipline_type.”

```
interface DisciplineQualification : ViewQualification {};

interface DisciplineQualificationFactory
{
  DisciplineQualification create(
    in CosPropertyService::PropertySet property_set)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

6.3.13 DisciplineContext

A **DisciplineContext** object indicates that a client is interested in items that qualify for a particular discipline.

```
interface DisciplineContext : ViewContext {};

interface DisciplineContextFactory
{
  DisciplineContext create( in string name)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

6.3.14 CompoundQualification

A **CompoundQualification** object is a collection of two or more Qualifications, used to express that an item or relationship qualified if both the component qualifications hold. For example, a Compound Qualification may be constructed to express that an item has a lifecycle of “as-designed” for the “MechanicalDesign” discipline.

```
interface CompoundQualification : Qualification
{
  void add_qualification( in Qualification the_qualification );
  void remove_qualification( in Qualification the_qualification );
};

interface CompoundQualificationFactory
{
  CompoundQualification create(
    in CosPropertyService::PropertySet property_set)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

6.3.15 CompoundContext

A **CompoundContext** object is a collection of two or more **PdmContexts**, used to express that the client is interested in items that are qualified for more than one context. For example, a Compound Context may be constructed to express that the client is interested in items with a lifecycle of “as-designed” for the “MechanicalDesign” discipline.

```
interface CompoundContext : PdmContext
{
  void add_context( in PdmContext the_context )
    raises (ITEM_CREATE_EXCEPTIONS);
  void remove_context( in PdmContext the_context )
    raises (ITEM_CREATE_EXCEPTIONS);
};

interface CompoundContextFactory
{
  CompoundContext create()
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

6.3.16 StateContext

A **StateContext** object indicates the state the client wants to use to determine qualified items. The indicated state or a more advanced state must appear in the state attribute of the related Stateable item, or the item is not used.

```
interface StateContext : ViewContext {};

interface StateContextFactory
{
  StateContext create( in string name )
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

6.3.17 Using Qualifications and PdmContexts

When using contexts to operate on **Qualifiable** items, the following default rules apply. For other particular requirements, **Qualifications** and **PdmContexts** may be extended to meet other behaviors.

Using Simple Qualifications and PdmContexts

A **Qualifiable** item is applicable under a **PdmContext** if the item is associated by a Qualifies relationship to a Qualification that matches the **PdmContext**.

If the Qualifiable item is...	Then...
not associated to any Qualification	it is applicable under all **PdmContext**s.

If the Qualifiable item is...	Then...
not associated to any Qualification of a particular type	it is applicable under any **PdmContext** of that corresponding type.
associated to one or more Qualifications of a particular type	it is applicable under a simple **PdmContext** of the corresponding type, which matches one of the associated Qualifications.
associated to one or more Qualifications of a particular type	it is not applicable under a **PdmContext**, which does not match any of the associated Qualifications.

Using Compound Qualifications and PdmContexts

CompoundQualification and **CompoundContext** have more complicated rules.

- A **CompoundQualification** matches a simple **PdmContext** if the **PdmContext** is matched by any of the components of the **CompoundQualification**.
- A **CompoundQualification** with components Q1 and Q2 matches a **CompoundContext** if the **CompoundContext** has a component that is matched by Q1 and has a component that is matched by Q2. This is not equivalent to having two separate simple Qualifications Q1 and Q2.
- The Qualification objects that compose a **CompoundQualification** should be of different types. The behavior of a **CompoundQualification** composed of Qualification objects of the same type is not defined.
- The **PdmContext** objects that compose a **CompoundContext** should be of different types. The behavior of a **CompoundContext** composed of **PdmContext** objects of the same type is not defined.
- The behavior of a **CompoundQualification** that contains another **CompoundQualification** is not defined.
- The behavior of a **CompoundContext** that contains another **CompoundContext** is not defined.

The following is an example of a **CompoundQualification**:

A **CompoundQualification** is composed of a **LocationQualification** of “Detroit” and a **DisciplineQualification** of “Mechanical.” A **PartMaster** is associated to this **CompoundQualification** (with a Qualifies relationship). The **PartMaster** is applicable or not, dependent on the **PdmContext** specified by a client.

- The **PartMaster** is applicable under a simple **LocationContext** of “Detroit.”
- The **PartMaster** is applicable under a simple **DisciplineContext** of “Mechanical.”
- The **PartMaster** is not applicable under a simple **LocationContext** of “Cleveland.”
- The **PartMaster** is not applicable under a simple **DisciplineContext** of “Electrical.”

- The **PartMaster** is applicable under a **CompoundContext** composed of a **LocationContext** of “Detroit” and a **DisciplineContext** of “Mechanical.”
- The **PartMaster** is not applicable under a **CompoundContext** composed of a **LocationContext** of “Detroit” and a **DisciplineContext** of “Electrical.”

6.4 PdmViews IDL

```
// PdmViews.idl

#ifndef  PDMVIEWS
#define  PDMVIEWS

#include <CosLifeCycle.idl>
#include <CosTransactions.idl>

#include <PdmFoundation.idl>
#include <PdmFramework.idl>

module PdmViews
{

  // Exceptions

  #define PDM_EXCEPTIONS \
    PdmFoundation::PdmError, \
    PdmFoundation::PermissionDenied, \
    PdmFoundation::ValidationError

  #define ITEM_CREATE_EXCEPTIONS \
    PdmFoundation::PdmError, \
    PdmFoundation::PermissionDenied, \
    PdmFoundation::ValidationError, \
    PdmFoundation::InvalidProperties, \
    PdmFoundation::NotUnique

  #define RELATIONSHIP_CREATE_EXCEPTIONS \
    PdmFoundation::PdmError, \
    PdmFoundation::PermissionDenied, \
    PdmFoundation::ValidationError, \
    PdmFoundation::InvalidProperties, \
    PdmFoundation::NotUnique, \
    PdmFoundation::CardinalityExceeded

exception UnsupportedTraversalName
{
  unsigned long error_code;
  string error_text;
};
```

```
  // Entities

  interface Qualification :
       CosTransactions::TransactionalObject,
       CosLifeCycle::LifeCycleObject,
       PdmFramework::Baselineable,
       PdmFoundation::Manageable {};

  interface PdmContext {};

interface PdmTraversalCriteriaFactory
{
  CosGraphs::TraversalCriteria create_simple_traversal_criteria(
       in PdmContext pdm_context, in CORBA::InterfaceDef role_type);
  CosGraphs::TraversalCriteria create_named_traversal_criteria(
       in PdmContext pdm_context, in string traversal_name)
    raises (UnsupportedTraversalName);
};

  interface ViewQualification : Qualification
  {
    attribute string name;
  };

  interface ViewQualificationFactory
  {
  ViewQualification create(
       in CosPropertyService::PropertySet property_set)
    raises (ITEM_CREATE_EXCEPTIONS);
  };

  interface ViewContext : PdmContext
  {
    attribute string name;
    attribute ViewQualification view_qualification;
  };

  interface LifeCycleQualification : ViewQualification {};

  interface LifeCycleQualificationFactory
  {
  LifeCycleQualification create(
       in CosPropertyService::PropertySet property_set)
    raises (ITEM_CREATE_EXCEPTIONS);
  };

  interface LifeCycleContext : ViewContext {};

  interface LifeCycleContextFactory
  {
    LifeCycleContext create(in string name)
```

```
    raises (ITEM_CREATE_EXCEPTIONS);
};

interface LocationQualification : ViewQualification {};

interface LocationQualificationFactory
{
LocationQualification create(
    in CosPropertyService::PropertySet property_set)
  raises (ITEM_CREATE_EXCEPTIONS);
};

interface LocationContext : ViewContext {};

interface LocationContextFactory
{
  LocationContext create( in string name)
    raises (ITEM_CREATE_EXCEPTIONS);
};

interface DisciplineQualification : ViewQualification {};

interface DisciplineQualificationFactory
{
DisciplineQualification create(
    in CosPropertyService::PropertySet property_set)
  raises (ITEM_CREATE_EXCEPTIONS);
};

interface DisciplineContext : ViewContext {};

interface DisciplineContextFactory
{
  DisciplineContext create( in string name)
    raises (ITEM_CREATE_EXCEPTIONS);
};

interface CompoundQualification : Qualification
{
  void add_qualification( in Qualification the_qualification );
  void remove_qualification( in Qualification the_qualification );
};

interface CompoundQualificationFactory
{
CompoundQualification create(
    in CosPropertyService::PropertySet property_set)
  raises (ITEM_CREATE_EXCEPTIONS);
};

interface CompoundContext : PdmContext
```

```
{
   void add_context( in PdmContext the_context )
      raises (ITEM_CREATE_EXCEPTIONS);
   void remove_context( in PdmContext the_context )
      raises (ITEM_CREATE_EXCEPTIONS);
};

interface CompoundContextFactory
{
   CompoundContext create()
      raises (ITEM_CREATE_EXCEPTIONS);
};

interface StateContext : ViewContext {};

interface StateContextFactory
{
   StateContext create( in string name )
      raises (ITEM_CREATE_EXCEPTIONS);
};

// Relationships

// Qualifies Relationship
//   role:  Qualified
//     name:  'Qualified'
//     entity:  Qualifiable
//     cardinality:  0..unbounded
//   role: Qualifier
//     name:  'Qualifier'
//     entity: Qualification
//     cardinality:  0..unbounded

interface Qualifies :
      PdmFramework::PdmReferenceRelationship,
      PdmFramework::Baselineable {};

interface QualifiesFactory
{
   Qualifies create(
         in CosPropertyService::PropertySet property_set,
         in PdmFramework::Qualifiable qualified,
         in Qualification qualifier)
      raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface Qualified : PdmFramework::PdmReferencesRole{};

interface Qualifier : PdmFramework::PdmReferencedByRole{};

// QualificationAggregation Relationship
```

```
//   role:  CompoundQualificationRole
//     name:  'CompoundQualificationRole'
//     entity:  CompoundQualification
//     cardinality:  2..unbounded
//   role: MemberQualificationRole
//     name:  'MemberQualificationRole'
//     entity: Qualification
//     cardinality:  0..unbounded

interface QualificationAggregation :
     PdmFramework::PdmReferenceRelationship {};

interface QualificationAggregationFactory
{
  QualificationAggregation create(
        in CosPropertyService::PropertySet property_set,
        in CompoundQualification compound_qualification,
        in Qualification qualification)
     raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface CompoundQualificationRole :
     PdmFramework::PdmReferencesRole{};

interface MemberQualificationRole :
     PdmFramework::PdmReferencedByRole{};

// ContextAggregation Relationship
//   role:  CompoundContextRole
//     name:  'CompoundContextRole'
//     entity:  CompoundContext
//     cardinality:  2..unbounded
//   role: MemberContextRole
//     name:  'MemberContextRole'
//     entity: PdmContext
//     cardinality:  0..unbounded

interface ContextAggregation :
     PdmFramework::PdmReferenceRelationship {};

interface ContextAggregationFactory
{
  ContextAggregation create(
        in CosPropertyService::PropertySet property_set,
        in CompoundContext compound_context,
        in PdmContext context_member)
     raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};
```

```
    interface CompoundContextRole :
        PdmFramework::PdmReferencesRole{};

    interface MemberContext : PdmFramework::PdmReferencedByRole{};

    };

#endif
```

PdmDocumentManagement Module 7

Contents

This chapter contains the following sections.

Section Title	Page
“Overview”	7-1
“PdmDocumentManagement Model”	7-3
“PdmDocumentManagement Description”	7-4
“Document Management IDL”	7-15

7.1 Overview

The scope of this module is to store and retrieve electronic documents comprising one or more files and track documents that are not actively managed by the PDM system. Compound documents and documents that are comprised of multiple files organized in other than a simple set are not in the current scope.

Tools that operate on documents, such as CAD, need to have the document files present on the client machines’ file system or a network directory accessible by the client. Almost all modern PDM systems are built on a client/server architecture, where the managed files reside on the PDM server. In order for the PDM server to implement checkin and checkout operations, the server, which typically resides on a different machine from the client, will have to be able to read from, and write to, the clients file system. PDM systems typically employ proprietary mechanisms to transfer files between the PDM server and client. Unfortunately, there is currently no OMG-approved file transfer mechanism. This module specifies a simple data transfer protocol that a client can use to transfer the contents of files between a PDM server and the client.

To perform a checkout, the client would first lock the file using the **lock()** operation. The client then transfers the files contents from the PDM server to its file system. To perform a checkin, the client creates a new **DocumentIteration**. It then creates a new File, associates it with the **DocumentIteration**, transfers the contents of the file from the client to the PDM server. Lastly, it unlocks the file.

The following sections describe a conceptual model for document management in UML, followed by descriptions of the CORBA interfaces. A complete IDL of this module is presented following the descriptions of the interfaces.

7.2 PdmDocumentManagement Model

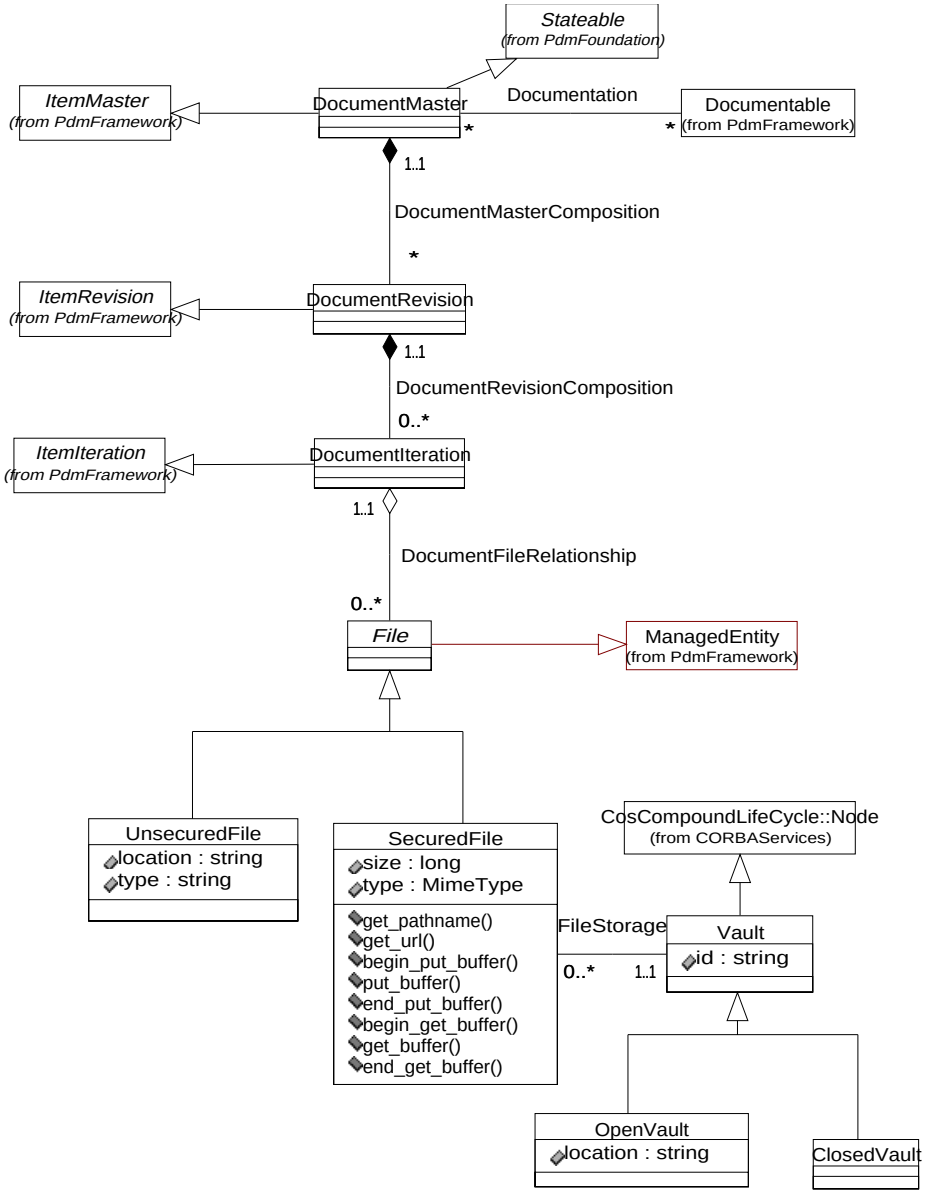

Figure 7-1 PdmDocumentManagement Model Diagram

7.3 PdmDocumentManagement Description

7.3.1 DocumentMaster

The **DocumentMaster** serves as a point of reference for the various revisions of the document. **DocumentMaster** also inherits, indirectly via **ItemMaster**, from **Identifiable**, which defines the document identifier (or document number) with respect to a specific identification context.

```
interface DocumentMaster : PdmFramework::ItemMaster { };
interface DocumentMasterFactory
{
DocumentMaster create(
  in CosPropertyService::PropertySet property_set)
raises (PdmFoundation::NotUnique,
        PdmFoundation::InvalidProperties,
        PdmFoundation::ValidationError,
        PdmFoundation::PermissionDenied,
        PdmFoundation::PdmError);
};
```

7.3.2 DocumentRevision

A **DocumentRevision** is a formally identified version of the document. It is a kind of **ItemRevision** and its revision identifier is obtained using the mechanism inherited from **Identifiable**. The **DocumentRevisionFactorys create()** operation takes the containing **DocumentMaster** as one of the arguments, ensuring that the **DocumentRevision** is always associated with the **DocumentMaster** for which it is a revision of.

```
interface DocumentRevision : ItemRevision { };

interface DocumentRevisionFactory
{
DocumentRevision create(
  in CosPropertyService::PropertySet property_set,
  in DocumentMaster document_master_for_revision)
raises (PdmFoundation::NotUnique,
        PdmFoundation::InvalidProperties,
        PdmFoundation::ValidationError,
        PdmFoundation::PermissionDenied,
        PdmFoundation::PdmError);
};
```

7.3.3 DocumentMasterComposition Relationship

DocumentMasters and **DocumentRevision**s are related by **DocumentMasterComposition** relations. This is a containment relationship. Each **DocumentRevision** is only valid within the scope of a **DocumentMaster** and does not have independent existence outside the scope of a **DocumentMaster**. Each **DocumentMaster** may be associated with multiple **DocumentRevision**s. The roles of the **DocumentMaster** and **DocumentRevision** in this relationship are those of container and containee respectively. There is no interface defined for the **DocumentMasterCompositionFactory** because a **DocumentRevision** cannot be created independently of its containing **DocumentMaster**.

```
// DocumentMasterComposition relationship
//   role: DocumentMasterForRevisions
//     name : 'DocumentMasterForRevisions'
//     entity: DocumentMaster
//     cardinality: 0..unbounded
//   role: DocumentRevisionForMaster
//     name: 'DocumentRevisionForMaster'
//     entity: DocumentRevision
//     cardinality: 1..1

interface DocumentMasterComposition :
  PdmFramework::PdmContainmentRelationship { };

interface DocumentMasterForRevisions :
  PdmFramework::PdmContainsRole { };

interface DocumentRevisionForMaster :
  PdmFramework::PdmContainedInRole { };
```

7.3.4 DocumentIteration

A **DocumentIteration** is where the documents data is stored. For documents, the bulk of the data consists of a file or a collection of files. In accordance with the scope of this module, the collection paradigm is that of a simple set, the files are not ordered and are not explicitly related. The **DocumentIterationFactorys' create()** operation takes the containing **DocumentRevision** as one of the arguments, ensuring that the **DocumentIteration** is always associated with the **DocumentRevision** for which it is an iteration of.

```
interface DocumentIteration : ItemIteration { };

interface DocumentIterationFactory
{
DocumentIteration create(
     in CosPropertyService::PropertySet property_set
     in DocumentRevision document_revision_for_iteration)
  raises (PdmFoundation::NotUnique,
       PdmFoundation::InvalidProperties,
```

```
        PdmFoundation::ValidationError,
        PdmFoundation::PermissionDenied,
        PdmFoundation::PdmError);
};
```

7.3.5 DocumentRevisionComposition Relationship

DocumentIterations are associated with **DocumentRevision**s through **DocumentRevisionComposition** relationships. This is a containment relationship. **DocumentIteration**s cannot exist outside the scope of the containing **DocumentRevision**. Each **DocumentRevision** can contain multiple **DocumentIteration**s.

The roles of the **DocumentRevision** and **DocumentIteration** in this relationship are those of container and containee respectively. There is no interface defined for the **DocumentRevisionCompositionFactory** because a **DocumentIteration** cannot be created independently of its containing **DocumentRevision**.

```
// DocumentRevisionComposition relationship
//   role: DocumentRevisionForIterations
//     name: 'DocumentRevisionForIterations'
//     entity: DocumentRevision
//     cardinality: 0..unbounded
//   role: DocumentIterationForRevision
//     name: 'DocumentIterationForRevision'
//     entity: DocumentIteration
//     cardinality: 1..1

interface DocumentRevisionComposition :
  PdmFramework::PdmContainmentRelationship {};

interface DocumentRevisionForIterations :
  PdmFramework::PdmContainsRole { };

interface DocumentIterationForRevision :
  PdmFramework::PdmContainedInRole { };
```

7.3.6 Documentable

The **Documentable** interface allows the concept that a PDM object can have documents associated to it. This potentially allows every **ManagedEntity** object to be associated to a **DocumentMaster** through the **Documentation relationship**. This relationship is useful when the maintenance and revision control of the **Documentable** object can be independent from that of documentation object.

In the PDM Enabler modules several **ManagedEntity** objects (such as **PartRevision**, **ProcessRevision**) have relationships to **DocumentRevision**. The **Documentation relationship** can be used for documents that are not part of the definition of an object but provide other information, which might be essential to

maintainence and revision control within the PDM system. Potential candidates for the **Documentation relationship** may be text notes to be attached to a part, design rationales, marketing specification, etc.

The IDL definition for the **Documentable** interface is found in the **PdmFramework** module (see Chapter 4).

7.3.7 Documentation Relationship

Documentation relationship associates a **ManagedEntity** to a **DocumentMaster**.

```
// Documentation relationship
//   role: Documented
//     name: 'Documented'
//     entity: Documentable
//     cardinality: 0..unbounded
//   role: Documenter
//     name: 'Documenter'
//     entity: DocumentMaster
//     cardinality: 0..unbounded

interface Documentation : PdmFramework::PdmReferenceRelationship {};

interface DocumentationFactory
{
Documentation create(
      in CosPropertyService::PropertySet property_set,
      in PdmFramework::Documentable documented,
      in DocumentMaster documenter)
   raises (PdmFoundation::NotUnique,
           PdmFoundation::InvalidProperties,
           PdmFoundation::ValidationError,
           PdmFoundation::PermissionDenied,
           PdmFoundation::PdmError);
};

interface Documented : PdmFramework::PdmReferencesRole {};

interface Documenter : PdmFramework::PdmReferencedByRole {};
```

7.3.8 File

In this module, *File* is an abstract class and therefore does not have a factory associated with it.

```
interface File : PdmFramework::ManagedEntity { };
```

There are a number of issues related to file naming in a multi-platform environment. For example, some file systems restrict the length of a file name, and certain characters cannot be used in file names. The **name** attribute is merely a convenient reference. Since **File** inherits from **ManagedEntity**, which in turn inherits from **Identifiable**, it is recommended that file system-specific file names be defined as non-unique identifiers within the scope of an identification context based on the file system.

7.3.9 DocumentFileRelationship Relationship

Files are associated with **DocumentIteration**s through a **DocumentFileRelationship** relationship. While a **File** may only be associated with one **DocumentIteration**, it can have independent existence outside the scope of a **DocumentIteration**. Also, a **DocumentIteration** does not have to contain **File**s as a kind of **ManagedEntity**, attributes can be associated with it using the Property Service. The role of the **DocumentIteration** in this relationship is that of the document for which files are associated. The role of the **File** is simply the file that is associated with a document. While a **File** need not be associated with a document, the cardinality of the **FileForDocument** role means that a **File** can be associated with only one **DocumentIteration**.

```
// DocumentFileRelationship relationship
//   role: DocumentForFiles
//     name: 'DocumentForFiles'
//     entity: DocumentIteration
//     cardinality: 0..unbounded
//   role: FileForDocument
//     name: 'FileForDocument'
//     entity: File
//     cardinality: 1..1

interface DocumentFileRelationship :
  PdmFramework::PdmReferenceRelationship { };

interface DocumentFileRelationshipFactory
{
DocumentFileRelationship create(
     in CosPropertyService::PropertySet property_set,
     in DocumentIteration document_for_files,
     in File file_for_document)
  raises (PdmFoundation::CardinalityExceeded,
          PdmFoundation::NotUnique,
          PdmFoundation::InvalidProperties,
          PdmFoundation::ValidationError,
          PdmFoundation::PermissionDenied,
          PdmFoundation::PdmError);
};
```

```
interface DocumentForFiles : PdmFramework::PdmReferencesRole { };

interface FileForDocument : PdmFramework::PdmReferencedByRole { };
```

7.3.10 UnsecuredFile

Unsecured files are tracked, but not stored or managed. Examples of unsecured files are paper drawings and files managed by a tape jukebox. This allows the PDM system to be used as a kind of drawing register to track these files, and perhaps to manually control checkin and checkout.

```
interface UnsecuredFile : File
{
  attribute string location;
  attribute string type;
};

interface UnsecuredFileFactory
{
  UnsecuredFile create(
       in CosPropertyService::PropertySet property_set)
    raises (PdmFoundation::InvalidProperties,
          PdmFoundation::ValidationError,
          PdmFoundation::PermissionDenied,
          PdmFoundation::PdmError);
};
```

location

Where a file might be found. This may be a physical location, such as rack 4 slot 5 in the case of hardcopy, or a network directory for electronic files.

type

Since unsecured files can refer to hardcopy as well as electronic media, the **type** indicator may indicate media such as aperture card.

7.3.11 SecuredFile

Secured files are those that are stored and managed by the PDM system. A secured file is stored in the Vault, which may be *open* or *closed*. Each **SecuredFile** is associated with only one logical vault even though some PDM systems allow for multiple physical vaults. The rationale is that, except for open vaults, the user should not have to know the physical location of the vault. This simplifies administration and allows PDM systems to more easily implement advanced features such as file replication. The type of the file is identified using standard MIME type, subtype and parameters as follows:

```
typedef string MimeMediaType;      // MIME media type
typedef string MimeMediaSubtype;   // MIME subtype
```

```
typedef string MimeMediaTypeParam;// MIME type parameters

struct MimeType
{
  MimeMediaType type;
  MimeMediaSubtype subtype;
  MimeMediaTypeParam parameters;
};
```

If the **parameter** field is empty and the **type/subtype** is text/plain, **parameter** "CHAR-SET=US-ASCII" is assumed. If the server does not recognize the **type/subtype**, it shall treat it as application/octet-stream.

For more information on MIME types, refer to RFC1521. Registered types are maintained by the Internet Assignment Numbers Authority (IANA) (http://www.iana.org) site at: http://www.isi.edu/in-notes/iana/assignments/media-types/. At this time, MIME has defined the following types: text, multipart, message, application, image, audio, video, and model.

The server will store the document as given by the client, and the server will return the document as stored. On **get_buffer** transfers of text files, it is the client's responsibility to modify end-of-line indicators to meet the needs of the client system and its applications. Since the storing client may be different from the receiving client, the retrieving client cannot expect the server to provide end-of-line markers in a consistent fashion.

```
typedef sequence<octet> buffer;

interface SecuredFile : File
{
  attribute long size;       // file size in octets
  attribute MimeType type;

  string get_pathname()
    raises (NotValidForClosedVault);

  long begin_put_buffer(in long bufsize,
       in long filesize,
       in MimeType type)
    raises (InvalidBufferSize,
            InvalidFileSize,
            InvalidMimeType,
            FileIsLocked,
            FileIsBusy,
            InsufficientStorage,
            PdmFoundation::PermissionDenied,
            PdmFoundation::PdmError);
  void put_buffer(in buffer buf)
    raises (TransferNotInitiated,
            InvalidBufferSize,
            FileSizeExceeded,
```

```
                BufferEmpty,
                InsufficientStorage,
                PdmFoundation::PdmError);
    void end_put_buffer()
      raises (TransferNotInitiated,
                NoDataTransferred,
                TransferIncomplete,
                PdmFoundation::PdmError);
    long begin_get_buffer(in long bufsize,
          in long filesize,
          in MimeType type)
      raises (InvalidBufferSize,
                FileIsLocked,
                FileIsBusy,
                NoDataToTransfer,
                PdmFoundation::PermissionDenied,
                PdmFoundation::PdmError);
    buffer get_buffer()
      raises (TransferNotInitiated,
                NoMoreDataToTransfer,
                PdmFoundation::PdmError);
    void end_get_buffer()
      raises (TransferNotInitiated,
                NoDataTransferred,
                TransferIncomplete,
                PdmFoundation::PdmError);
    string get_url()
      raises (NotAvailable, NotValidForClosedVault);
};

interface SecuredFileFactory
{
    SecuredFile create(
          in CosPropertyService::PropertySet property_set,
          in Vault file_vault)
      raises (PdmFoundation::InvalidProperties,
                PdmFoundation::ValidationError,
                PdmFoundation::PermissionDenied,
                PdmFoundation::PdmError);
};
```

get_pathname

If the secured file is stored in an open vault, it is possible to obtain the full pathname of the file (from the servers perspective) by calling the **get_pathname()** operation. If the file is stored in a closed vault, a NotValidForClosedVault exception will be thrown.

get_url

If the **SecuredFile** is stored in an open vault that is accessible by some standard remote file access protocol, the client may obtain a pathname to the file in the form of an Internet Universal Resource Locator (URL) via the **get_url** operation. If the vault is closed, the NotValidForClosedVault exception will be thrown. If the vault is not accessible by a remote access service, the NotAvailable exception will be thrown.

transfer operations

Refer to Section 7.3.14, “Data Transfer,” on page 7-13 for a description of the other operations on **SecuredFile**.

7.3.12 Vaults

A secured file is stored in a Vault, which may be open or closed. The find method on the **VaultFactory** interface is used to get a handle to a Vault with a particular id. The attributes of the Vault are readonly since vaults are managed by the PDM system.

An open vault is a file system directory where the file may be directly accessed. The **location** attribute is the network directory path from the servers point of view. The path name of a file in an open vault may be obtained through the **SecuredFile get_pathname()** method, which will return the full path name of the file from the servers perspective. Whenever possible, a network- or cell-wide absolute pathname, such as a Windows UNC name, will be returned. If the client has access to the network directory, it may unlock the file and operate on it without having to transfer its contents to its local directory.

A closed vault is a black box, no location information is available to the client.

```
interface Vault : CosCompoundLifeCycle::Node
{
  readonly attribute string id;
};

typedef  sequence<Vault> Vaults;

interface VaultFactory
{
  Vault find(in string vault_id)
    raises(PdmFoundation::PdmError);
  Vaults get_vaults()
    raises(PdmFoundation::PdmError);
};

interface OpenVault : Vault
{
  readonly attribute string location;  // network directory path
};

interface ClosedVault : Vault {};
```

7.3.13 FileStorage Relationship

Only **SecuredFiles** are associated with a **Vault** and each **SecuredFile** can be associated with only one **Vault** through the **FileStorage** relationship. The **Vault** may be an **OpenVault** or a **ClosedVault**. The role of the **SecuredFile** in this relationship is that which is stored in a vault. The role of the vault in this relationship is that which stored the file.

```
// FileStorage relationship
//   role: FileForVault
//     name: 'FileForVault'
//     entity:  SecuredFile
//     cardinality: 1..1
//   role: VaultForFiles
//     name: 'VaultForFiles'
//     entity:  Vault
//     cardinality: 0..unbounded

interface FileStorage : PdmFramework::PdmContainmentRelationship {};

interface VaultForFiles : PdmFramework::PdmContainsRole { };

interface FileForVaults : PdmFramework::PdmContainedInRole { };
```

7.3.14 Data Transfer

Management of secured files requires the ability to transfer the contents of a file between a users machine and the PDM system. The requirements are:

- The mechanism must be able to transfer very large files that may not fit entirely in memory.
- The transfer mechanism must not consume excessive memory on the client or server side. For example, reading an entire 10 MB file into memory is not acceptable.
- The file must remain unchanged after a round-trip through the PDM vault. Cross-platform file format issues, such as moving files between Macintosh and Windows platforms, are not in the scope of this module.

This module specifies an extremely simple data transfer protocol, transferring a file as a series of octet sequences. The goals are simple and cheap. More sophisticated protocols have been considered and rejected in favor of simplicity and ease of implementation. It is expected that this protocol will be replaced when an OMG standard for file transfer becomes available. The section following this explains why the Externalization Service is unsuitable for our purpose.

The protocol is a client-side push-pull protocol, so called because the client does all the work, it pushes the file to the server on create and checkin, and it pulls the file from the server on checkout and copyout.

On send, there is a negotiation phase during which the sender invokes **begin_put_buffer()** to inform the server/receiver of the buffer size it proposes to use. Additional parameters include the size of the file in octets and its MIME type. The server/receiver responds with a buffer size that must be equal to, or smaller than the senders proposed buffer size. This is the buffer size that will be used for the transfer. The negotiated buffer size shall not be smaller than 256 octets. Except for the last buffer, all buffers used in the transfer must be the exact negotiated size. The last buffer may be smaller than 256 octets. The client/sender then executes a series of **put_buffer()** calls until the content of the file is completely transferred. The client/sender then calls **end_put_buffer()** to signal the end of the transfer; **end_put_buffer()** can also be called to abort a transfer. The only requirements of this protocol are that the data be sent in sequence, and the **put_buffer()** must return before sending the next buffer. This process is depicted in Figure 7-2. If the file already contains data when **begin_put_buffer()** is called, its data will be replaced.

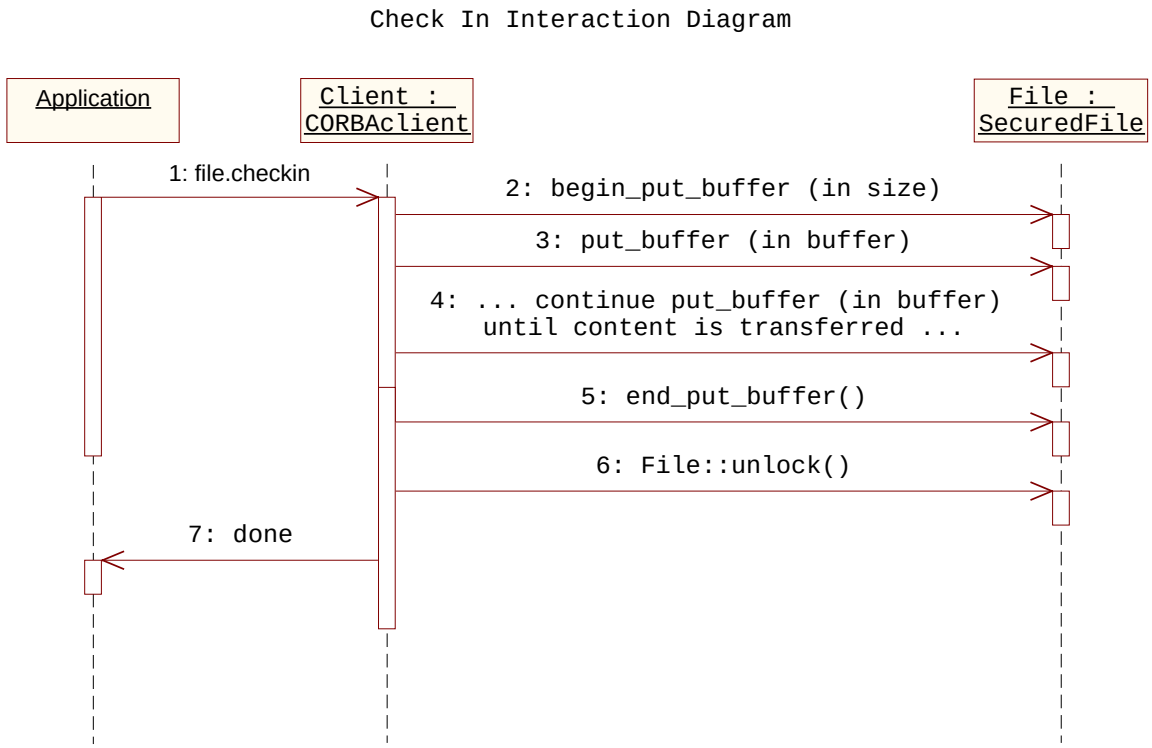

Figure 7-2 How the data transfer protocol may be used to check in a document file.

On completion of the transfer, the client unlocks the file to complete the checkin process.

The process is similar when the client receives a file. The client/receiver calls **begin_get_buffer()** and proposes a buffer size and transfer encoding. The server/sender responds with a buffer size that must be equal to, or smaller than the proposed size. This is the buffer size that will be used. The client/receiver then makes a series of **get_buffer()** calls until the file content is transferred. This process is shown in Figure 7-3.

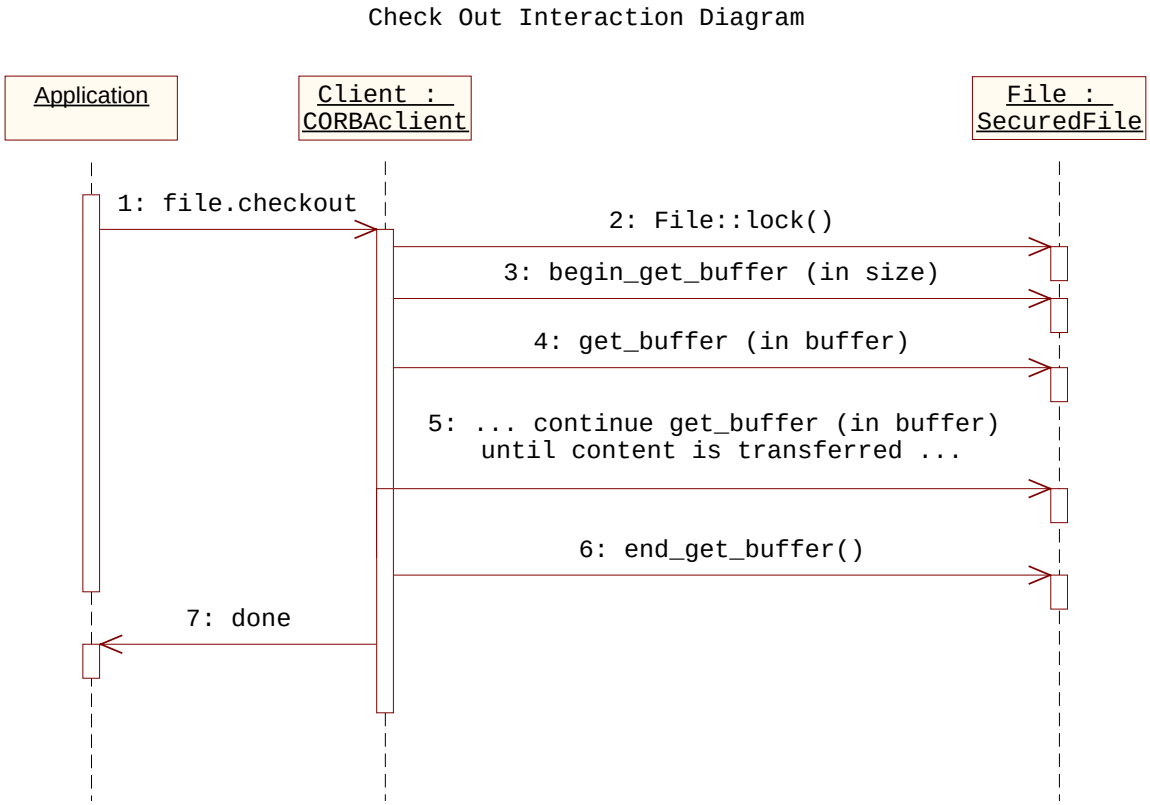

Figure 7-3 How the data transfer protocol may be used to check out a document file.

This is *not* a file transfer protocol, but rather a data transfer protocol. Reading and writing files into and from buffers is the responsibility of the application. A sophisticated client may interleave network and disk i/o by executing **put_buffer()** or **get_buffer()** as a CORBA deferred synchronous call.

7.4 Document Management IDL

```
// PdmDocumentManagement.idl

#ifndef  PDMDOCUMENTMANAGEMENT
#define  PDMDOCUMENTMANAGEMENT

#include <CosPropertyService.idl>
#include <CosRelationships.idl>
```

```
#include <PdmFoundation.idl>
#include <PdmFramework.idl>

module PdmDocumentManagement
{

// Forward references
interface DocumentRevision;

// Data Types
typedef string MimeMediaType;      // MIME media type
typedef string MimeMediaSubtype;   // MIME subtype
typedef string MimeMediaTypeParam; // MIME type parameters

struct MimeType
{
  MimeMediaType type;
  MimeMediaSubtype subtype;
  MimeMediaTypeParam parameters;
};

typedef sequence<octet> buffer;

// Exceptions
exception BufferEmpty
{
  unsigned long error_code;
  string error_text;
};

exception FileIsBusy  // transaction lock on file
{
  unsigned long error_code;
  string error_text;
};

exception FileIsLocked  // locked by someone else
{
  unsigned long error_code;
  string error_text;
};

exception FileSizeExceeded
{
  unsigned long error_code;
  string error_text;
};

exception InsufficientStorage
{
  unsigned long error_code;
```

```
  string error_text;
};

exception InvalidBufferSize
{
  unsigned long error_code;
  string error_text;
};

exception InvalidFileSize
{
  unsigned long error_code;
  string error_text;
};

exception InvalidMimeType
{
  unsigned long error_code;
  string error_text;
};

exception TransferNotInitiated
{
  unsigned long error_code;
  string error_text;
};

exception NoDataTransferred
{
  unsigned long error_code;
  string error_text;
};

exception NoDataToTransfer
{
  unsigned long error_code;
  string error_text;
};

exception NoMoreDataToTransfer
{
  unsigned long error_code;
  string error_text;
};

exception NotValidForClosedVault
{
  unsigned long error_code;
  string error_text;
};
```

```
exception NotAvailable
{
    unsigned long error_code;
    string error_text;
};

exception TransferIncomplete
{
   unsigned long error_code;
   string error_text;
};

// Entities

interface Vault : CosCompoundLifeCycle::Node
{
   readonly attribute string id;
};

typedef sequence<Vault> Vaults;

interface VaultFactory
{
   Vault find(in string vault_id)
      raises(PdmFoundation::PdmError);
Vaults get_vaults()
   raises(PdmFoundation::PdmError);
};

interface ClosedVault : Vault { };

interface OpenVault : Vault
{
   readonly attribute string location;
};

interface DocumentIteration : PdmFramework::ItemIteration { };

interface DocumentIterationFactory
{
   DocumentIteration create(
         in CosPropertyService::PropertySet property_set,
           in DocumentRevision document_revision_for_iterations)
      raises (PdmFoundation::NotUnique,
            PdmFoundation::InvalidProperties,
            PdmFoundation::ValidationError,
            PdmFoundation::PermissionDenied,
            PdmFoundation::PdmError);
};

interface DocumentMaster : PdmFramework::ItemMaster { };
```

```
interface DocumentMasterFactory
{
  DocumentMaster create(
        in CosPropertyService::PropertySet property_set)
    raises (PdmFoundation::NotUnique,
          PdmFoundation::InvalidProperties,
          PdmFoundation::ValidationError,
          PdmFoundation::PermissionDenied,
          PdmFoundation::PdmError);
};

interface DocumentRevision : PdmFramework::ItemRevision { };

interface DocumentRevisionFactory
{
  DocumentRevision create(
        in CosPropertyService::PropertySet property_set,
          in DocumentMaster document_master_for_revisions)
    raises (PdmFoundation::NotUnique,
          PdmFoundation::InvalidProperties,
          PdmFoundation::ValidationError,
          PdmFoundation::PermissionDenied,
          PdmFoundation::PdmError);
};

interface File : PdmFramework::ManagedEntity
{
  attribute string name;
};

interface SecuredFile : File
{
  attribute long size;
  attribute MimeType type;

  string get_pathname()
    raises(NotValidForClosedVault);

  long begin_put_buffer(in long bufsize,
        in long filesize,
        in MimeType type)
    raises (InvalidBufferSize,
          InvalidFileSize,
          InvalidMimeType,
          FileIsLocked,
          FileIsBusy,
          InsufficientStorage,
          PdmFoundation::PermissionDenied,
          PdmFoundation::PdmError);
  void put_buffer(in buffer buf)
```

```
      raises (TransferNotInitiated,
            InvalidBufferSize,
            FileSizeExceeded,
            BufferEmpty,
            InsufficientStorage,
            PdmFoundation::PdmError);
   void end_put_buffer()
      raises (TransferNotInitiated,
            NoDataTransferred,
            TransferIncomplete,
            PdmFoundation::PdmError);

long begin_get_buffer(in long bufsize,
      in long filesize,
         in MimeType type)
      raises (InvalidBufferSize,
            FileIsLocked,
            FileIsBusy,
            NoDataToTransfer,
            PdmFoundation::PermissionDenied,
            PdmFoundation::PdmError);
   buffer get_buffer()
      raises (TransferNotInitiated,
            NoMoreDataToTransfer,
            PdmFoundation::PdmError);
   void end_get_buffer()
      raises (TransferNotInitiated,
            NoDataTransferred,
            TransferIncomplete,
            PdmFoundation::PdmError);
   string get_url()
      raises (NotAvailable,
            NotValidForClosedVault);
};

interface SecuredFileFactory
{
   SecuredFile create(in CosPropertyService::PropertySet property_set,
                  in Vault file_vault)
      raises (PdmFoundation::InvalidProperties,
            PdmFoundation::ValidationError,
            PdmFoundation::PermissionDenied,
            PdmFoundation::PdmError);
};

interface UnsecuredFile : File
{
   attribute string location;
   attribute string type;
};
```

```
interface UnsecuredFileFactory
{
  UnsecuredFile create(
        in CosPropertyService::PropertySet property_set)
     raises (PdmFoundation::InvalidProperties,
           PdmFoundation::ValidationError,
           PdmFoundation::PermissionDenied,
           PdmFoundation::PdmError);
};

// Relationships

// Documentation relationship
//   role: Documented
//     name: 'Documented'
//     entity: Documentable
//     cardinality: 0..unbounded
//   role: Documenter
//     name: 'Documenter'
//     entity: DocumentMaster
//     cardinality: 0..unbounded

interface Documentation : PdmFramework::PdmReferenceRelationship { };

interface DocumentationFactory
{
  Documentation create(
        in CosPropertyService::PropertySet property_set,
        in PdmFramework::Documentable documented,
        in DocumentMaster documenter)
     raises (PdmFoundation::NotUnique,
           PdmFoundation::InvalidProperties,
           PdmFoundation::ValidationError,
           PdmFoundation::PermissionDenied,
           PdmFoundation::PdmError);
};

interface Documented : PdmFramework::PdmReferencesRole { };

interface Documenter : PdmFramework::PdmReferencedByRole { };

// DocumentFileRelationship relationship
//   role: DocumentForFiles
//     name: 'DocumentForFiles'
//     entity: DocumentIteration
//     cardinality: 0..unbounded
//   role: FileForDocument
//     name: 'FileForDocument'
//     entity: File
//     cardinality: 1..1
```

```
interface DocumentFileRelationship :
     PdmFramework::PdmReferenceRelationship { };

interface DocumentFileRelationshipFactory
{
  DocumentFileRelationship create(
        in CosPropertyService::PropertySet property_set,
        in DocumentIteration document_for_files,
        in File file_for_document)
     raises (PdmFoundation::CardinalityExceeded,
           PdmFoundation::NotUnique,
           PdmFoundation::InvalidProperties,
           PdmFoundation::ValidationError,
           PdmFoundation::PermissionDenied,
           PdmFoundation::PdmError);
};

interface DocumentForFiles : PdmFramework::PdmReferencesRole { };

interface FileForDocument : PdmFramework::PdmReferencedByRole { };

// DocumentMasterComposition relationship
//   role: DocumentMasterForRevisions
//     name : 'DocumentMasterForRevisions'
//     entity: DocumentMaster
//     cardinality: 0..unbounded
//   role: DocumentRevisionForMaster
//     name: 'DocumentRevisionForMaster'
//     entity: DocumentRevision
//     cardinality: 1..1

interface DocumentMasterComposition :
     PdmFramework::PdmContainmentRelationship { };

interface DocumentMasterForRevisions :
     PdmFramework::PdmContainsRole { };

interface DocumentRevisionForMaster :
     PdmFramework::PdmContainedInRole { };

// DocumentRevisionComposition relationship
//   role: DocumentRevisionForIterations
//     name: 'DocumentRevisionForIterations'
//     entity: DocumentRevision
//     cardinality: 0..unbounded
//   role: DocumentIterationForRevision
//     name: 'DocumentIterationForRevision'
//     entity: DocumentIteration
//     cardinality: 1..1

interface DocumentRevisionComposition :
```

```
        PdmFramework::PdmContainmentRelationship {};

    interface DocumentRevisionForIterations :
        PdmFramework::PdmContainsRole { };

    interface DocumentIterationForRevision :
        PdmFramework::PdmContainedInRole { };

    // FileStorage relationship
    //   role: FileForVault
    //     name: 'FileForVault'
    //     entity:  SecuredFile
    //     cardinality: 1..1
    //   role: VaultForFiles
    //     name: 'VaultForFiles'
    //     entity:  Vault
    //     cardinality: 0..unbounded

    interface FileStorage : PdmFramework::PdmContainmentRelationship { };


    interface VaultForFiles : PdmFramework::PdmContainsRole { };

    interface FileForVault : PdmFramework::PdmContainedInRole { };

    };

    #endif
```

PdmProductStructureDefinition Module 8

Contents

This chapter contains the following sections.

Section Title	Page
“Overview”	8-1
“PdmProductStructureDefinition Model”	8-3
“PdmProductStructureDefinition Description”	8-4
“PdmProductStructureDefinition IDL”	8-17

8.1 Overview

This enabler includes the primary objects used for product data management. At the heart of this enabler is the group of objects that define parts and the bill of material relationships between items for discrete manufacturing. However, only the **PartStructure** and **Usage** interfaces are specific to discrete manufacturing. Other product description interfaces may also apply to other manufacturing industries, but may not be complete.

A part may represent one of a variety of physical entities used in discrete manufacturing; including raw material, semi-finished parts, assemblies, instruction manuals, kits, manufacturing by-products, and products. The manufacturing industry is defined by the design, production, and sales of parts, and almost every business activity in some way works with data that describes parts. Therefore, a part is not defined by a single object with a set of attributes, but a collection of objects and relationships, each describing different aspects of the part. For example, a part definition may consist of several engineering attributes, links to suppliers of the part, references to CAD

drawings describing the parts geometry, and a list of components used to assemble the part. These different pieces of the part definition will be referred to as part data objects.

These enablers do not attempt to define every part characteristic, but to define a set of objects that can represent a typical part definition for a modern manufacturing organization. Obviously, there is not a single model of a part that fits the needs of every company, but to achieve the goal of interoperability described by the RFP, it is necessary to provide a model that includes the majority of item attributes used within manufacturing. The enablers should also provide an obvious and structured means to expand this part definition to include additional attributes and relations and still maintain the established interoperability.

The objects, attributes, and relations below define a part definition template and are based on existing standards, such as the STEP PDM-related integrated resources that are the basis for AP203 and AP214, and data models implemented by numerous PDM customers.

8.2 PdmProductStructureDefinition Model

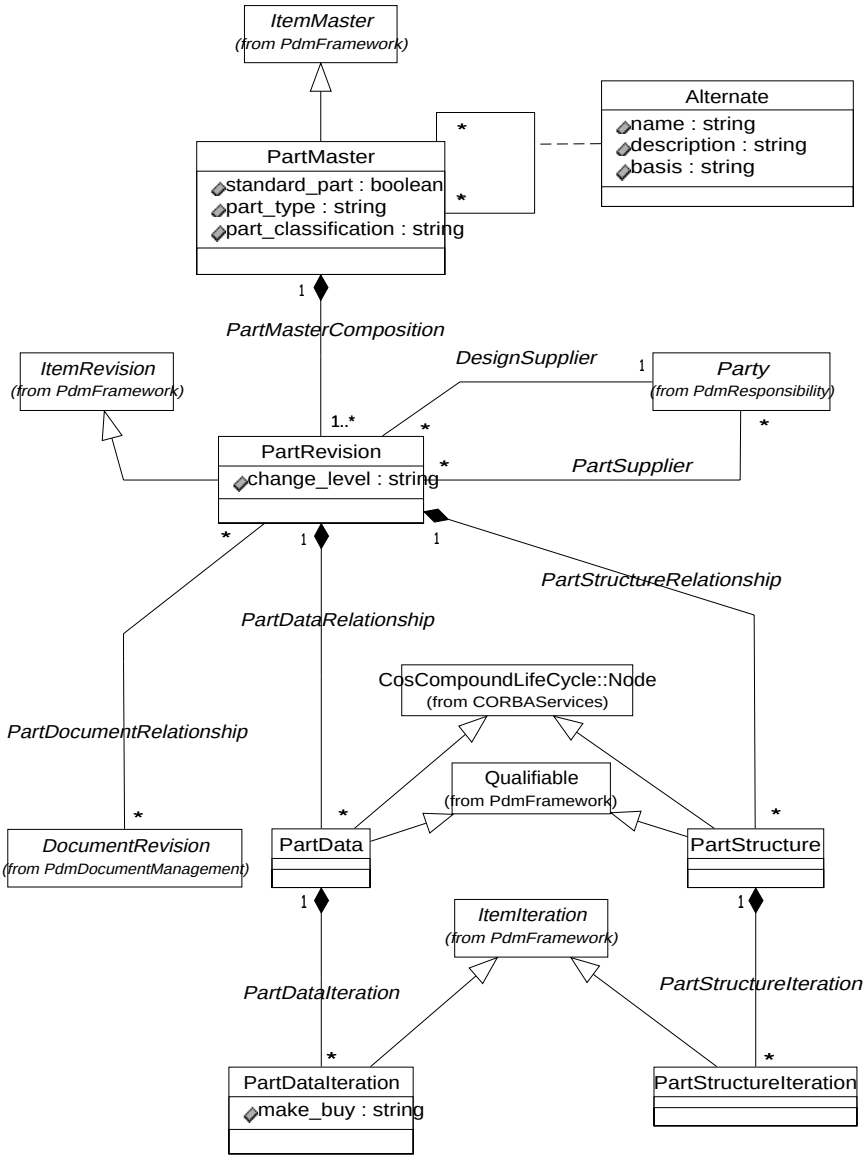

Figure 8-1 PdmProductStructureDefinition Model Diagram (1)

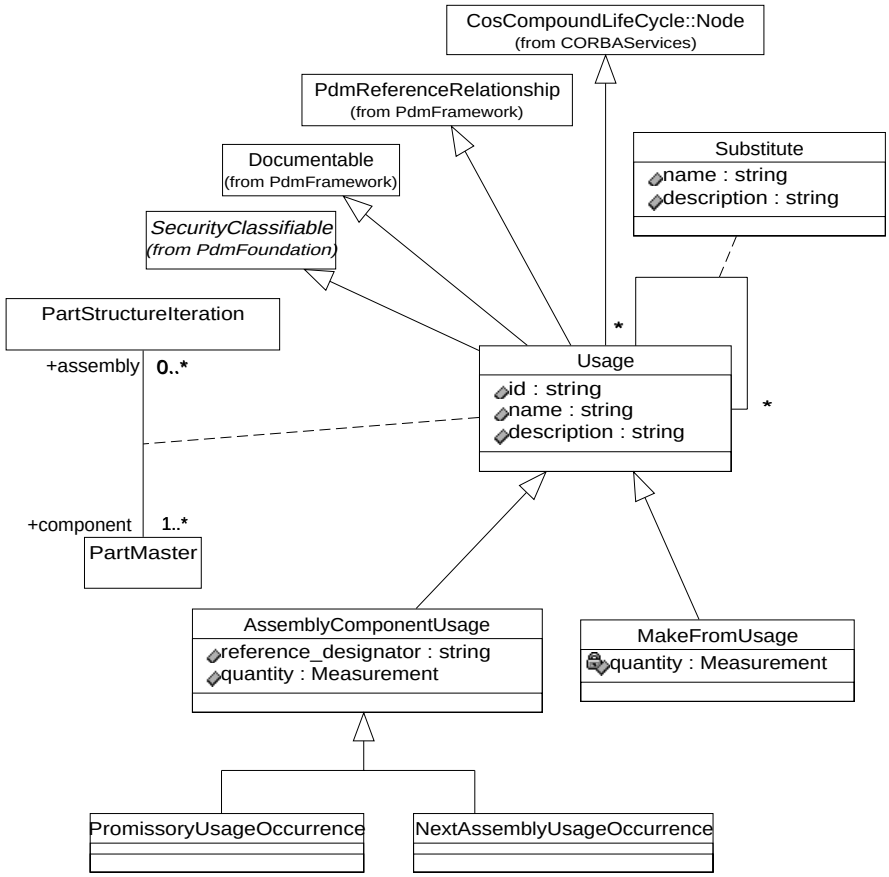

Figure 8-2 PdmProductStructureDefinition Model Diagram (2)

8.3 PdmProductStructureDefinition Description

8.3.1 Alternate

An **Alternate** object is a part that is interchangeable with another part with respect to form, fit, and function.

// Alternate Relationship
// role : BasePart
// name : 'BasePart'
// entity : PartMaster

```
//    cardinality : 0..unbounded
//  role : AlternatePart
//    name : 'AlternatePart'
//    entity : PartMaster
//    cardinality : 0..unbounded

interface Alternate : PdmFramework::PdmReferenceRelationship
{
  attribute string name;
  attribute string description;
  attribute string basis;
};

interface AlternateFactory
{
  Alternate create(
        in CosPropertyService::PropertySet property_set,
        in PartMaster alternate_part,
        in PartMaster base_part);
     raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface BasePart : PdmFramework::PdmReferencesRole {};

interface AlternatePart : PdmFramework::PdmReferencedByRole {};
```

name

Provides a name of the alternate part.

description

Contains text that describes the alternate part.

basis

Provides a text description to specify the rationale and domain of applicability of the alternate product.

8.3.2 AssemblyComponentUsage

An **AssemblyComponentUsage** represents the relationship between an assembly and a sub-assembly or component. The assembly need not be the immediate parent of the sub-assembly or component.

```
// AssemblyComponentUsage Relationship
//  role : Assembly (from Usage)
//    name : 'Assembly'
//    entity : PartStructureIteration
//    cardinality : 1..unbounded
//  role : Component (from Usage)
```

```
//     name : 'Component'
//     entity : PartMaster
//     cardinality : 0..unbounded

interface AssemblyComponentUsage :  Usage
{
  attribute string reference_designator;
  attribute PdmFoundation::Measurement quantity;
};
```

reference_designator

Refers to a unique identifier that distinguishes an instance of a component in an assembly that uses more than one instance of the component. For example, R1 and R2 are used to distinguish between two resistors in a circuit board.

quantity

The quantity attribute specifies how much or how many of the component is required in the next higher assembly. For example, 2.1 feet of hose or 5 each bolts. If the approximate flag in the Measurement structure is true, the quantity (if given) is an estimate of the quantity required. For example, when painting an automobile, you only use as much paint as required to achieve the required coverage.

8.3.3 DesignSupplierRelationship

The **DesignSupplierRelationship** relates the **PartRevisonChangeLevel** object to the **Party**, which plays the role of design supplier for this object.

```
// DesignSupplierRelationship Relationship
//   role : DesignSupplier
//     name : 'DesignSupplier'
//     entity : Party
//     cardinality : 0..unbounded
//   role : design_supplied
//     name : 'DesignSupplied'
//     entity : PartRevision
//     cardinality : 1..1

interface DesignSupplierRelationship :
  PdmFramework::PdmReferenceRelationship {};

interface DesignSupplierRelationshipFactory
{
  DesignSupplierRelationship create(
        in CosPropertyService::PropertySet property_set,
        in PartRevision supplied,
        in PdmResponsibility::Party supplier);
     raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};
```

```
interface DesignSupplied : PdmFramework::PdmReferencesRole {};

interface DesignSupplier : PdmFramework::PdmReferencedByRole {};
```

8.3.4 PartDocumentRelationship

The **PartDocumentRelationship** relates a **PartRevision** to a **DocumentRevision** that defines some aspect(s) of the part/product.

```
// PartDocumentRelationship Relationship
//   role : PartRevisionOfPartDocument
//    name : 'PartRevisionOfPartDocument'
//    entity : PartRevision
//    cardinality : 0..unbounded
//   role : DocumentRevisionOfPartDocument
//    name : 'DocumentRevisionOfPartDocument'
//    entity : PdmDocumentManagement::DocumentRevision
//    cardinality : 0..unbounded

interface PartDocumentRelationship :
  PdmFramework::PdmReferenceRelationship {};

interface PartDocumentRelationshipFactory
{
  PartDocumentRelationship create (
        in CosPropertyService::PropertySet property_set,
        in PartRevision part_revision,
        in PdmDocumentManagement::DocumentRevision document)
     raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface PartRevisionOfPartDocument :
  PdmFramework::PdmReferencesRole {};

interface DocumentRevisionOfPartDocument :
  PdmFramework::PdmReferencedByRole {};
```

8.3.5 NextAssemblyUsageOccurrence

A **NextAssemblyUsageOccurrence** is the relationship between the **PartStructureIteration** object and the **PartMaster** object.

```
// NextAssemblyUsageOccurrence Relationship
//   role : Assembly (from AssemblyComponentUsage)
//     name : 'Assembly'
//     entity : PartStructureIteration
//     cardinality : 1..unbounded
//   role : Component (from AssemblyComponentUsage)
```

```
//    name : 'Component'
//    entity : PartMaster
//    cardinality : 0..unbounded

interface NextAssemblyUsageOccurrence : AssemblyComponentUsage {};

interface NextAssemblyUsageOccurrenceFactory
{
  NextAssemblyUsageOccurrence create(
        in CosPropertyService::PropertySet property_set,
        in PartStructureIteration assembly,
        in PartMaster component);
      raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};
```

8.3.6 PartData

The **PartData** is an object that captures those attributes related to the part, which may be subject to iterations.

```
interface PartData : CosCompoundLifeCycle::Node,
    PdmFramework::Qualifiable {};

interface PartDataFactory
{
  PartData create(
        in PartRevision part_revision);
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

8.3.7 PartDataIteration

A **PartDataIteration** is a form, fit, and functional representative of a single **PartRevision**. It differs from other **PartDataIteration** of the same **PartRevision** through an informal change.

```
interface PartDataIteration : PdmFramework::ItemIteration
{
  attribute string make_buy;
};

interface PartDataIterationFactory
{
  PartDataIteration create(
    in CosPropertyService::PropertySet property_set,
    in PartData part_data);
  raises (ITEM_CREATE_EXCEPTIONS);
};
```

make_buy

Indicates that the organization plans to manufacture the part if the value is made. If the value is bought, the organization plans to purchase the part. The **make_buy** attribute can differ between revisions. Suggested values are: Make, Buy, Both, Unknown.

8.3.8 PartDataIterationRelationship

The **PartDataIterationRelationship** object relates the **PartData** to its iterations.

```
// PartDataIterationRelationship Relationship
//   role : PartDataToPartDataIteration
//     name : 'PartDataToPartDataIteration'
//     entity : PartData
//     cardinality : 0..unbounded
//   role : PartDataIterationToPartData
//     name : 'PartDataIterationToPartData'
//     entity : PartDataIteration
//     cardinality : 1..1

interface PartDataIterationRelationship :
  PdmFramework::PdmContainmentRelationship {};

interface PartDataToPartDataIteration :
  PdmFramework::PdmContainsRole {};

interface PartDataIterationToPartData :
  PdmFramework::PdmContainedInRole {};
```

8.3.9 PartDataRelationship

The **PartDataRelationship** relates the **PartData** object to its **PartRevisionChangeLevel**.

```
// PartDataRelationship Relationship
//   role : PartRevisionToPartData
//     name : 'PartRevisionToPartData'
//     entity : PartRevision
//     cardinality : 0..unbounded
//   role : PartDataToPartRevision
//     name : 'PartDataToPartRevision'
//     entity : PartData
//     cardinality : 1..1

interface PartDataRelationship :
  PdmFramework::PdmContainmentRelationship {};

interface PartRevisionToPartData :
  PdmFramework::PdmContainsRole {};
```

```
interface PartDataToPartRevision :
  PdmFramework::PdmContainedInRole {};
```

8.3.10 PartMaster

The **PartMaster** represents an item that is intended to be produced or employed in a production process. It encapsulates attributes and behavior that do not change through form, fit, and function replaceable revisions of the part.

```
interface PartMaster : PdmFramework::ItemMaster
{
  attribute boolean standard_part;
  attribute string part_type;
  attribute string part_classification;
};

interface PartMasterFactory
{
  PartMaster create(
    in CosPropertyService::PropertySet property_set);
  raises (ITEM_CREATE_EXCEPTIONS);
};
```

standard_part

Indicates whether or not the part is a standard component intended to be used in multiple designs.

part_type

Indicates the type of the part. The meaning and valid values of **part_type** may be determined by the site's business rules or by reference to a standard information model, such as a STEP application protocol.

part_classification

Provides a way to classify the part. The meaning and valid values of **part_classification** may be determined by the site's business rules or by reference to a standard information model, such as a STEP application protocol.

8.3.11 PartMasterComposition

The **PartMasterComposition** relates the **PartRevision** to the **PartMaster** from which it is derived.

```
// PartMasterComposition Relationship
//   role : PartMasterToPartRevision
//     name : 'PartMasterToPartRevision'
//     entity : PartMaster
//     cardinality : 1
```

```
//   role : PartRevisionToPartMaster
//     name : 'PartRevisionToPartMaster'
//     entity : PartRevision
//     cardinality : 1..unbounded

interface PartMasterComposition :
  PdmFramework::PdmContainmentRelationship {};

interface PartMasterToPartRevision : PdmFramework::PdmContainsRole {};

interface PartRevisionToPartMaster  :
PdmFramework::PdmContainedInRole {};
```

8.3.12 PartRevision

The **PartRevision** represents a part that differs from other revision/change levels of the same part through a formal release or change but is a form, fit, and functional replacement for other revision/change levels.

```
interface PartRevision : PdmFramework::ItemRevision
{
  attribute string change_level;
};

interface PartRevisionFactory
{
  PartRevision create(
        in CosPropertyService::PropertySet property_set,
        in PartMaster part_master);
     raises (ITEM_CREATE_EXCEPTIONS);
};
```

change_level

Used to uniquely identify levels of change within a revision.

8.3.13 PartStructure

The **PartStructure** object represents an official revision of the product structure. It is concerned with only the component/assembly relationships of a part.

```
interface PartStructure : CosCompoundLifeCycle::Node,
  PdmFramework::Qualifiable {};

interface PartStructureFactory
{
  PartStructure create(
        in PartRevision part_revision);
     raises (ITEM_CREATE_EXCEPTIONS);
};
```

8.3.14 PartStructureIteration

The **PartStructureIteration** object captures the different iterations of the product structure.

```
interface PartStructureIteration : PdmFramework::ItemIteration
{
};

interface PartStructureIterationFactory
{
  PartStructureIteration create(
       in CosPropertyService::PropertySet property_set,
       in PartStructure part_structure);
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

8.3.15 PartStructureIterationRelationship

The **PartStructureIterationRelationship** relates the **PartStructure** to its iterations.

```
// PartStructureIterationRelationship Relationship
//   role : PartStructureToPartStructureIteration
//     name : 'PartStructureToPartStructureIteration'
//     entity : PartStructre
//     cardinality : 0..unbounded
//   role : PartStructureIterationToPartStructure
//     name : 'PartStructureIterationToPartStructure'
//     entity : PartStructureIteration
//     cardinality : 1..1

interface PartStructureIterationRelationship :
  PdmFramework::PdmContainmentRelationship {};

interface PartStructureToPartStructureIteration :
  PdmFramework::PdmContainsRole {};

interface PartStructureIterationToPartStructure :
  PdmFramework::PdmContainedInRole {};
```

8.3.16 PartStructureRelationship

The **PartStructureRelationship** relates the **PartRevision** to its **PartStructure** object.

```
// PartStructureRelationship Relationship
//   role : PartRevisionToPartStructure
//     name : 'PartRevisionToPartStructure'
//     entity : PartRevision
```

```
//     cardinality : 0..unbounded
//   role : PartStructureToPartRevision
//     name : 'PartStructureToPartRevision'
//     entity : PartStructure
//     cardinality : 1..1

interface PartStructureRelationship :
  PdmFramework::PdmContainmentRelationship {};

interface PartRevisionToPartStructure :
  PdmFramework::PdmContainsRole {};

interface PartStructureToPartRevision :
  PdmFramework::PdmContainedInRole {};
```

8.3.17 PartSupplierRelationship

The **PartSupplierRelationship** relates the **PartRevision** object to the **Party**, which plays the role of part supplier for this object.

```
// PartSupplierRelationship Relationship
//   role : PartSupplied
//     name : 'PartSupplied'
//     entity : PartRevision
//     cardinality : 0..unbounded
//   role : PartSupplier
//     name : 'PartSupplier'
//     entity : Party
//     cardinality : 0..unbounded

interface PartSupplierRelationship :
  PdmFramework::PdmReferenceRelationship {};

Interface PartSupplierRelationshipFactory
{
  PartSupplierRelationship create(
        in CosPropertyService::PropertySet property_set,
        in PartRevision part_supplied,
        in PdmResponsibility::Party part_supplier);
     raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface PartSupplied : PdmFramework::PdmReferencesRole {};

interface PartSupplier : PdmFramework::PdmReferencedByRole {};
```

8.3.18 PromissoryUsageOccurrence

A **PromissoryUsageOccurrence** is the relationship of a part to a higher assembly where the next higher assembly of the part has not been defined.

```
// PromissoryUsageOccurrence Relationship
//   role : Assembly (from AssemblyComponentUsage)
//     name : 'Assembly'
//     entity : PartStructureIteration
//     cardinality : 1..unbounded
//   role : Component (from ComponentUsage)
//     name : 'Component'
//     entity : PartMaster
//     cardinality : 0..unbounded

interface PromissoryUsageOccurrence : AssemblyComponentUsage {};

interface PromissoryUsageOccurrenceFactory
{
  PromissoryUsageOccurrence create(
        in CosPropertyService::PropertySet property_set,
        in PartStructureIteration assembly,
        in PartMaster component)
     raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};
```

8.3.19 Substitute

A **Substitute** is a component within an assembly whose form, fit, and function might be different from the component for which it is a replacement, but that can fulfill the requirements of another part within the context of the assembly. A substitute is specified by defining another usage relationship, the substitute, which can replace the original usage relationship, the base.

```
// Substitute Relationship
//   role : BaseUsage
//     name : 'BaseUsage'
//     entity : Usage
//     cardinality : 0..unbounded
//   role : SubstituteUsage
//     name : 'SubstituteUsage'
//     entity : Usage
//     cardinality : 0..unbounded

interface Substitute :
  PdmFramework::PdmReferenceRelationship
{
  attribute string name;
  attribute string description;
};
```

```
interface SubstituteFactory
{
  Substitute create(
       in CosPropertyService::PropertySet property_set,
       in Usage base_usage,
       in Usage substitute_usage);
    raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface BaseUsage : PdmFramework::PdmReferencesRole {};

interface SubstituteUsage : PdmFramework::PdmReferencedByRole {};
```

name

Provides the name of the **Substitute** object.

description

Contains text that describes the **Substitute** object.

8.3.20 Usage

A **Usage** object provides an association between an assembly's **PartStructureIteration** and its component **PartMaster**. The semantics of the association for a particular context is defined in specializations of this object.

```
// Usage Relationship
//   role : Assembly
//     name : 'Assembly'
//     entity : PartStructureIteration
//     cardinality : 1..unbounded
//   role : Component
//     name : 'Component'
//     entity : PartMaster
//     cardinality : 0..unbounded

interface Usage : CosCompoundLifeCycle::Node,
  PdmFramework::PdmReferenceRelationship,
  PdmFoundation::SecurityClassifiable,
  PdmFramework::Documentable
{
  attribute string id;
  attribute string name;
  attribute string description;
};

interface Assembly : PdmFramework::PdmReferencesRole {};
```

```
interface Component : PdmFramework::PdmReferencedByRole {};
```

id

Identifies the **Usage** object.

name

Provides the name of the **Usage** object.

description

Contains text that describes the nature of the association.

8.3.21 Make From Usage

A **MakeFromUsage** is a relationship between a **PartStructureIteration** and a **PartMaster** where the **PartStructureIteration** is derived from the **PartMaster**. The **MakeFromUsage** relationship can be used to indicate that the design of a particular part is derived from the design of another part or that a particular part is used as the basic for manufacturing of another part. The exact stage of life-cycle for which the **MakeFromUsage** relationship applies can be specified using **LifeCycleQualification**.

Note that the **Assembly** role indicates the **PartStructureIteration** object, which is derived from the **PartMaster** and the **Component** role indicates the **PartMaster** object from which the **PartStructureIteration** object is derived.

```
// MakeFromUsage Relationship
//    role : Assembly (from Usage)
//         name : 'Assembly'
//         entity : PartStructureIteration
//         cardinality : 1..unbounded
//    role : Component (from Usage)
//         name : 'Component'
//         entity : PartMaster
//         cardinality : 0..unbounded

interface MakeFromUsage : Usage
{
   attribute PdmFoundation::Measurement quantity;
};

interface MakeFromUsageFactory
{
   MakeFromUsage create(
         in CosPropertyService::PropertySet property_set,
         in PartMaster make_from_source,
         in PartStructureIteration make_from_target)
      raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};
```

quantity

The quantity attribute specifies how much or how many of the component (**PartMaster**) is needed to produce one unit of the assembly (**PartStructureIteration**).

8.4 PdmProductStructureDefinition IDL

```
// PdmProductStructureDefinition.idl

#ifndef  PDMPRODUCTSTRUCTUREDEFINITION
#define  PDMPRODUCTSTRUCTUREDEFINITION

#include <CosCompoundLifeCycle.idl>
#include <CosPropertyService.idl>

#include <PdmDocumentManagement.idl>
#include <PdmFramework.idl>
#include <PdmFoundation.idl>
#include <PdmResponsibility.idl>

module PdmProductStructureDefinition
{

// Exceptions

#define PDM_EXCEPTIONS \
   PdmFoundation::PdmError, \
   PdmFoundation::PermissionDenied, \
   PdmFoundation::ValidationError

#define ITEM_CREATE_EXCEPTIONS \
   PdmFoundation::PdmError, \
   PdmFoundation::PermissionDenied, \
   PdmFoundation::ValidationError, \
   PdmFoundation::InvalidProperties, \
   PdmFoundation::NotUnique

#define RELATIONSHIP_CREATE_EXCEPTIONS \
   PdmFoundation::PdmError, \
   PdmFoundation::PermissionDenied, \
   PdmFoundation::ValidationError, \
   PdmFoundation::InvalidProperties, \
   PdmFoundation::NotUnique, \
   PdmFoundation::CardinalityExceeded

// Forward References
interface PartRevision;

// Entities
```

```
interface PartData : CosCompoundLifeCycle::Node,
                     PdmFramework::Qualifiable
{
};

interface PartDataFactory
{
  PartData create(
    in PartRevision part_revision)
    raises (ITEM_CREATE_EXCEPTIONS);
};

interface PartDataIteration : PdmFramework::ItemIteration
{
};

interface PartDataIterationFactory
{
  PartDataIteration create(
    in CosPropertyService::PropertySet property_set,
    in PartData part_data)
    raises (ITEM_CREATE_EXCEPTIONS);
};

interface PartMaster : PdmFramework::ItemMaster
{
  attribute boolean standard_part;
  attribute string part_type;
  attribute string part_classification;
};

interface PartMasterFactory
{
  PartMaster create(
    in CosPropertyService::PropertySet property_set)
    raises (ITEM_CREATE_EXCEPTIONS);
};

interface PartRevision : PdmFramework::ItemRevision
{
  attribute string change_level;
};

interface PartRevisionFactory
{
  PartRevision create(
    in CosPropertyService::PropertySet property_set,
    in PartMaster part_master)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

```
interface PartStructure : CosCompoundLifeCycle::Node,
                          PdmFramework::Qualifiable
{
};

interface PartStructureFactory
{
  PartStructure create(
    in PartRevision part_revision)
    raises (ITEM_CREATE_EXCEPTIONS);
};

interface PartStructureIteration : PdmFramework::ItemIteration
{
};

interface PartStructureIterationFactory
{
  PartStructureIteration create(
    in CosPropertyService::PropertySet property_set,
    in PartStructure part_structure)
    raises (ITEM_CREATE_EXCEPTIONS);
};

// Relationships

// Usage Relationship
//   role : Assembly
//     name : 'Assembly'
//     entity : PartStructureIteration
//     cardinality : 1..unbounded
//   role : Component
//     name : 'Component'
//     entity : PartMaster
//     cardinality : 0..unbounded

interface Usage : CosCompoundLifeCycle::Node,
  PdmFramework::PdmReferenceRelationship,
  PdmFoundation::SecurityClassifiable,
  PdmFramework::Documentable
{
  attribute string id;
  attribute string name;
  attribute string description;
};

interface Assembly : PdmFramework::PdmReferencesRole {};

interface Component : PdmFramework::PdmReferencedByRole {};

// Alternate Relationship
```

```
//   role : BasePart
//     name : 'BasePart'
//     entity : PartMaster
//     cardinality : 0..unbounded
//   role : AlternatePart
//     name : 'AlternatePart'
//     entity : PartMaster
//     cardinality : 0..unbounded

interface Alternate : PdmFramework::PdmReferenceRelationship
{
  attribute string name;
  attribute string description;
  attribute string basis;
};

interface AlternateFactory
{
  Alternate create(
        in CosPropertyService::PropertySet property_set,
        in PartMaster alternate_part,
        in PartMaster base_part)
    raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface BasePart : PdmFramework::PdmReferencesRole {};

interface AlternatePart : PdmFramework::PdmReferencedByRole {};

// AssemblyComponentUsage Relationship
//   role : Assembly (from Usage)
//     name : 'Assembly'
//     entity : PartStructureIteration
//     cardinality : 1..unbounded
//   role : Component (from Usage)
//     name : 'Component'
//     entity : PartMaster
//     cardinality : 0..unbounded

interface AssemblyComponentUsage :  Usage
{
  attribute string reference_designator;
  attribute PdmFoundation::Measurement quantity;
};

// DesignSupplierRelationship Relationship
//   role : DesignSupplier
//     name : 'DesignSupplier'
//     entity : Party
//     cardinality : 0..unbounded
//   role : design_supplied
```

```
//     name : 'DesignSupplied'
//     entity : PartRevision
//     cardinality : 1..1

interface DesignSupplierRelationship :
   PdmFramework::PdmReferenceRelationship {};

interface DesignSupplierRelationshipFactory
{
   DesignSupplierRelationship create(
         in CosPropertyService::PropertySet property_set,
         in PartRevision supplied,
         in PdmResponsibility::Party supplier)
      raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface DesignSupplied : PdmFramework::PdmReferencesRole {};

interface DesignSupplier : PdmFramework::PdmReferencedByRole {};

// PartDocumentRelationship Relationship
//   role : PartRevisionOfPartDocument
// name : 'PartRevisionOfPartDocument'
// entity : PartRevision
// cardinality : 0..unbounded
//   role : DocumentRevisionOfPartDocument
// name : 'DocumentRevisionOfPartDocument'
// entity : PdmDocumentManagement::DocumentRevision
// cardinality : 0..unbounded

 interface PartDocumentRelationship :
      PdmFramework::PdmReferenceRelationship {};

 interface PartDocumentRelationshipFactory
 {
      PartDocumentRelationship create (
         in CosPropertyService::PropertySet property_set,
         in PartRevision part_revision,
         in PdmDocumentManagement::DocumentRevision document)
      raises (RELATIONSHIP_CREATE_EXCEPTIONS);
 };

 interface PartRevisionOfPartDocument :
PdmFramework::PdmReferencesRole {};

 interface DocumentRevisionOfPartDocument :
PdmFramework::PdmReferencedByRole {};
// NextAssemblyUsageOccurrence Relationship
//   role : Assembly (from AssemblyComponentUsage)
//     name : 'Assembly'
//     entity : PartStructureIteration
```

```
//     cardinality : 1..unbounded
//   role : Component (from AssemblyComponentUsage)
//     name : 'Component'
//     entity : PartMaster
//     cardinality : 0..unbounded

interface NextAssemblyUsageOccurrence : AssemblyComponentUsage {};

interface NextAssemblyUsageOccurrenceFactory
{
  NextAssemblyUsageOccurrence create(
        in CosPropertyService::PropertySet property_set,
        in PartStructureIteration assembly,
        in PartMaster component)
      raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

// PartDataIterationRelationship Relationship
//   role : PartDataToPartDataIteration
//     name : 'PartDataToPartDataIteration'
//     entity : PartData
//     cardinality : 0..unbounded
//   role : PartDataIterationToPartData
//     name : 'PartDataIterationToPartData'
//     entity : PartDataIteration
//     cardinality : 1..1

interface PartDataIterationRelationship :
  PdmFramework::PdmContainmentRelationship {};

interface PartDataToPartDataIteration :
  PdmFramework::PdmContainsRole {};

interface PartDataIterationToPartData :
  PdmFramework::PdmContainedInRole {};

interface PartDataToPartDataIteration :
  PdmFramework::PdmContainsRole {};

// PartDataRelationship Relationship
//   role : PartRevisionToPartData
//     name : 'PartRevisionToPartData'
//     entity : PartRevision
//     cardinality : 0..unbounded
//   role : PartDataToPartRevision
//     name : 'PartDataToPartRevision'
//     entity : PartData
//     cardinality : 1..1

interface PartDataRelationship :
  PdmFramework::PdmContainmentRelationship {};
```

```
interface PartRevisionToPartData : PdmFramework::PdmContainsRole {};

interface PartDataToPartRevision : PdmFramework::PdmContainedInRole
{};

// PartMasterComposition Relationship
//   role : PartMasterToPartRevision
//     name : 'PartMasterToPartRevision'
//     entity : PartMaster
//     cardinality : 1
//   role : PartRevisionToPartMaster
//     name : 'PartRevisionToPartMaster'
//     entity : PartRevision
//     cardinality : 1..unbounded

interface PartMasterComposition :
  PdmFramework::PdmContainmentRelationship {};


interface PartMasterToPartRevision : PdmFramework::PdmContainsRole {};

interface PartRevisionToPartMaster  :
  PdmFramework::PdmContainedInRole {};

// PartStructureIterationRelationship Relationship
//   role : PartStructureToPartStructureIteration
//     name : 'PartStructureToPartStructureIteration'
//     entity : PartStructre
//     cardinality : 0..unbounded
//   role : PartStructureIterationToPartStructure
//     name : 'PartStructureIterationToPartStructure'
//     entity : PartStructureIteration
//     cardinality : 1..1

interface PartStructureIterationRelationship :
  PdmFramework::PdmContainmentRelationship {};


interface PartStructureToPartStructureIteration :
  PdmFramework::PdmContainsRole {};

interface PartStructureIterationToPartStructure :
  PdmFramework::PdmContainedInRole {};

// PartStructureRelationship Relationship
//   role : PartRevisionToPartStructure
//     name : 'PartRevisionToPartStructure'
//     entity : PartRevision
//     cardinality : 0..unbounded
```

```
//   role : PartStructureToPartRevision
//     name : 'PartStructureToPartRevision'
//     entity : PartStructure
//     cardinality : 1..1

interface PartStructureRelationship :
   PdmFramework::PdmContainmentRelationship {};


interface PartRevisionToPartStructure : PdmFramework::PdmContainsRole
{};

interface PartStructureToPartRevision :
   PdmFramework::PdmContainedInRole {};

// PartSupplierRelationship Relationship
//   role : PartSupplied
//     name : 'PartSupplied'
//     entity : PartRevision
//     cardinality : 0..unbounded
//   role : PartSupplier
//     name : 'PartSupplier'
//     entity : Party
//     cardinality : 0..unbounded

interface PartSupplierRelationship :
   PdmFramework::PdmReferenceRelationship {};

interface PartSupplierRelationshipFactory
{
   PartSupplierRelationship create(
         in CosPropertyService::PropertySet property_set,
         in PartRevision part_supplied,
         in PdmResponsibility::Party part_supplier)
      raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface PartSupplied : PdmFramework::PdmReferencesRole {};

interface PartSupplier : PdmFramework::PdmReferencedByRole {};

// PromissoryUsageOccurrence Relationship
//   role : Assembly (from AssemblyComponentUsage)
//     name : 'Assembly'
//     entity : PartStructureIteration
//     cardinality : 1..unbounded
//   role : Component (from ComponentUsage)
//     name : 'Component'
//     entity : PartMaster
//     cardinality : 0..unbounded
```

```
interface PromissoryUsageOccurrence : AssemblyComponentUsage
{ };

interface PromissoryUsageOccurrenceFactory
{
  PromissoryUsageOccurrence create(
        in CosPropertyService::PropertySet property_set,
        in PartStructureIteration assembly,
        in PartMaster component)
    raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

// Substitute Relationship
//   role : BaseUsage
//     name : 'BaseUsage'
//     entity : Usage
//     cardinality : 0..unbounded
//   role : SubstituteUsage
//     name : 'SubstituteUsage'
//     entity : Usage
//     cardinality : 0..unbounded

interface Substitute :
  PdmFramework::PdmReferenceRelationship
{
  attribute string name;
  attribute string description;
};

interface SubstituteFactory
{
  Substitute create(
        in CosPropertyService::PropertySet property_set,
        in Usage base_usage,
        in Usage substitute_usage)
    raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface BaseUsage : PdmFramework::PdmReferencesRole {};

interface SubstituteUsage : PdmFramework::PdmReferencedByRole {};

// MakeFromUsage Relationship
//   role : Assembly (from Usage)
//       name : 'Assembly'
//       entity : PartStructureIteration
//       cardinality : 1..unbounded
//   role : Component (from Usage)
//       name : 'Component'
//       entity : PartMaster
//       cardinality : 0..unbounded
```

```
interface MakeFromUsage : Usage
{
   attribute PdmFoundation::Measurement quantity;
};

interface MakeFromUsageFactory
{
      MakeFromUsage create(
         in CosPropertyService::PropertySet property_set,
         in PartMaster make_from_source,
         in PartStructureIteration make_from_target)
        raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

#endif
```

PdmEffectivity Module 9

Contents

This chapter contains the following sections.

Section Title	Page
“Overview”	9-1
“PdmEffectivity Model”	9-2
“PdmEffectivity Description”	9-2
“PdmEffectivity IDL”	9-8

9.1 Overview

The Effectivity enablers object model is contained within the **PdmEffectivity** module. The **PdmEffectivity** module supports the effectivity of products and components. It relies on the **Qualification** and **PdmContext** concepts from the **PdmViews** module.

PdmEffectivity provides the ability to specify a range in which an item revision, component, or other qualified item should be used in production. Engineering usually specifies a planned effectivity, which manufacturing will use along with other information like the availability of parts to determine an actual effectivity.

An effectivity range can be expressed by dates during which the product is being manufactured, by the serial number of a product being manufactured, or by lot numbers for the products being manufactured.

Like other qualifications, Effectivity may be applied to any Qualified item, such as part revision, in which case it determines that the revision is used in that effectivity range. Or, it may be applied to a part structure usage relationship, in which case it determines that the component is used in the assembly in that range.

More than one effectivity range may be applied to the same item or relationship to indicate that it is effective in multiple discontinuous ranges.

9.2 PdmEffectivity Model

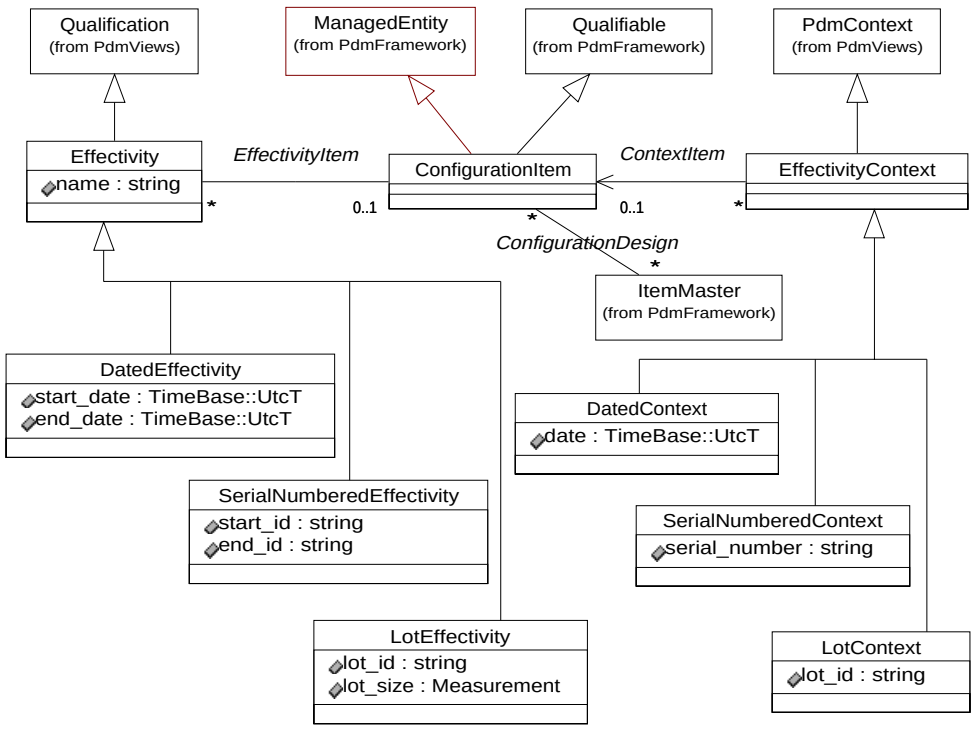

Figure 9-1 PdmEffectivity Model Diagram

9.3 PdmEffectivity Description

9.3.1 ConfigurationItem

The product that is planned for manufacture is referred to as the **ConfigurationItem**. It is usually visible to customers of the organization that does the configuration management.

A **ConfigurationItem** object is used to manage the composition of constituents for actual units of manufacture. All configuration management within an organization is done using these **ConfigurationItem**s. A **ConfigurationItem** can be an entire product concept or some portion of a product concept. This definition is taken from ISO10303-44.

A **ConfigurationItem** is referenced by **Effectivity** objects and **PdmContext** objects to specify the product for which the effectivity is being expressed.

```
interface ConfigurationItem :
  PdmFramework::ManagedEntity,
  PdmFramework::Qualifiable {};
```

9.3.2 *ConfigurationDesign Relationship*

Often, a **Configuration** item corresponds directly to a whole product or other high level item in a product structure, which indicates the top level of the design of the configuration item. The **ConfigurationDesign** relationship optionally relates the **ConfigurationItem** to an **ItemMaster**.

```
// ConfigurationDesign Relationship
//   role:  ConfigurationItemForDesign
//     name:  'ConfigurationItemForDesign'
//     entity:  ConfigurationItem
//     cardinality:  0..unbounded
//   role: DesignItemForConfiguration
//     name:  'DesignItemForConfiguration'
//     entity: PdmFramework::ItemMaster
//     cardinality:  0..unbounded

interface ConfigurationDesign:
  PdmFramework::PdmReferenceRelationship{};

interface ConfigurationDesignFactory
{
  ConfigurationDesign create(
        in CosPropertyService::PropertySet property_set,
        in ConfigurationItem configuration_item,
        in PdmFramework::ItemMaster design_item)
     raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ConfigurationItemForDesign :
  PdmFramework::PdmReferencesRole{};

interface DesignItemForConfiguration:
  PdmFramework::PdmReferencedByRole{};
```

9.3.3 *Effectivity*

An **Effectivity** is an abstract subclass of **Qualification**. It specifies that a relationship or object qualifies (is effective) under an effectivity constraint. The **Effectivity** object is specialized to support various kinds of effectivities.

Specifying a **ConfigurationItem** is necessary for **SerialNumberedEffectivity** and **LotEffectivity**, and may be optional for **DatedEffectivity**. When the **ConfigurationItem** is manufactured during the range of the **Effectivity**, the assigned **Qualified** items are “solutions” for the **ConfigurationItem**.

```
interface Effectivity : PdmViews::Qualification
{
  attribute string name;
};
```

name

Provides the name of the **Effectivity** object.

9.3.4 *EffectivityItem Relationship*

The **EffectivityItem** relationship specifies the product for which the effectivity is being expressed.

```
// EffectivityItem Relationship
//   role: EffectivityForItem
//     name:  'EffectivityForItem'
//     entity: Effectivity
//     cardinality:  0..unbounded
//   role:  ItemOfEffectivity
//     name:  'ItemOfEffectivity'
//     entity:  ConfigurationItem
//     cardinality:  0..1

interface EffectivityItem :
  PdmFramework::PdmReferenceRelationship,
  PdmFramework::Baselineable {};

interface EffectivityItemFactory
{
  EffectivityItem create(
        in CosPropertyService::PropertySet property_set,
        in Effectivity effectivity_for_item,
        in ConfigurationItem item_of_effectivity)
     raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface EffectivityForItem :
  PdmFramework::PdmReferencesRole {};
```

```
interface ItemOfEffectivity :
  PdmFramework::PdmReferencedByRole {};
```

9.3.5 DatedEffectivity

A **DatedEffectivity** is used to indicate that a **Qualified** item is effective while the configuration item is being produced during a date range.

DatedEffectivity is sometimes used to express other concerns than the production date of an item. For example, documents may have date effectivity for particular design purposes. A **DatedEffectivity** may be paired in a **CompoundQualification** with another type of **Qualification** that helps express the meaning of the **DatedEffectivity**.

A **start_date** must be given. If no **end_date** is given, the end date for the effectivity is not yet determined.

```
interface DatedEffectivity : Effectivity
{
  attribute TimeBase::UtcT start_date;
  attribute TimeBase::UtcT end_date;
};
interface DatedEffectivityFactory
{
  DatedEffectivity create(
       in CosPropertyService::PropertySet property_set)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

start_date

The date and/or time when the effectivity starts.

end_date

The date and/or time when the effectivity ends. If a value for this attribute is not defined, then the effectivity has no defined end.

9.3.6 LotEffectivity

A **LotEffectivity** is used to indicate that a **Qualified** item is effective when the configuration item is being produced in a specified lot.

```
interface LotEffectivity : Effectivity
{
  attribute string lot_id;
  attribute PdmFoundation::Measurement lot_size;
};
interface LotEffectivityFactory
```

```
{
  LotEffectivity create(
      in CosPropertyService::PropertySet property_set)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

lot_id

The identification of the batch of items that the effectivity applies to.

lot_size

The size of the batch of items that the effectivity applies to.

9.3.7 SerialNumberedEffectivity

A **SerialNumberedEffectivity** is used to indicate that a **Qualified** item is effective when the configuration item is being produced in a range of serial numbered units.

```
interface SerialNumberedEffectivity : Effectivity
{
  attribute string start_id;
  attribute string end_id;
};
interface SerialNumberedEffectivityFactory
{
  SerialNumberedEffectivity create(
      in CosPropertyService::PropertySet property_set)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

start_id

The serial number of the first item that the effectivity applies to.

end_id

The serial number of the last item that the effectivity applies to. If a value for this attribute is undefined, then the effectivity has no defined end.

9.3.8 EffectivityContext

An **EffectivityContext** is an abstract subclass of **PdmContext**. It specifies a particular effectivity-related context.

```
interface EffectivityContext : PdmViews::PdmContext
{
  attribute ConfigurationItem context_item;
};
```

context_item

Optional. The **ConfigurationItem** for which the context is expressed.

9.3.9 DatedContext

A **DatedContext** is a subclass of **EffectivityContext** that specifies a particular date and time. When a **DatedContext** is used in operations, only **Qualified** items with a **DatedEffectivity** whose date range includes the **DateContext** date are used for the purposes of the operation.

```
interface DatedContext : EffectivityContext
{
  attribute TimeBase::UtcT date;
};
interface DatedContextFactory
{
  DatedContext create(
        in ConfigurationItem context_item,
        in TimeBase::UtcT date)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

date

The date and/or time of the context.

9.3.10 SerialNumberedContext

A **SerialNumberedContext** is a subclass of **EffectivityContext** that specifies a specific serial number. When a **SerialNumberedContext** is used in operations, only **Qualified** items with a **SerialNumberedEffectivity** whose serial number range includes the **SerialNumberedContext serial_number** are used for the purposes of the operation.

```
interface SerialNumberedContext : EffectivityContext
{
  attribute string serial_number;
};

interface SerialNumberedContextFactory
{
  SerialNumberedContext create(
        in ConfigurationItem context_item,
        in string serial_number)
    raises (ITEM_CREATE_EXCEPTIONS);
};
```

serial_number

The serial number id of the particular item specified by the context.

9.3.10.1 LotContext

A **LotContext** is a subclass of **EffectivityContext** that specifies a specific lot id. When a **LotContext** is used in operations, only **Qualified** items with a **LotEffectivity** whose lot number equals the **LotContext lot_id** are used for the purposes of the operation.

```
interface LotContext : EffectivityContext
{
  attribute string lot_id;
};
interface LotContextFactory
{
  LotContext create(
        in ConfigurationItem context_item,
        in string lot_id)
     raises (ITEM_CREATE_EXCEPTIONS);
};
```

lot_id

The lot id of the particular batch of items specified by the context.

9.4 PdmEffectivity IDL

```
// PdmEffectivity.idl

#ifndef  PDMEFFECTIVITY
#define  PDMEFFECTIVITY

#include <CosPropertyService.idl>
#include <CosTime.idl>

#include <PdmFramework.idl>
#include <PdmViews.idl>

module PdmEffectivity
{

// Exceptions

#define PDM_EXCEPTIONS \
  PdmFoundation::PdmError, \
  PdmFoundation::PermissionDenied, \
  PdmFoundation::ValidationError
```

```
#define ITEM_CREATE_EXCEPTIONS \
   PdmFoundation::PdmError, \
   PdmFoundation::PermissionDenied, \
   PdmFoundation::ValidationError, \
   PdmFoundation::InvalidProperties, \
   PdmFoundation::NotUnique

#define RELATIONSHIP_CREATE_EXCEPTIONS \
   PdmFoundation::PdmError, \
   PdmFoundation::PermissionDenied, \
   PdmFoundation::ValidationError, \
   PdmFoundation::InvalidProperties, \
   PdmFoundation::NotUnique, \
   PdmFoundation::CardinalityExceeded

// Entities

interface ConfigurationItem :
      PdmFramework::ManagedEntity,
      PdmFramework::Qualifiable {};

interface Effectivity : PdmViews::Qualification
{
   attribute string name;
};

interface DatedEffectivity : Effectivity
{
   attribute TimeBase::UtcT start_date;
   attribute TimeBase::UtcT end_date;
};
interface DatedEffectivityFactory
{
   DatedEffectivity create(
      in CosPropertyService::PropertySet property_set)
      raises (ITEM_CREATE_EXCEPTIONS);
};

interface LotEffectivity : Effectivity
{
   attribute string lot_id;
   attribute PdmFoundation::Measurement lot_size;
};

interface LotEffectivityFactory
{
LotEffectivity create(
   in CosPropertyService::PropertySet property_set)
   raises (ITEM_CREATE_EXCEPTIONS);
};
```

```
interface SerialNumberedEffectivity : Effectivity
{
  attribute string start_id;
  attribute string end_id;
};
interface SerialNumberedEffectivityFactory
{
  SerialNumberedEffectivity create(
    in CosPropertyService::PropertySet property_set)
    raises (ITEM_CREATE_EXCEPTIONS);
};

interface EffectivityContext : PdmViews::PdmContext
{
  attribute ConfigurationItem context_item;
};

interface DatedContext : EffectivityContext
{
  attribute TimeBase::UtcT date;
};
interface DatedContextFactory
{
  DatedContext create(
      in ConfigurationItem context_item,
      in TimeBase::UtcT date)
    raises (ITEM_CREATE_EXCEPTIONS);
};

interface SerialNumberedContext : EffectivityContext
{
  attribute string serial_number;
};

interface SerialNumberedContextFactory
{
  SerialNumberedContext create(
      in ConfigurationItem context_item,
      in string serial_number)
    raises (ITEM_CREATE_EXCEPTIONS);
};

interface LotContext : EffectivityContext
{
  attribute string lot_id;
};
interface LotContextFactory
{
  LotContext create(
      in ConfigurationItem context_item,
      in string lot_id)
```

```
      raises (ITEM_CREATE_EXCEPTIONS);
};

// Relationships

// ConfigurationDesign Relationship
//   role:  ConfigurationItemForDesign
//     name:  'ConfigurationItemForDesign'
//     entity:  ConfigurationItem
//     cardinality:  0..unbounded
//   role: DesignItemForConfiguration
//     name:  'DesignItemForConfiguration'
//     entity: PdmFramework::ItemMaster
//     cardinality:  0..unbounded

interface ConfigurationDesign:
     PdmFramework::PdmReferenceRelationship{};

interface ConfigurationDesignFactory
{
  ConfigurationDesign create(
        in CosPropertyService::PropertySet property_set,
        in ConfigurationItem configuration_item,
        in PdmFramework::ItemMaster design_item)
     raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ConfigurationItemForDesign :
     PdmFramework::PdmReferencesRole{};

interface DesignItemForConfiguration:
     PdmFramework::PdmReferencedByRole{};

// EffectivityItem Relationship
//   role: EffectivityForItem
//     name:  'EffectivityForItem'
//     entity: Effectivity
//     cardinality:  0..unbounded
//   role:  ItemOfEffectivity
//     name:  'ItemOfEffectivity'
//     entity:  ConfigurationItem
//     cardinality:  0..1

interface EffectivityItem :
     PdmFramework::PdmReferenceRelationship,
     PdmFramework::Baselineable {};

interface EffectivityItemFactory
{
  EffectivityItem create(
        in CosPropertyService::PropertySet property_set,
```

```
        in Effectivity effectivity_for_item,
        in ConfigurationItem item_of_effectivity)
    raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface EffectivityForItem :
    PdmFramework::PdmReferencesRole{};

interface ItemOfEffectivity :
    PdmFramework::PdmReferencedByRole{};

};

#endif
```

PdmChangeManagement Module 10

Contents

This chapter contains the following sections.

Section Title	Page
“Overview”	10-1
“PdmChangeManagement Model”	10-3
“PdmChangeManagement Description”	10-4
“PdmChangeManagement IDL”	10-15

10.1 Overview

Engineering Change is the process by which companies request, implement, and effect change to products, documents, components, assemblies, manufactured or purchased parts, processes, or even suppliers. The expression Engineering Change so widely used refers to several separate processes:

1. issue collection
2. requesting change
3. implementing change
4. notification of change

These four processes are addressed in the model by the following classes, which are collectively referred to as Engineering Change Items:

- Engineering Change Issue (ECI)

- Engineering Change Request (ECR)
- Engineering Change Order (ECO)
- Engineering Change Notice (ECN)

Engineering change is concerned with communicating the state of an object, which is the condition of an object and its ability to invoke or permit certain actions and operations from other objects in the system. The concept of a state in the life cycle of an object is central to the engineering change process. The engineering change process tracks changes to objects and requests in the system. These changes may be dependent on one another and require objects to be able to communicate and understand the working state of other objects in the system. It is important to distinguish between the concept of a working state in a life cycle and the workflow aspects of a process.

An Engineering Change Item (ECI) affects Changeable Object(s). An Engineering Change Order (ECO) may be contingent on the prior implementation of some other ECO(s), in which case they are explicitly sequential (that is, there is a pre-requirement). Similarly, an ECO(s) may be contingent on the concurrent implementation of some other ECO(s), in which case they are explicitly concurrent (that is, there is a co-requirement). The same two possibilities may occur with Engineering Change Notices (ECN).

10.2 PdmChangeManagement Model

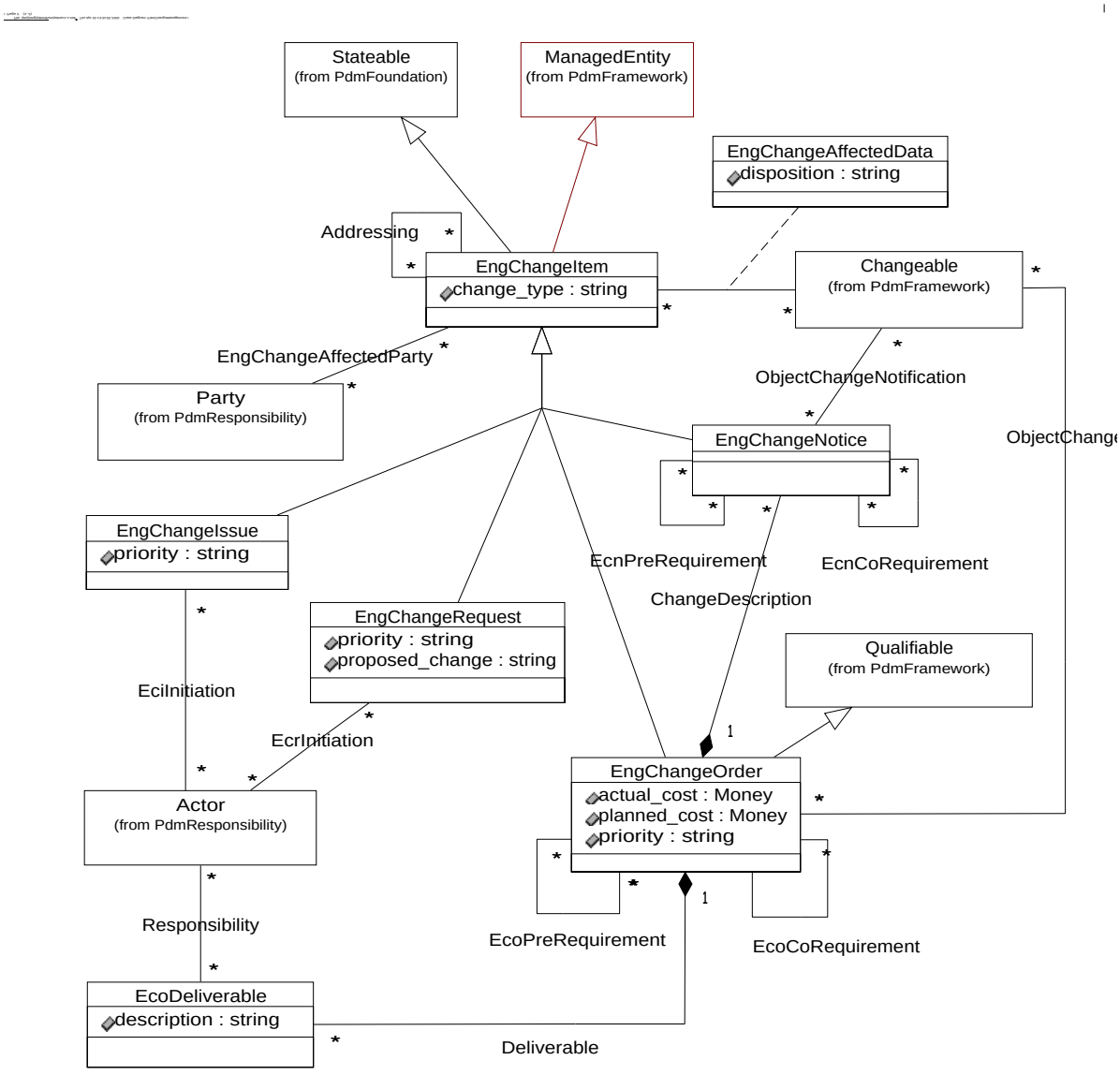

Figure 10-1 PdmChangeManagement Model Diagram

Note – EcoDeliverable inherits from **PdmFoundation::Stateable**, **CosTransactions::TransactionalObject**, **CosLifeCycle::LifeCycleObject**, and **CosCompoundLifeCycle::Node**.

10.3 PdmChangeManagement Description

10.3.1 EngChangeItem

EngChangeItem is the abstract parent object from which the four Engineering Change objects are derived.

An example categorization of **change_type** is adaptive, corrective, perfective, and preventive. An adaptive change maintains functionality for a different platform or environment. A corrective change corrects a defect. A perfective change adds functionality. A preventive change improves maintainability. Other categorizations, however, are possible.

```
interface EngChangeItem : PdmFramework::ManagedEntity,
  PdmFoundation::Stateable
{
  attribute string change_type;
  attribute string description;
};
```

change_type

Type of change (adaptive, corrective, perfective, preventive, etc.).

description

Provides a description of the **EngineeringChangeItem**.

10.3.2 EcoDeliverable

EcoDeliverable represents a single Engineering Change Order action item or task to be completed before the ECO is complete (that is, a work item).

```
interface EcoDeliverable : PdmFoundation::Stateable,
  CosTransactions::TransactionalObject,
  CosLifeCycle::LifeCycleObject,
  CosCompoundLifeCycle::Node
{
  attribute string description;
};

interface EcoDeliverableFactory
{
  EcoDeliverable create(in CosPropertyService::PropertySet property_set,
      in EngChangeOrder eco_work_item_directive)
    raises(ITEM_CREATE_EXCEPTIONS);
};
```

description

Provides a description of the **ECODeliverable** object.

10.3.3 EngChangeIssue

EngChangeIssue (ECI) represents an identified and collected engineering issue. The issue may come from a variety of sources (including other Engineering Change Items, manufacturing, distributors, customers) and the issue may result in one or more ECRs. In many organizations, ECIs are not filtered, that is, almost anyone can raise an ECI, for just about any reason.

```
interface EngChangeIssue : EngChangeItem
{
  attribute string priority;
};

interface EngChangeIssueFactory
{
  EngChangeIssue create(in CosPropertyService::PropertySet
      property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};
```

priority

Priority of Engineering Change Issue.

10.3.4 EngChangeNotice

EngChangeNotice (ECN) represents a notification to manufacturing, vendors, marketing, the field, or other entities, and it describes part or all of the implemented change of a single ECO. It is normal practice for the ECN to provide information also about the effectivity of the **EngChangeOrder**.

```
interface EngChangeNotice : EngChangeItem { };

interface EngChangeNoticeFactory
{
  EngChangeNotice create(
      in CosPropertyService::PropertySet property_set,
      in EngChangeOrder described_eco)
    raises(ITEM_CREATE_EXCEPTIONS);
};
```

10.3.5 EngChangeOrder

EngChangeOrder (ECO) represents a directive to implement an approved ECR. The deliverable items are referred to as ECO Deliverables.

```
interface EngChangeOrder : EngChangeItem, PdmFramework::Qualifiable
{
  attribute FbcCurrency::Money actual_cost;
  attribute FbcCurrency::Money planned_cost;
  attribute string priority;
};

interface EngChangeOrderFactory
{
  EngChangeOrder create(
       in CosPropertyService::PropertySet property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};
```

actual_cost

Actual cost of ECO.

planned_cost

Planned cost of ECO.

priority

Priority of ECO (e.g., urgent, normal, low).

10.3.6 EngChangeRequest

EngChangeRequest (ECR) represents a request for an engineering change, which normally addresses one or more ECIs. Usually, an ECR is created after one or more ECIs are evaluated and deemed to warrant, technically, a more formal treatment. In most organizations, a small group or organization proposes and prioritizes ECRs.

```
interface EngChangeRequest : EngChangeItem
{
  attribute string priority;
  attribute string proposed_change;
};

interface EngChangeRequestFactory
{
  EngChangeRequest create(in CosPropertyService::PropertySet
       property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};
```

priority

Priority of ECR.

proposed_change

Change proposed by ECR.

10.3.7 Addressing

This relationship provides a mechanism for relating any two instances derived from **EngChangeItem**, where one instance addresses another instance. For example, it may be employed to denote that an ECO addresses the ECR from which it resulted, that an ECN addresses the ECO from which it resulted. It may be employed also to denote that an ECI was raised in response to something within an ECR or ECO.

Due to the generic nature of the relationship, some of the potential relationships will probably not be utilized. That is, an ECO would not normally address an ECI, nor would an ECN address an ECR, since that would circumvent normal practice. Similarly, it seems unlikely that an ECI would address an ECN.

```
// Addressing Relationship
//   role: AddressingEngChangeItem
//     name: 'AddressingEngChangeItem'
//     entity: EngChangeItem
//     cardinality: 0..unbounded
//   role: AddressedEngChangeItem
//     name: 'AddressedEngChangeItem'
//     entity: EngChangeItem
//     cardinality: 0..unbounded

interface Addressing : PdmFramework::PdmReferenceRelationship { };

interface AddressingFactory
{
  Addressing create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeItem addressed_eng_change_item,
        in EngChangeItem addressing_eng_change_item)
    raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface AddressingEngChangeItem :
  PdmFramework::PdmReferencesRole { };

interface AddressedEngChangeItem :
  PdmFramework::PdmReferencedByRole { };
```

10.3.8 ChangeDescription

ChangeDescription relates an ECO to one or more ECNs. Combined, a set of ECNs should completely describe all changes implemented in an ECO. Since ECNs are sent to various audiences, any specific change in an ECO may be described by multiple ECNs.

```
// ChangeDescription Relationship
//   role: DescribingEcn
//     name: 'DescribingEcn'
//     entity: EngChangeNotice
//     cardinality: 1..1
//   role: DescribedEco
//     name: 'DescribedEco'
//     entity: EngChangeOrder
//     cardinality: 0..unbounded

interface ChangeDescription :
  PdmFramework::PdmContainmentRelationship { };

interface DescribingEcn : PdmFramework::PdmContainsRole { };

interface DescribedEco : PdmFramework::PdmContainedInRole { };
```

10.3.9 Deliverable

Changes implemented for an ECO are enumerated as a set of delivered items, or Deliverables. Combined, a set of **ECODeliverables** should completely enumerate and partition all changes to be implemented in an ECO.

```
// Deliverable Relationship
//   role: DeliverableEcoWorkItem
//     name: 'DeliverableEcoWorkItem'
//     entity: EcoDeliverable
//     cardinality: 1..1
//   role: EcoWorkItemDirective
//     name: 'EcoWorkItemDirective'
//     entity: EngChangeOrder
//     cardinality: 0..unbounded

interface Deliverable : PdmFramework::PdmContainmentRelationship { };

interface DeliverableEcoWorkItem : PdmFramework::PdmContainsRole { };

interface EcoWorkItemDirective : PdmFramework::PdmContainedInRole { };
```

10.3.10 EciInitiation

An **EciInitiation** relates an Engineering Change Issue to an **Actor** that initiated or reported the ECI.

```
// EciInitiation Relationship
//   role: InitiatedEci
//     name: 'InitiatedEci'
//     entity: EngChangeIssue
//     cardinality: 0..unbounded
```

```
//   role:  EciInitiator
//     name: 'EciInitiator'
//     entity: PdmResponsibility::Actor
//     cardinality: 0..unbounded

interface EciInitiation : PdmFramework::PdmReferenceRelationship { };

interface EciInitiationFactory
{
  EciInitiation create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeIssue initiated_eci,
        in PdmResponsibility::Actor eci_initiator)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface InitiatedEci : PdmFramework::PdmReferencesRole { };

interface EciInitiator : PdmFramework::PdmReferencedByRole { };
```

10.3.11 EcnCoRequirement

ECNs may be subject to co-requirements. Thus, all ECNs subject to a **CoRequirement** must be issued simultaneously.

```
// EcnCoRequirement Relationship
//   role: EcnCoRequirement1
//     name: 'EcnCoRequirement1'
//     entity: EngChangeNotice
//     cardinality: 0..unbounded
//   role: EcnCoRequirement2
//     name: 'EcnCoRequirement2'
//     entity: EngChangeNotice
//     cardinality: 0..unbounded

interface EcnCoRequirement :
  PdmFramework::PdmReferenceRelationship { };

interface EcnCoRequirementFactory
{
  EcnCoRequirement create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeNotice ecn_co_requirement_1,
        in EngChangeNotice ecn_co_requirement_2)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface EcnCoRequirement1 : PdmFramework::PdmReferencesRole { };

interface EcnCoRequirement2 : PdmFramework::PdmReferencedByRole { };
```

10.3.12 EcnPreRequirement

ECNs may be subject to sequential, or pre-, requirements. Thus, this relationship will establish the order in which ECNs are issued.

```
// EcnPreRequirement Relationship
//   role: EcnPredecessor
//     name: 'EcnPredecessor'
//     entity: EngChangeNotice
//     cardinality: 0..unbounded
//   role: EcnSuccessor
//     name: 'EcnSuccessor'
//     entity: EngChangeNotice
//     cardinality: 0..unbounded

interface EcnPreRequirement :
  PdmFramework::PdmReferenceRelationship { };

interface EcnPreRequirementFactory
{
  EcnPreRequirement create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeNotice ecn_predecessor,
        in EngChangeNotice ecn_successor)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface EcnPredecessor : PdmFramework::PdmReferencesRole { };

interface EcnSuccessor : PdmFramework::PdmReferencedByRole { };
```

10.3.13 EcoCoRequirement

ECOs may be subject to co-requirements. Thus, all ECOs subject to a **CoRequirement** must be implemented simultaneously.

```
// EcoCoRequirement Relationship
//   role: EcoCoRequirement1
//     name: 'EcoCorequirement1'
//     entity: EngChangeOrder
//     cardinality: 0..unbounded
//   role: EcoCoRequirement2
//     name: 'EcoCorequirement2'
//     entity: EngChangeOrder
//     cardinality: 0..unbounded

interface EcoCoRequirement :
PdmFramework::PdmReferenceRelationship { };

interface EcoCoRequirementFactory
```

```
{
  EcoCoRequirement create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeOrder eco_corequirement_1,
        in EngChangeOrder eco_corequirement_2)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface EcoCoRequirement1 : PdmFramework::PdmReferencesRole { };

interface EcoCoRequirement2 : PdmFramework::PdmReferencedByRole { };
```

10.3.14 EcoPreRequirement

ECOs may be subject to sequential, or pre-, requirements. Thus, the predecessor ECO must be implemented before the successor ECO.

```
// EcoPreRequirement Relationship
//   role: EcoPredecessor
//     name: 'EcoPredecessor'
//     entity: EngChangeOrder
//     cardinality: 0..unbounded
//   role: EcoSuccessor
//     name: 'Ecosuccessor'
//     entity: EngChangeOrder
//     cardinality: 0..unbounded

interface EcoPreRequirement :
  PdmFramework::PdmReferenceRelationship { };

interface EcoPreRequirementFactory
{
  EcoPreRequirement create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeOrder eco_predecessor,
        in EngChangeOrder eco_successor)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface EcoPredecessor : PdmFramework::PdmReferencesRole { };

interface EcoSuccessor : PdmFramework::PdmReferencedByRole { };
```

10.3.15 EcrInitiation

An **EcrInitiation** relates an Engineering Change Request to an **Actor** that initiated or reported the ECR.

```
// EcrInitiation Relationship
```

```
//   role: InitiatedEcr
//     name: 'InitiatedEcr'
//     entity: EngChangeRequest
//     cardinality: 0..unbounded
//   role: EcrInitiator
//     name: 'EcrInitiator'
//     entity: PdmResponsibility::Actor
//     cardinality: 0..unbounded

interface EcrInitiation : PdmFramework::PdmReferenceRelationship { };

interface EcrInitiationFactory
{
  EcrInitiation create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeRequest initiated_ecr,
        in PdmResponsibility::Actor ecr_initiator)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface InitiatedEcr : PdmFramework::PdmReferencesRole { };

interface EcrInitiator : PdmFramework::PdmReferencedByRole { };
```

10.3.16 EngChangeAffectedData

Engineering Change Items may affect Changeable objects with a stated disposition. An example of the applicability of such a relationship is an object that has been changed, and, in the normal course of events, has been replaced with a newer revision. However, not all **EngChangeItems** will. For example, the initiator of an ECI may not initially know exactly which object is affected. Examples of disposition include: scrap, rework, etc.

```
// EngChangeAffectedData Relationship
//   role: AffectingEngChangeItem
//     name: 'AffectingEngChangeItem'
//     entity: EngChangeItem
//     cardinality: 0..unbounded
//   role: AffectedChangeableData
//     name: 'AffectedChangeableData'
//     entity: PdmFramework::Changeable
//     cardinality: 0..unbounded

interface EngChangeAffectData :
  PdmFramework::PdmReferenceRelationship
{
  attribute string disposition;
};

interface EngChangeAffectedDataFactory
```

```
{
  EngChangeAffectedData create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeItem affecting_eng_change_item,
        in PdmFramework::Changeable affected_changeable_data)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface AffectingEngChangeItem :
  PdmFramework::PdmReferencesRole { };

interface AffectedChangeableData :
  PdmFramework::PdmReferencedByRole { };
```

disposition

Provides a disposition for **EngChangeItem** and the **ChangeableObj**.

10.3.17 EngChangeAffectedParty

EngChangeAffectedParty relates an **EngChangeItem** to a **Party** affected by the Change. This may be used to identify the people needing to be notified of a pending change, participate in approving a change, or receive a change notification when distributed.

```
// EngChangeAffectedParty Relationship
//   role: AffectingEngChange
//     name: 'AffectingEngChange'
//     entity: EngChangeItem
//     cardinality: 0..unbounded
//   role: AffectedParty
//     name: 'AffectedParty'
//     entity: PdmResponsibility::Party
//     cardinality: 0..unbounded

interface EngChangeAffectedParty :
  PdmFramework::PdmReferenceRelationship { };

interface EngChangeAffectedPartyFactory
{
  EngChangeAffectedParty create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeItem affecting_eng_change,
        in PdmResponsibility::Party affected_party)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface AffectingEngChange : PdmFramework::PdmReferencesRole { };

interface AffectedParty : PdmFramework::PdmReferencedByRole { };
```

10.3.18 ObjectChange

Each ECO may dictate changes in any number of **Changeable** objects via this relationship. The intention here is to the relationship between an ECO and the new revision of a **Changeable** object.

```
// ObjectChange Relationship
//   role: EcoChange
//     name: 'EcoChange'
//     entity: EngChangeOrder
//     cardinality: 0..unbounded
//   role: EcoChangedObject
//     name: 'EcoChangedObject'
//     entity: PdmFramework::Changeable
//     cardinality: 0..unbounded

interface ObjectChange : PdmFramework::PdmReferenceRelationship { };

interface ObjectChangeFactory
{
  ObjectChange create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeOrder eco_change,
        in PdmFramework::Changeable eco_changed_object)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface EcoChange : PdmFramework::PdmReferencesRole { };

interface EcoChangedObject : PdmFramework::PdmReferencedByRole { };
```

10.3.19 ObjectChangeNotification

Each ECN may provide notification about changes in any number of **Changeable** objects via this relationship.

```
// ObjectChangeNotification Relationship
//   role: EcnNotice
//     name: 'EcnNotice'
//     entity: EngChangeNotice
//     cardinality: 0..unbounded
//   role: EcnChangedObject
//     name: 'EcnChangedObject'
//     entity: PdmFramework::Changeable
//     cardinality: 0..unbounded

interface ObjectChangeNotification :
  PdmFramework::PdmReferenceRelationship { };

interface ObjectChangeNotificationFactory
```

```
{
  ObjectChangeNotification create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeNotice ecn_notice,
        in PdmFramework::Changeable ecn_changed_object)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface EcnNotice : PdmFramework::PdmReferencesRole { };

interface EcnChangedObject : PdmFramework::PdmReferencedByRole { };
```

10.3.20 Responsibility

Responsibility relates an **Actor** to the ECO Deliverables they are responsible for.

```
// Responsibility Relationship
//   role: OwnedEcoDeliverable
//     name: 'OwnedEcoDeliverable'
//     entity: EcoDeliverable
//     cardinality: 0..unbounded
//     name: 'OwnerOfEcoDeliverable'
//   role: OwnerOfEcoDeliverable
//     entity: PdmResponsibility::Actor
//     cardinality: 0..unbounded

interface Responsibility : PdmFramework::PdmReferenceRelationship { };

interface ResponsibilityFactory
{
  Responsibility create(
        in CosPropertyService::PropertySet property_set,
        in PdmResponsibility::Actor owner_of_eco_deliverable,
        in EcoDeliverable owned_eco_deliverable)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface OwnedEcoDeliverable : PdmFramework::PdmReferencesRole { };

interface OwnerOfEcoDeliverable : PdmFramework::PdmReferencedByRole
{ };
```

10.4 PdmChangeManagement IDL

```
// PdmChangeManagement.idl

#ifndef PDMCHANGEMANAGEMENT
#define PDMCHANGEMANAGEMENT

#ifdef   SOM_COMPILE
```

```
#include <somobj.idl>                    // SOM COMPILE
#endif

#ifdef  ORBIX_COMPILE
#ifndef  IFR
#define  IFR
#include <ifr.idl>
#endif
#endif

#include <CosCompoundLifeCycle.idl>
#include <CosLifeCycle.idl>
#include <CosPropertyService.idl>
#include <CosTransactions.idl>
#include <FbcCurrency.idl>

#include <PdmFoundation.idl>
#include <PdmFramework.idl>
#include <PdmResponsibility.idl>
#include <PdmViews.idl>

module PdmChangeManagement
{

// Exceptions

#define ITEM_CREATE_EXCEPTIONS \
     PdmFoundation::PdmError, \
     PdmFoundation::PermissionDenied, \
     PdmFoundation::ValidationError, \
     PdmFoundation::InvalidProperties, \
     PdmFoundation::NotUnique

#define RELATIONSHIP_CREATE_EXCEPTIONS \
     PdmFoundation::PdmError, \
     PdmFoundation::PermissionDenied, \
     PdmFoundation::ValidationError, \
     PdmFoundation::InvalidProperties, \
     PdmFoundation::NotUnique, \
     PdmFoundation::CardinalityExceeded

// Forward References
interface EngChangeOrder;

// Entities

interface EngChangeItem : PdmFramework::ManagedEntity,
     PdmFoundation::Stateable
{
  attribute string change_type;
  attribute string description;
```

```
};

interface EcoDeliverable : PdmFoundation::Stateable,
  CosTransactions::TransactionalObject,
  CosLifeCycle::LifeCycleObject,
  CosCompoundLifeCycle::Node
{
  attribute string description;
};

interface EcoDeliverableFactory
{
  EcoDeliverable create(in CosPropertyService::PropertySet
       property_set,
       in EngChangeOrder eco_work_item_directive)
    raises(ITEM_CREATE_EXCEPTIONS);
};

interface EngChangeIssue : EngChangeItem
{
  attribute string priority;
};

interface EngChangeIssueFactory
{
  EngChangeIssue create(in CosPropertyService::PropertySet
       property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};

interface EngChangeNotice : EngChangeItem { };

interface EngChangeNoticeFactory
{
  EngChangeNotice create(in CosPropertyService::PropertySet
       property_set,
       in EngChangeOrder described_eco)
    raises(ITEM_CREATE_EXCEPTIONS);
};

interface EngChangeOrder : EngChangeItem, PdmFramework::Qualifiable
{
  attribute FbcCurrency::Money actual_cost;
  attribute FbcCurrency::Money planned_cost;
  attribute string priority;
};

interface EngChangeOrderFactory
{
  EngChangeOrder create(in CosPropertyService::PropertySet
       property_set)
```

```
      raises(ITEM_CREATE_EXCEPTIONS);
};

interface EngChangeRequest : EngChangeItem
{
  attribute string priority;
  attribute string proposed_change;
};

interface EngChangeRequestFactory
{
  EngChangeRequest create(in CosPropertyService::PropertySet
       property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};

// Relationships

// Addressing Relationship
//   role: AddressingEngChangeItem
//     name: 'AddressingEngChangeItem'
//     entity: EngChangeItem
//     cardinality: 0..unbounded
//   role: AddressedEngChangeItem
//     name: 'AddressedEngChangeItem'
//     entity: EngChangeItem
//     cardinality: 0..unbounded

interface Addressing : PdmFramework::PdmReferenceRelationship { };

interface AddressingFactory
{
  Addressing create(
       in CosPropertyService::PropertySet property_set,
       in EngChangeItem addressed_eng_change_item,
       in EngChangeItem addressing_eng_change_item)
    raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface AddressingEngChangeItem :
PdmFramework::PdmReferencesRole
{ };

interface AddressedEngChangeItem :
PdmFramework::PdmReferencedByRole
{ };

// ChangeDescription Relationship
//   role: DescribingEcn
//     name: 'DescribingEcn'
//     entity: EngChangeNotice
```

```
//     cardinality: 1..1
//   role: DescribedEco
//     name: 'DescribedEco'
//     entity: EngChangeOrder
//     cardinality: 0..unbounded

interface ChangeDescription :
PdmFramework::PdmContainmentRelationship
{ };


interface DescribingEcn : PdmFramework::PdmContainsRole { };

interface DescribedEco : PdmFramework::PdmContainedInRole { };

// Deliverable Relationship
//   role: DeliverableEcoWorkItem
//     name: 'DeliverableEcoWorkItem'
//     entity: EcoDeliverable
//     cardinality: 1..1
//   role: EcoWorkItemDirective
//     name: 'EcoWorkItemDirective'
//     entity: EngChangeOrder
//     cardinality: 0..unbounded

interface Deliverable : PdmFramework::PdmContainmentRelationship { };


interface DeliverableEcoWorkItem : PdmFramework::PdmContainsRole { };

interface EcoWorkItemDirective : PdmFramework::PdmContainedInRole { };

// EciInitiation Relationship
//   role: InitiatedEci
//     name: 'InitiatedEci'
//     entity: EngChangeIssue
//     cardinality: 0..unbounded
//   role:  EciInitiator
//     name: 'EciInitiator'
//     entity: PdmResponsibility::Actor
//     cardinality: 0..unbounded

interface EciInitiation : PdmFramework::PdmReferenceRelationship { };

interface EciInitiationFactory
{
  EciInitiation create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeIssue initiated_eci,
        in PdmResponsibility::Actor eci_initiator)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
```

```
};

interface InitiatedEci : PdmFramework::PdmReferencesRole { };

interface EciInitiator : PdmFramework::PdmReferencedByRole { };

// EcnCoRequirement Relationship
//   role: EcnCoRequirement1
//     name: 'EcnCoRequirement1'
//     entity: EngChangeNotice
//     cardinality: 0..unbounded
//   role: EcnCoRequirement2
//     name: 'EcnCoRequirement2'
//     entity: EngChangeNotice
//     cardinality: 0..unbounded

interface EcnCoRequirement : PdmFramework::PdmReferenceRelationship
{ };

interface EcnCoRequirementFactory
{
  EcnCoRequirement create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeNotice ecn_co_requirement_1,
        in EngChangeNotice ecn_co_requirement_2)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface EcnCoRequirement1 : PdmFramework::PdmReferencesRole { };

interface EcnCoRequirement2 : PdmFramework::PdmReferencedByRole { };

// EcnPreRequirement Relationship
//   role: EcnPredecessor
//     name: 'EcnPredecessor'
//     entity: EngChangeNotice
//     cardinality: 0..unbounded
//   role: EcnSuccessor
//     name: 'EcnSuccessor'
//     entity: EngChangeNotice
//     cardinality: 0..unbounded

interface EcnPreRequirement : PdmFramework::PdmReferenceRelationship
{ };

interface EcnPreRequirementFactory
{
  EcnPreRequirement create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeNotice ecn_predecessor,
        in EngChangeNotice ecn_successor)
```

```
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface EcnPredecessor : PdmFramework::PdmReferencesRole { };

interface EcnSuccessor : PdmFramework::PdmReferencedByRole { };

// EcoCoRequirement Relationship
//   role: EcoCoRequirement1
//     name: 'EcoCorequirement1'
//     entity: EngChangeOrder
//     cardinality: 0..unbounded
//   role: EcoCoRequirement2
//     name: 'EcoCorequirement2'
//     entity: EngChangeOrder
//     cardinality: 0..unbounded

interface EcoCoRequirement : PdmFramework::PdmReferenceRelationship
{ };

interface EcoCoRequirementFactory
{
  EcoCoRequirement create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeOrder eco_corequirement_1,
        in EngChangeOrder eco_corequirement_2)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface EcoCoRequirement1 : PdmFramework::PdmReferencesRole { };

interface EcoCoRequirement2 : PdmFramework::PdmReferencedByRole { };

// EcoPreRequirement Relationship
//   role: EcoPredecessor
//     name: 'EcoPredecessor'
//     entity: EngChangeOrder
//     cardinality: 0..unbounded
//   role: EcoSuccessor
//     name: 'Ecosuccessor'
//     entity: EngChangeOrder
//     cardinality: 0..unbounded

interface EcoPreRequirement : PdmFramework::PdmReferenceRelationship
{ };

interface EcoPreRequirementFactory
{
  EcoPreRequirement create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeOrder eco_predecessor,
```

```
        in EngChangeOrder eco_successor)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface EcoPredecessor : PdmFramework::PdmReferencesRole { };

interface EcoSuccessor : PdmFramework::PdmReferencedByRole { };

// EcrInitiation Relationship
//   role: InitiatedEcr
//     name: 'InitiatedEcr'
//     entity: EngChangeRequest
//     cardinality: 0..unbounded
//   role: EcrInitiator
//     name: 'EcrInitiator'
//     entity: PdmResponsibility::Actor
//     cardinality: 0..unbounded

interface EcrInitiation : PdmFramework::PdmReferenceRelationship { };

interface EcrInitiationFactory
{
  EcrInitiation create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeRequest initiated_ecr,
        in PdmResponsibility::Actor ecr_initiator)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface InitiatedEcr : PdmFramework::PdmReferencesRole { };

interface EcrInitiator : PdmFramework::PdmReferencedByRole { };

// EngChangeAffectedData Relationship
//   role: AffectingEngChangeItem
//     name: 'AffectingEngChangeItem'
//     entity: EngChangeItem
//     cardinality: 0..unbounded
//   role: AffectedChangeableData
//     name: 'AffectedChangeableData'
//     entity: PdmFramework::Changeable
//     cardinality: 0..unbounded

interface EngChangeAffectedData :
PdmFramework::PdmReferenceRelationship
{
  attribute string disposition;
};

interface EngChangeAffectedDataFactory
{
```

```
  EngChangeAffectedData create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeItem affecting_eng_change_item,
        in PdmFramework::Changeable affected_changeable_data)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface AffectingEngChangeItem : PdmFramework::PdmReferencesRole
{ };

interface AffectedChangeableData :
PdmFramework::PdmReferencedByRole
{ };

// EngChangeAffectedParty Relationship
//   role: AffectingEngChange
//     name: 'AffectingEngChange'
//     entity: EngChangeItem
//     cardinality: 0..unbounded
//   role: AffectedParty
//     name: 'AffectedParty'
//     entity: PdmResponsibility::Party
//     cardinality: 0..unbounded

interface EngChangeAffectedParty :
  PdmFramework::PdmReferenceRelationship { };

interface EngChangeAffectedPartyFactory
{
  EngChangeAffectedParty create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeItem affecting_eng_change,
        in PdmResponsibility::Party affected_party)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface AffectingEngChange : PdmFramework::PdmReferencesRole { };

interface AffectedParty : PdmFramework::PdmReferencedByRole { };

// ObjectChange Relationship
//   role: EcoChange
//     name: 'EcoChange'
//     entity: EngChangeOrder
//     cardinality: 0..unbounded
//   role: EcoChangedObject
//     name: 'EcoChangedObject'
//     entity: PdmFramework::Changeable
//     cardinality: 0..unbounded

interface ObjectChange : PdmFramework::PdmReferenceRelationship { };
```

```
interface ObjectChangeFactory
{
  ObjectChange create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeOrder eco_change,
        in PdmFramework::Changeable eco_changed_object)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface EcoChange : PdmFramework::PdmReferencesRole { };

interface EcoChangedObject : PdmFramework::PdmReferencedByRole { };

// ObjectChangeNotification Relationship
//   role: EcnNotice
//     name: 'EcnNotice'
//     entity: EngChangeNotice
//     cardinality: 0..unbounded
//   role: EcnChangedObject
//     name: 'EcnChangedObject'
//     entity: PdmFramework::Changeable
//     cardinality: 0..unbounded

interface ObjectChangeNotification :
  PdmFramework::PdmReferenceRelationship { };

interface ObjectChangeNotificationFactory
{
  ObjectChangeNotification create(
        in CosPropertyService::PropertySet property_set,
        in EngChangeNotice ecn_notice,
        in PdmFramework::Changeable ecn_changed_object)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface EcnNotice : PdmFramework::PdmReferencesRole { };

interface EcnChangedObject : PdmFramework::PdmReferencedByRole { };

// Responsibility Relationship
//   role: OwnedEcoDeliverable
//     name: 'OwnedEcoDeliverable'
//     entity: EcoDeliverable
//     cardinality: 0..unbounded
//     name: 'OwnerOfEcoDeliverable'
//   role: OwnerOfEcoDeliverable
//     entity: PdmResponsibility::Actor
//     cardinality: 0..unbounded

interface Responsibility : PdmFramework::PdmReferenceRelationship { };
```

```
interface ResponsibilityFactory
{
  Responsibility create(
        in CosPropertyService::PropertySet property_set,
        in PdmResponsibility::Actor owner_of_eco_deliverable,
        in EcoDeliverable owned_eco_deliverable)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface OwnedEcoDeliverable : PdmFramework::PdmReferencesRole { };

interface OwnerOfEcoDeliverable : PdmFramework::PdmReferencedByRole
{ };

};

#endif
```

PdmManufacturingImplementation Module 11

Contents

This chapter contains the following sections.

Section Title	Page
“Overview”	11-1
“PdmManufacturingImplementation Model”	11-2
“PdmManufacturingImplementation Description”	11-3
“PdmManufacturingImplementation IDL”	11-12

11.1 Overview

This chapter describes the objects that support the manufacturing process specification and the relationships between the process specifications, items produced, manufacturing location, engineering change orders. The use of **ProcessMaster** and **ProcessOperation** objects support manufacturing views that are not possible if parts are the only constructs. For example, two process plans for the same item can be distinguished by the manufacturing location. Further, data such as NC programs and tools used in a manufacturing process, whose association with a part would be ambiguous, can be directly related to the appropriate process operation.

This model primarily supports the process requirements of the discrete manufacturing industry. The intent is not to specify the level of detail that would be required by a Manufacturing Execution System (MES) for controlling manufacturing on the plant floor. However, we feel that there must be sufficient detail to allow for assessment of manufacturing feasibility and for estimating the cost of manufacturing.

11.2 PdmManufacturingImplementation Model

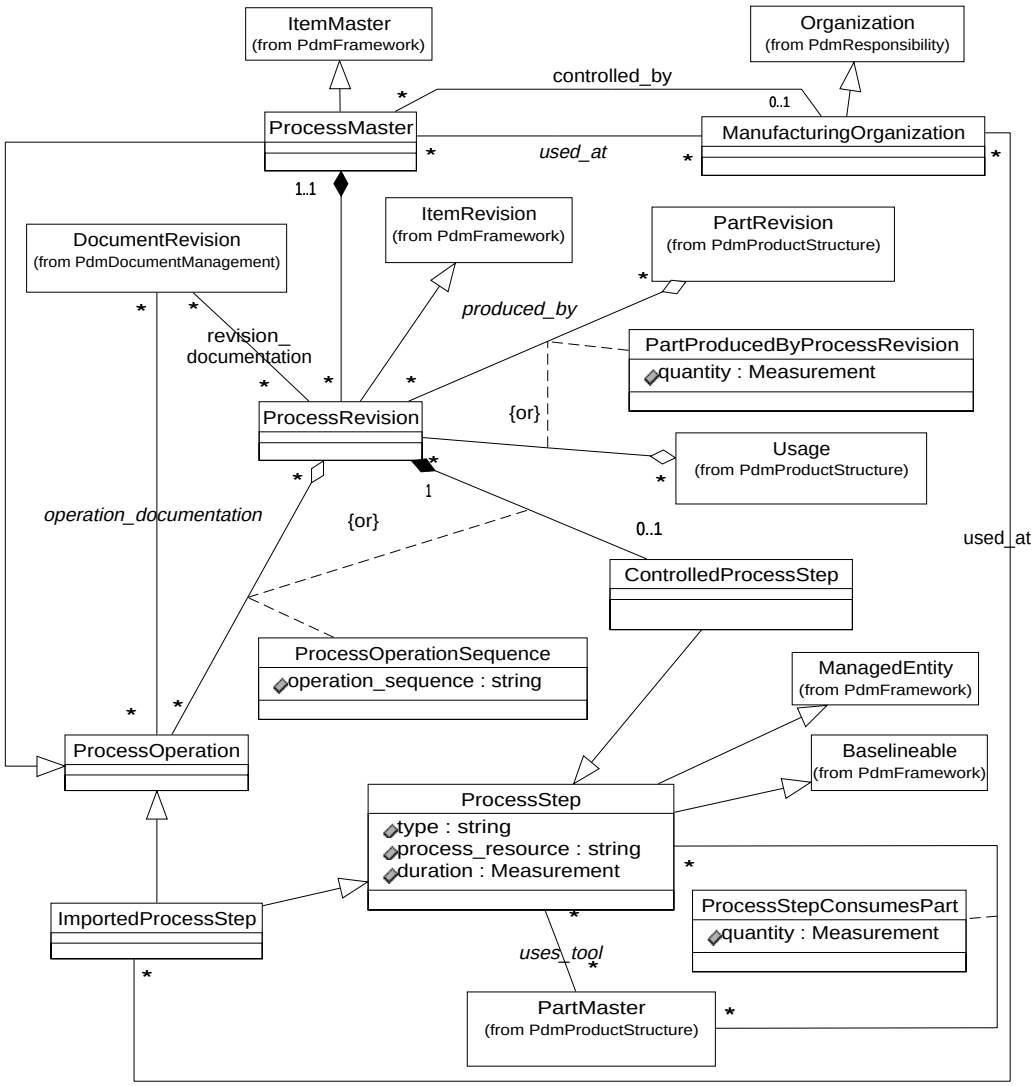

Figure 11-1 PdmManufacturingImplementation Model Diagram

11.3 PdmManufacturingImplementation Description

11.3.1 ProcessMaster

The **ProcessMaster** object is the highest level object and is used to collect a set of requirements for the definition of a manufacturing process. It associates the process with the manufacturing location that uses the process. The **ProcessMaster** inherits from the **ProcessOperation** to allow the definition of a hierarchy of process operations.

```
interface ProcessMaster : PdmFramework::ItemMaster,
  ProcessOperation { };

interface ProcessMasterFactory
{
ProcessMaster create( in CosPropertyService::PropertySet
     property_set )
  raises( ITEM_CREATE_EXCEPTIONS );
};
```

11.3.2 ProcessMasterControlledByOrganization Relationship

The **ProcessMasterControlledByOrganization** relationship object relates a process master definition with the manufacturing organization that has control over the process definition.

```
// ProcessMasterControlledByOrganization Relationship
//   role:  ProcessMasterControllingOrganization
//     name:  'ProcessMasterControllingOrganization'
//     entity:  ProcessMaster
//     cardinality:  0..1
//   role:  OrganizationControllingProcessMaster
//     name:  'OrganizationControllingProcessMaster'
//     entity:  ManufacturingOrganization
//     cardinality:  0..unbounded

interface ProcessMasterControlledByOrganization :
  PdmFramework::PdmReferenceRelationship { };

interface ProcessMasterControlledByOrganizationFactory
{
  ProcessMasterControlledByOrganization create(
       in CosPropertyService::PropertySet property_set,
       in ProcessMaster the_master,
       in ManufacturingOrganization the_organization)
    raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

interface ProcessMasterControllingOrganization :
```

```
  PdmFramework::PdmReferencesRole {};

interface OrganizationControllingProcessMaster :
  PdmFramework::PdmReferencedByRole {};
```

11.3.3 ProcessMasterComposition Relationship

The **ProcessMasterComposition** relationship is used to associate a process master with all of the revisions that have been created based on the process master.

```
// ProcessMasterComposition Relationship
//   role:  ProcessMasterForRevision
//     name:  'ProcessMasterForRevision'
//     entity:  ProcessMaster
//     cardinality:  0..unbounded
//   role:  ProcessRevisionForMaster
//     name:  'ProcessRevisionForMaster'
//     entity:  ProcessRevision
//     cardinality:  1..1

interface ProcessMasterComposition :
  PdmFramework::PdmContainmentRelationship {};

interface ProcessMasterForRevision :
  PdmFramework::PdmContainsRole {};

interface ProcessRevisionForMaster :
  PdmFramework::PdmContainedInRole {};
```

11.3.4 ProcessMasterUsedAtOrganization Relationship

Used to identify which organizations use a process definition.

```
// ProcessMasterUsedAtOrganization Relationship
//   role:  ProcessMasterForUsingOrganization
//     name:  'ProcessMasterForUsingOrganization'
//     entity:  ProcessMaster
//     cardinality:  0..unbounded
//   role:  OrganizationUsingProcessMaster
//     name:  'OrganizationUsingProcessMaster'
//     entity:  ManufacturingOrganization
//     cardinality:  0..unbounded

interface ProcessMasterUsedAtOrganization :
  PdmFramework::PdmReferenceRelationship {};

interface ProcessMasterUsedAtOrganizationFactory
{
  ProcessMasterControlledByOrganization create(
```

```
        in CosPropertyService::PropertySet property_set,
        in ProcessMaster the_master,
        in ManufacturingOrganization the_organization)
      raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

interface ProcessMasterForUsingOrganization :
  PdmFramework::PdmReferencesRole {};

interface OrganizationUsingProcessMaster :
  PdmFramework::PdmReferencedByRole {};
```

11.3.5 ProcessRevision

The **ProcessRevision** object allows multiple configurations of a manufacturing process to be defined. It is the level at which a manufacturing process is configuration controlled. It is also the granularity at which a process is documented.

```
interface ProcessRevision : PdmFramework::ItemRevision {};

interface ProcessRevisionFactory
{
  ProcessRevision create(
        in CosPropertyService::PropertySet property_set,
        in ProcessMaster the_master                    )
      raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};
```

11.3.6 ProcessRevisionComposition Relationship

Supports the revision control of a process step. This is used for process steps that are revision controlled by the PDM system. Note, only a single controlled process step may be defined in a process revision; this is because a process definition is defined as a sequence of steps using the **ProcessOperationSequence** aggregate relationship.

```
// ProcessRevisionComposition Relationship
//   role:  ProcessRevisionForControlledProcessStep
//     name:  'ProcessRevisionForControlledProcessStep'
//     entity:  ProcessRevision
//     cardinality:  0..1
//   role:  ControlledProcessStepForRevision
//     name:  'ControlledProcessStepForRevision'
//     entity:  ControlledProcessStep
//     cardinality:  1..1

interface ProcessRevisionComposition :
  PdmFramework::PdmContainmentRelationship {};

interface ProcessRevisionForControlledProcessStep :
```

```
  PdmFramework::PdmContainsRole {};

interface ControlledProcessStepForRevision :
  PdmFramework::PdmContainedInRole {};
```

11.3.7 ProcessRevisionDocumentation Relationship

This relationship associates the process revision with the set of documents that define the detailed process.

```
// ProcessRevisionDocumentation Relationship
//   role:  ProcessRevisionForDocument
//     name:  'ProcessRevisionForDocument'
//     entity:  ProcessRevision
//     cardinality:  0..unbounded
//   role:  ProcessDocumentForRevision
//     name:  'ProcessDocumentForRevision'
//     entity:  PdmDocumentManagement::DocumentRevision
//     cardinality:  0..unbounded

interface ProcessRevisionDocumentation :
  PdmFramework::PdmReferenceRelationship {};

interface ProcessRevisionDocumentationFactory
{
  ProcessRevisionDocumentation create(
        in CosPropertyService::PropertySet property_set,
        in ProcessRevision the_revision,
        in PdmDocumentManagement::DocumentRevision the_document)
     raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

interface ProcessRevisionForDocument :
  PdmFramework::PdmReferencesRole {};

interface ProcessDocumentForRevision :
  PdmFramework::PdmReferencedByRole {};
```

11.3.8 PartProducedByProcessRevision Relationship

This relationship is used to track a set of process revision objects used to produce a particular revision of a part or a particular usage relationship.

```
// PartProducedByProcessRevision Relationship
//   role : ProducedPart
//     name : 'ProducedPart'
//     entity :  PdmProductStructureDefinition::PartRevision
//     cardinality : 0..unbounded
//   role : ProducedUsage
```

```
//     name : 'ProducedUsage'
//     entity : PdmProductStructure::Usage
//     cardinality : 0..unbounded
//   role : ProducingProcess
//     name : 'ProducingProcess'
//     entity : ProcessRevision
//     cardinality : 0..unbounded

interface PartProducedByProcessRevision :
  PdmFramework::PdmReferenceRelationship
{
  attribute PdmFoundation::Measurement quantity;
};

interface PartProducedByProcessRevisionFactory
{
  PartProducedByProcessRevision create_using_part_revision(
        in CosPropertyService::PropertySet property_set,
        in ProcessRevision producing_process,
        in PdmProductStructureDefinition::Usage  produced_usage)
     raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

interface ProducedPart : PdmFramework::PdmReferencesRole {};

interface ProductUsage : PdmFramework::PdmReferencesRole {};

interface ProducingProcess : PdmFramework::PdmReferencedByRole {};
```

quantity

The quantity of parts that are produced by the process.

11.3.9 ProcessOperation

The **ProcessOperation** is an abstract class. It is used to allow both **ProcessMaster** and **ProcessStep** objects to be used in the definition of a **ProcessRevision**. This supports a multi-level process definition much like a bill of materials for a product assembly. A sequencing of the **ProcessOperation** may be defined but it is not used by the PDM.

```
interface ProcessOperation { };
```

11.3.10 ProcessOperationDocumentation Relationship

The **ProcessOperationDocumentation** relationship is used to capture the set of documents that describe the specifics of a process operation.

```
// ProcessOperationDocumentation Relationship
//   role:  ProcessOperationForDocument
```

```
//     name:  'ProcessOperationForDocument'
//     entity:  ProcessOperation
//     cardinality:  0..unbounded
//   role:  ProcessDocumentForOperation
//     name:  'ProcessDocumentForOperation'
//     entity:  PdmDocumentManagement::DocumentRevision
//     cardinality:  0..unbounded

interface ProcessOperationDocumentation :
  PdmFramework::PdmReferenceRelationship {};

interface ProcessOperationDocumentationFactory
{
  ProcessOperationDocumentation create(
        in CosPropertyService::PropertySet property_set,
        in ProcessOperation the_operation,
        in PdmDocumentManagement::DocumentRevision the_document)
     raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

interface ProcessOperationForDocument :
  PdmFramework::PdmReferencesRole {};

interface ProcessDocumentForOperation :
  PdmFramework::PdmReferencedByRole {};
```

11.3.11 ProcessOperationSequence Relationship

This relationship object is used to track the sequence of operations within a process revision. A relationship object is used so that a process operation may be used in multiple processes at a given manufacturing location.

```
// ProcessOperationSequence Relationship
//   role:  ProcessRevisionForOperation
//     name:  'ProcessRevisionForOperation'
//     entity:  ProcessRevision
//     cardinality:  0..unbounded
//   role:  ProcessOperationForRevision
//     name:  'ProcessOperationForRevision'
//     entity:  ProcessOperation
//     cardinality:  0..unbounded

interface ProcessOperationSequence :
  PdmFramework::PdmReferenceRelationship
{
  attribute string operation_sequence;
};

interface ProcessOperationSequenceFactory
{
```

```
  ProcessOperationSequence create(
        in CosPropertyService::PropertySet property_set,
        in ProcessOperation the_operation,
        in ProcessRevision the_revision)
      raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

interface ProcessRevisionForOperation :
  PdmFramework::PdmReferencesRole {};

interface ProcessOperationForRevision :
  PdmFramework::PdmReferencedByRole {};
```

sequence

This is the order, relative to the other process operations for this process revision, in which this process operation is performed. It is assumed to be unique; however, it is not interpreted by the PDM.

11.3.12 ProcessStep

The **ProcessStep** is an abstract class that is used to describe the individual steps that comprise a process. Actual instantiations of process steps will use the **ControlledProcessStep** or **ImportedProcessStep** interfaces. It is at this level that one documents the parts used in a process operation, the tools required, and the location where the step is executed. Changes to a process are also controlled at this level.

```
interface ProcessStep : PdmFramework::ManagedEntity,
  PdmFramework::Baselineable
{
  attribute string type;
  attribute string process_resource;
  attribute PdmFoundation::Measurement  duration;
};
```

type

Used to optionally describe the operation being performed. For example, machining, degreasing.

process_resource

The name of a processing resource (or resource type) that will perform the processing step (e.g., ‘VMC-4’). This name may be used as an identifier for a Resource object whose interface is provided elsewhere.

duration

The estimated time to perform the operation.

11.3.13 ProcessStepUsesTool

Identifies the tools that are used by this process step (e.g., deburrer, drill).

```
// ProcessStepUsesTool Relationship
//   role:  ProcessStepForTool
//     name:  'ProcessStepForTool'
//     entity:  ProcessStep
//     cardinality:  0..unbounded
//   role:  ToolForProcessStep
//     name:  'ToolForProcessStep'
//     entity:  PdmConfigurationManagement::PartMaster
//     cardinality:  0..unbounded

interface ProcessStepUsesTool :
  PdmFramework::PdmReferenceRelationship {};

interface ProcessStepUsesToolFactory
{
  ProcessStepUsesTool create(
        in CosPropertyService::PropertySet property_set,
        in ProcessStep the_step,
        in PdmProductStructureDefinition::PartMaster the_tool)
    raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

interface ProcessStepForTool : PdmFramework::PdmReferencesRole {};

interface ToolForProcessStep : PdmFramework::PdmReferencedByRole {};
```

11.3.14 ProcessStepConsumesPart

Identifies a component that is consumed by this process step. For example, a sheet of aluminum might be drilled to produce a new part; the sheet of aluminum is consumed by this operation.

```
// ProcessStepConsumesPart Relationship
//   role:  ProcessStepForPart
//     name:  'ProcessStepForPart'
//     entity:  ProcessStep
//     cardinality:  0..unbounded
//   role:  PartForProcessStep
//     name:  'PartForProcessStep'
//     entity:  PdmProductStructureDefinition::PartMaster
//     cardinality:  0..unbounded

interface ProcessStepConsumesPart :
  PdmFramework::PdmReferenceRelationship
{
  attribute PdmFoundation::Measurement quantity;
```

```
};

interface ProcessStepConsumesPartFactory
{
  ProcessStepConsumesPart create(
        in CosPropertyService::PropertySet property_set,
        in ProcessStep the_step,
        in PdmProductStructureDefinition::PartMaster the_part)
    raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

interface ProcessStepForPart : PdmFramework::PdmReferencesRole {};

interface PartForProcessStep : PdmFramework::PdmReferencedByRole {};
```

quantity

The quantity of parts that are consumed by the process.

11.3.15 ControlledProcessStep

Identifies a process step that is revision controlled by the PDM system. Thus, it will have a **ProcessMaster** and a **ProcessRevision**.

```
interface ControlledProcessStep : ProcessStep { };

interface ControlledProcessStepFactory
{
  ControlledProcessStep create(
        in CosPropertyService::PropertySet property_set,
        in ProcessRevision the_revision              )
    raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};
```

11.3.16 ImportedProcessStep

Identifies a process step that is revision controlled outside the PDM system. This type of process step will not have an associated **ProcessMaster** or **Revision**.

```
interface ImportedProcessStep : ProcessOperation, ProcessStep { };

interface ImportedProcessStepFactory
{
  ImportedProcessStep create(
        in CosPropertyService::PropertySet property_set )
    raises( ITEM_CREATE_EXCEPTIONS );
};
```

11.3.17 ImportedProcessStepUsedAtOrganization Relationship

Identifies which organizations use a process step that is not revision controlled by the PDM system.

```
// ImportedProcessStepUsedAtOrganization Relationship
//   role:  ImportedProcessStepForOrganization
//     name:  'ImportedProcessStepForOrganization'
//     entity:  ImportedProcessStep
//     cardinality:  0..unbounded
//   role:  OrganizationForImportedProcessStep
//     name:  'OrganizationForImportedProcessStep'
//     entity:  ManufacturingOrganization
//     cardinality:  0..unbounded

interface ImportedProcessStepUsedAtOrganization :
  PdmFramework::PdmReferenceRelationship {};

interface ImportedProcessStepUsedAtOrganizationFactory
{
  ImportedProcessStepUsedAtOrganization create(
        in CosPropertyService::PropertySet property_set,
        in ImportedProcessStep the_step,
        in ManufacturingOrganization the_organization)
      raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

interface ImportedProcessStepForOrganization :
  PdmFramework::PdmReferencesRole {};

interface OrganizationForImportedProcessStep :
  PdmFramework::PdmReferencedByRole {};
```

11.3.18 ManufacturingOrganization

```
interface ManufacturingOrganization : PdmResponsibility::Organization {};

interface ManufacturingOrganizationFactory
{
  ManufacturingOrganization create(
        in CosPropertyService::PropertySet property_set )
      raises( ITEM_CREATE_EXCEPTIONS );
};
```

11.4 PdmManufacturingImplementation IDL

```
// PdmManufacturingImplementation.idl

#ifndef  PDMMANUFACTURINGIMPLEMENTATION
```

```
#define  PDMMANUFACTURINGIMPLEMENTATION

#include <CosPropertyService.idl>

#include <PdmChangeManagement.idl>
#include <PdmDocumentManagement.idl>
#include <PdmFoundation.idl>
#include <PdmFramework.idl>
#include <PdmResponsibility.idl>
#include <PdmProductStructureDefinition.idl>

module PdmManufacturingImplementation
{

interface ProcessRevision;

// Exceptions

#define PDM_EXCEPTIONS \
     PdmFoundation::PdmError, \
     PdmFoundation::PermissionDenied, \
     PdmFoundation::ValidationError

#define ITEM_CREATE_EXCEPTIONS \
     PdmFoundation::PdmError, \
     PdmFoundation::PermissionDenied, \
     PdmFoundation::ValidationError, \
     PdmFoundation::InvalidProperties, \
     PdmFoundation::NotUnique

#define RELATIONSHIP_CREATE_EXCEPTIONS \
     PdmFoundation::PdmError, \
     PdmFoundation::PermissionDenied, \
     PdmFoundation::ValidationError, \
     PdmFoundation::InvalidProperties, \
     PdmFoundation::NotUnique, \
     PdmFoundation::CardinalityExceeded



// Entities

interface ProcessOperation
{
};

interface ProcessStep : PdmFramework::ManagedEntity,
     PdmFramework::Baselineable
{
  attribute string type;
  attribute string process_resource;
```

```
  attribute PdmFoundation::Measurement  duration;
};

interface ControlledProcessStep : ProcessStep
{
};

interface ControlledProcessStepFactory
{
  ControlledProcessStep create(
       in CosPropertyService::PropertySet property_set,
       in ProcessRevision the_revision              )
    raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

interface ImportedProcessStep : ProcessOperation, ProcessStep
{
};

interface ImportedProcessStepFactory
{
  ImportedProcessStep create(
       in CosPropertyService::PropertySet property_set )
    raises( ITEM_CREATE_EXCEPTIONS );
};

interface ManufacturingOrganization : PdmResponsibility::Organization
{
};

interface ManufacturingOrganizationFactory
{
  ManufacturingOrganization create(
       in CosPropertyService::PropertySet property_set )
    raises( ITEM_CREATE_EXCEPTIONS );
};

interface ProcessMaster : PdmFramework::ItemMaster,
    ProcessOperation
{
};

interface ProcessMasterFactory
{
  ProcessMaster create( in CosPropertyService::PropertySet
       property_set )
    raises( ITEM_CREATE_EXCEPTIONS );
};

interface ProcessRevision : PdmFramework::ItemRevision
{
```

```
};

interface ProcessRevisionFactory
{
  ProcessRevision create(
        in CosPropertyService::PropertySet property_set,
        in ProcessMaster the_master                    )
     raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

// Relationships

// ImportedProcessStepUsedAtOrganization Relationship
//   role:  ImportedProcessStepForOrganization
//     name:  'ImportedProcessStepForOrganization'
//     entity:  ImportedProcessStep
//     cardinality:  0..unbounded
//   role:  OrganizationForImportedProcessStep
//     name:  'OrganizationForImportedProcessStep'
//     entity:  ManufacturingOrganization
//     cardinality:  0..unbounded

interface ImportedProcessStepUsedAtOrganization :
     PdmFramework::PdmReferenceRelationship
{
};

interface ImportedProcessStepUsedAtOrganizationFactory
{
  ImportedProcessStepUsedAtOrganization create(
        in CosPropertyService::PropertySet property_set,
        in ImportedProcessStep the_step,
        in ManufacturingOrganization the_organization)
     raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

interface ImportedProcessStepForOrganization :
     PdmFramework::PdmReferencesRole
{
};

interface OrganizationForImportedProcessStep :
     PdmFramework::PdmReferencedByRole
{
};

// PartProducedByProcessRevision Relationship
//   role : ProducedPart
//     name : 'ProducedPart'
//     entity :
//       PdmProductStructureDefinition::PartRevision
```

```
//     cardinality : 0..unbounded
//    role : ProducedUsage
//     name : `ProducedUsage'
//     entity :  PdmProductStructure::Usage
//     cardinality : 0..unbounded
//   role : ProducingProcess
//     name : 'ProducingProcess'
//     entity : ProcessRevision
//     cardinality : 0..unbounded

interface PartProducedByProcessRevision :
          PdmFramework::PdmReferenceRelationship
{
  attribute PdmFoundation::Measurement quantity;
};

interface PartProducedByProcessRevisionFactory
{
  PartProducedByProcessRevision create_using_part_revision(
       in CosPropertyService::PropertySet property_set,
       in ProcessRevision producing_process,
       in PdmProductStructureDefinition::PartRevision
         produced_part)
    raises( RELATIONSHIP_CREATE_EXCEPTIONS );

  PartProducedByProcessRevision create_using_usage (
       in CosPropertyService::PropertySet property_set,
       in ProcessRevision producing_process,
       in PdmProductStructureDefinition::Usage produced_Usage)
    raises( RELATIONSHIP_CREATE_EXCEPTIONS );
  };

  interface ProducedPart : PdmFramework::PdmReferencesRole {};

  interface ProducedUsage : PdmFramework::PdmReferencesRole {};

  interface ProducingProcess : PdmFramework::PdmReferencedByRole {};


// ProcessMasterComposition Relationship
//   role:  ProcessMasterForRevision
//     name:  'ProcessMasterForRevision'
//     entity:  ProcessMaster
//     cardinality:  0..unbounded
//   role:  ProcessRevisionForMaster
//     name:  'ProcessRevisionForMaster'
//     entity:  ProcessRevision
//     cardinality:  1..1

interface ProcessMasterComposition :
     PdmFramework::PdmContainmentRelationship
```

```
{
};

interface ProcessMasterForRevision : PdmFramework::PdmContainsRole
{
};

interface ProcessRevisionForMaster :
PdmFramework::PdmContainedInRole
{
};

// ProcessMasterControlledByOrganization Relationship
//   role:  ProcessMasterControllingOrganization
//     name:  'ProcessMasterControllingOrganization'
//     entity:  ProcessMaster
//     cardinality:  0..1
//   role:  OrganizationControllingProcessMaster
//     name:  'OrganizationControllingProcessMaster'
//     entity:  ManufacturingOrganization
//     cardinality:  0..unbounded

interface ProcessMasterControlledByOrganization :
      PdmFramework::PdmReferenceRelationship
{
};

interface ProcessMasterControlledByOrganizationFactory
{
   ProcessMasterControlledByOrganization create(
         in CosPropertyService::PropertySet property_set,
         in ProcessMaster the_master,
         in ManufacturingOrganization the_organization)
      raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

interface ProcessMasterControllingOrganization :
      PdmFramework::PdmReferencesRole
{
};

interface OrganizationControllingProcessMaster :
      PdmFramework::PdmReferencedByRole
{
};

// ProcessMasterUsedAtOrganization Relationship
//   role:  ProcessMasterForUsingOrganization
//     name:  'ProcessMasterForUsingOrganization'
//     entity:  ProcessMaster
//     cardinality:  0..unbounded
```

```
//   role:  OrganizationUsingProcessMaster
//     name:  'OrganizationUsingProcessMaster'
//     entity:  ManufacturingOrganization
//     cardinality:  0..unbounded

interface ProcessMasterUsedAtOrganization :
     PdmFramework::PdmReferenceRelationship
{
};

interface ProcessMasterUsedAtOrganizationFactory
{
  ProcessMasterControlledByOrganization create(
        in CosPropertyService::PropertySet property_set,
        in ProcessMaster the_master,
        in ManufacturingOrganization the_organization)
     raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};


interface ProcessMasterForUsingOrganization :
     PdmFramework::PdmReferencesRole
{
};

interface OrganizationUsingProcessMaster :
     PdmFramework::PdmReferencedByRole
{
};

// ProcessOperationDocumentation Relationship
//   role:  ProcessOperationForDocument
//     name:  'ProcessOperationForDocument'
//     entity:  ProcessOperation
//     cardinality:  0..unbounded
//   role:  ProcessDocumentForOperation
//     name:  'ProcessDocumentForOperation'
//     entity:  PdmDocumentManagement::DocumentRevision
//     cardinality:  0..unbounded

interface ProcessOperationDocumentation :
     PdmFramework::PdmReferenceRelationship
{
};

interface ProcessOperationDocumentationFactory
{
  ProcessOperationDocumentation create(
        in CosPropertyService::PropertySet property_set,
        in ProcessOperation the_operation,
        in PdmDocumentManagement::DocumentRevision the_document)
```

```
        raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

interface ProcessOperationForDocument :
        PdmFramework::PdmReferencesRole
{
};

interface ProcessDocumentForOperation :
        PdmFramework::PdmReferencedByRole
{
};

// ProcessOperationSequence Relationship
//   role:  ProcessRevisionForOperation
//     name:  'ProcessRevisionForOperation'
//     entity:  ProcessRevision
//     cardinality:  0..unbounded
//   role:  ProcessOperationForRevision
//     name:  'ProcessOperationForRevision'
//     entity:  ProcessOperation
//     cardinality:  0..unbounded

interface ProcessOperationSequence :
        PdmFramework::PdmReferenceRelationship
{
  attribute string operation_sequence;
};

interface ProcessOperationSequenceFactory
{
   ProcessOperationSequence create(
          in CosPropertyService::PropertySet property_set,
          in ProcessOperation the_operation,
          in ProcessRevision the_revision)
        raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

interface ProcessRevisionForOperation :
        PdmFramework::PdmReferencesRole
{
};

interface ProcessOperationForRevision :
        PdmFramework::PdmReferencedByRole
{
};

// ProcessRevisionComposition Relationship
//   role:  ProcessRevisionForControlledProcessStep
//     name:  'ProcessRevisionForControlledProcessStep'
```

```
//     entity:  ProcessRevision
//     cardinality:  0..1
//   role:  ControlledProcessStepForRevision
//     name:  'ControlledProcessStepForRevision'
//     entity:  ControlledProcessStep
//     cardinality:  1..1

interface ProcessRevisionComposition :
     PdmFramework::PdmContainmentRelationship
{
};

interface ProcessRevisionForControlledProcessStep :
     PdmFramework::PdmContainsRole
{
};

interface ControlledProcessStepForRevision :
     PdmFramework::PdmContainedInRole
{
};

// ProcessRevisionDocumentation Relationship
//   role:  ProcessRevisionForDocument
//     name:  'ProcessRevisionForDocument'
//     entity:  ProcessRevision
//     cardinality:  0..unbounded
//   role:  ProcessDocumentForRevision
//     name:  'ProcessDocumentForRevision'
//     entity:  PdmDocumentManagement::DocumentRevision
//     cardinality:  0..unbounded

interface ProcessRevisionDocumentation :
     PdmFramework::PdmReferenceRelationship
{
};

interface ProcessRevisionDocumentationFactory
{
  ProcessRevisionDocumentation create(
        in CosPropertyService::PropertySet property_set,
        in ProcessRevision the_revision,
        in PdmDocumentManagement::DocumentRevision the_document)
     raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

interface ProcessRevisionForDocument :
     PdmFramework::PdmReferencesRole
{
};
```

```
interface ProcessDocumentForRevision :
      PdmFramework::PdmReferencedByRole
{
};

// ProcessStepConsumesPart Relationship
//   role:  ProcessStepForPart
//     name:  'ProcessStepForPart'
//     entity:  ProcessStep
//     cardinality:  0..unbounded
//   role:  PartForProcessStep
//     name:  'PartForProcessStep'
//     entity:  PdmProductStructureDefinition::PartMaster
//     cardinality:  0..unbounded

interface ProcessStepConsumesPart :
      PdmFramework::PdmReferenceRelationship
{
   attribute PdmFoundation::Measurement quantity;
};

interface ProcessStepConsumesPartFactory
{
   ProcessStepConsumesPart create(
         in CosPropertyService::PropertySet property_set,
         in ProcessStep the_step,
         in PdmProductStructureDefinition::PartMaster the_part)
      raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

interface ProcessStepForPart : PdmFramework::PdmReferencesRole
{
};

interface PartForProcessStep : PdmFramework::PdmReferencedByRole
{
};

// ProcessStepUsesTool Relationship
//   role:  ProcessStepForTool
//     name:  'ProcessStepForTool'
//     entity:  ProcessStep
//     cardinality:  0..unbounded
//   role:  ToolForProcessStep
//     name:  'ToolForProcessStep'
//     entity:  PdmConfigurationManagement::PartMaster
//     cardinality:  0..unbounded

interface ProcessStepUsesTool :
      PdmFramework::PdmReferenceRelationship
{
```

```
};

interface ProcessStepUsesToolFactory
{
  ProcessStepUsesTool create(
        in CosPropertyService::PropertySet property_set,
        in ProcessStep the_step,
        in PdmProductStructureDefinition::PartMaster the_tool)
     raises( RELATIONSHIP_CREATE_EXCEPTIONS );
};

interface ProcessStepForTool : PdmFramework::PdmReferencesRole
{
};

interface ToolForProcessStep : PdmFramework::PdmReferencedByRole
{
};

};

#endif
```

PdmConfigurationManagement Module 12

Contents

This chapter contains the following sections.

Section Title	Page
“Overview”	12-1
“PdmConfigurationManagement Model”	12-3
“PdmConfigurationManagement Description”	12-5
“PdmConfigurationManagement IDL”	12-28

12.1 Overview

The Configuration Management Enabler is an extension of the Product Structure Enabler to address the requirements of enterprises that offer many possible configurations of their products for sale. In most cases, the different configurations of a product differ from each other in only minor ways. Because of the high effort required to maintain independent product structures for each possible configuration, the desire is to have the ability to define a single, generic product structure that includes a means of specifying different components to be included in the structure based on the configuration being built. The Configuration Management Enabler provides first the ability to define the features or specifications that distinguish the different configurations and secondly, the ability to define rules to select specific components based on the feature values.

Several vendors participating in the joint submission have some form of Configuration Management control in their existing products, but it was generally agreed that all of these current implementations have their shortcomings. Among the many problems was the difficulty of mapping these implementations to new STEP standards for

Configuration Management. Therefore, we have decided to borrow most of the design for this enabler from the STEP AP214 specification currently under review. Although this AP was designed specially for the automobile industry, we believe this model is at least as universal as any other configuration management model currently available to us.

Below are a few paragraphs to give a high level overview of the model before getting to the UML itself. In most instances we have used object names that are very similar to the STEP entities. The model is divided into two sections. The first section defines the classes of products being produced and the available specification categories and specification options available for each product. (A specification category could be Brake-Type with its specification options being Disk Brakes, Drum Brakes, or ABS.)

To assist in standardizing specifications across various products, this section of the model includes the **ProductClass** (for example, Car) object and **ProductClass** hierarchy. The leaves of the **ProductClass** hierarchy represent actual products offered for sale (for example, 4 Door Luxury Sedan). Characteristics shared by several **ProductClass**es can be abstracted up to parent ProductClass, creating a product hierarchy allowing specifications to be inherited from parent classes. Thus the possible Specification Categories can be inherited from parent **ProductClass**es, eliminating repetitive data. The result is a type of feature classification hierarchy.

This section also includes the ability to define simple rule expressions that allow the definition of specification packages and/or the dependencies between specifications that are required or prohibited. For example, the combination of the A/C option and the automatic transmission requires that the 6 cylinder engine option be included.

The second section of the model allows the mapping of specification conditions to parts in the product structure. This is done via **ProductComponent**. A **ProductComponent** represents a generic node in the product structure where there are several choices of parts that may be used. The association between a **ProductComponent** and each of the possible part solutions can be qualified by specifying the specification condition that requires the use of that specific part.

ProductComponents can be related together in the same way that parts are related to form a BOM. This **ProductComponent** hierarchy could represent a functional decomposition of the product and serves as a template to which all variants of the Product must conform. This not only keeps consistency within the product family, but also allows new configurations to be defined quickly.

Because this model is based on the configuration management model defined in STEP AP214, additional information and description of how to use the model can be found in the ARM model and other documentation on AP214.

12.2 PdmConfigurationManagement Model

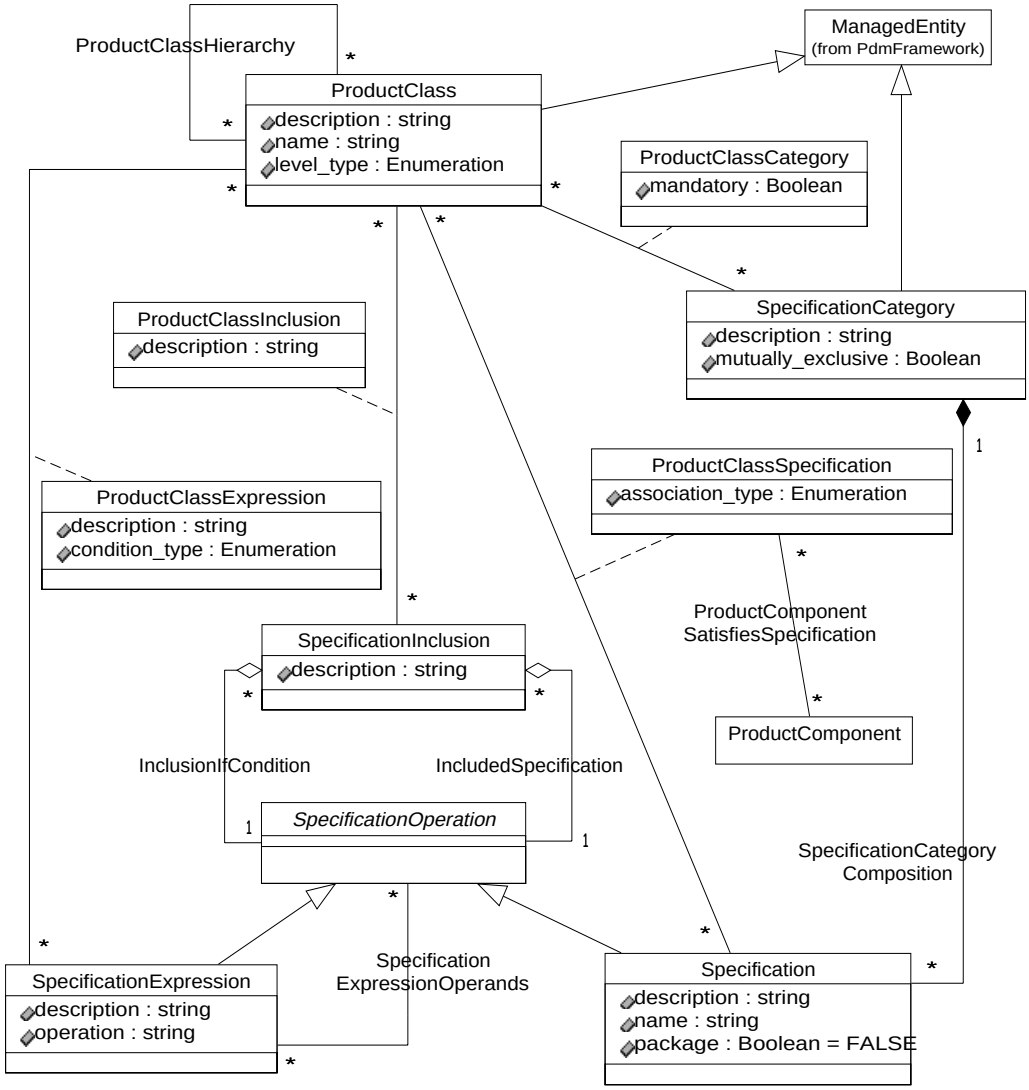

Figure 12-1 PdmConfigurationManagement Model Diagram (1)

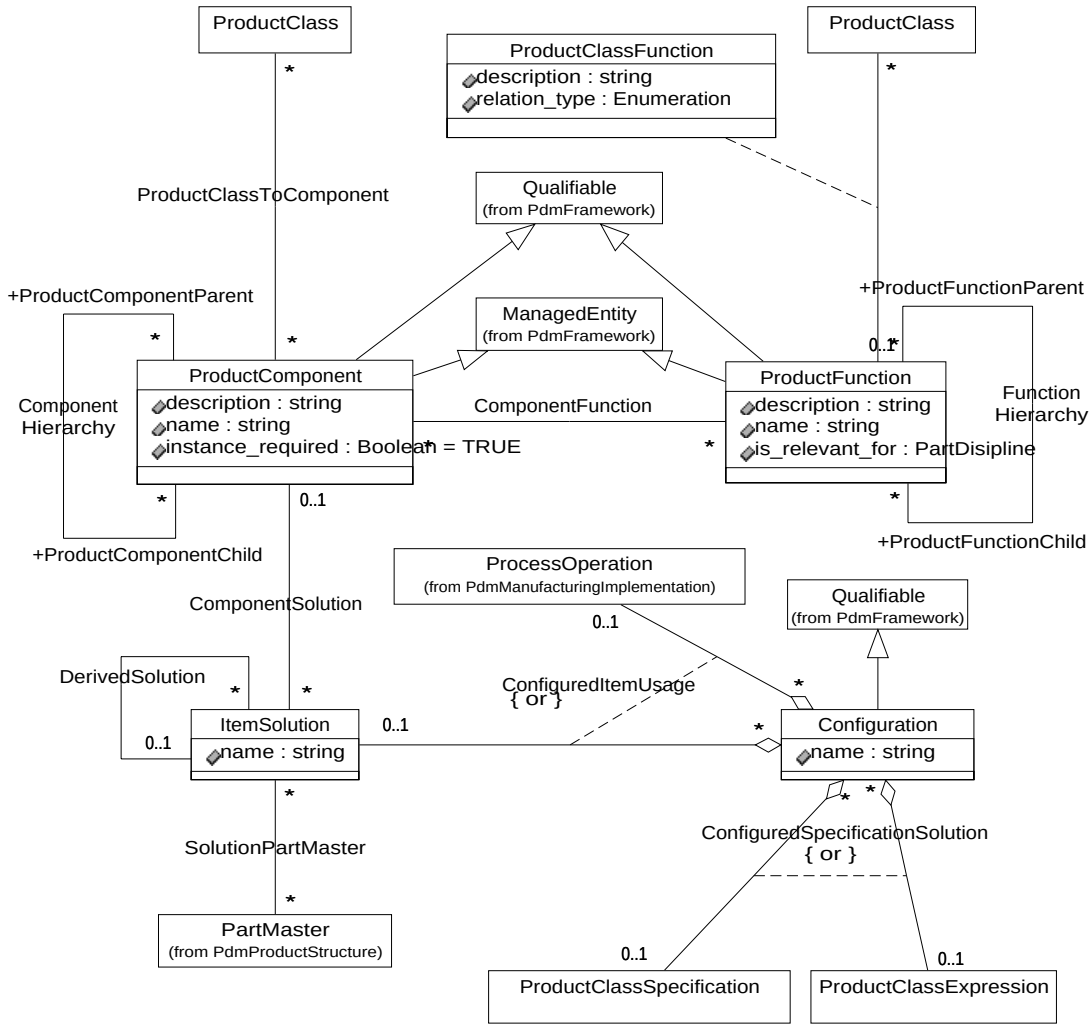

Figure 12-2 PdmConfigurationManagement Model Diagram (2)

Note – In the above model the interfaces - Configuration, ItemSolution - inherit from **CosCompoundLifeCycle::Node**, **CosLifeCycle::LifeCycleObject**, **CosTransactions::TransactionalObject**, and **PdmFramework::Attributable**. However these inheritances have not been shown on the diagram.

12.3 PdmConfigurationManagement Description

12.3.1 PartDiscipline

```
typedef sequence<string> PartDiscipline;
```

12.3.2 ProductClass

ProductClass objects are used to group and relate similar sets of products produced by the enterprise. Each **ProductClass** is related to the features (specifications) that define the configurable characteristics of the products of this object. A hierarchy of **ProductClass**es can be defined to allow shared features to be pushed up and defined on a common parent **ProductClass**, forming a type of Product feature classification hierarchy.

```
interface ProductClass : PdmFramework::ManagedEntity
{
  attribute string description;
  attribute string name;
  attribute string level_type;
};

interface ProductClassFactory
{
  ProductClass create(in CosPropertyService::PropertySet property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};
```

Example for an object of the type **ProductClass** may be:

- Trucks
- 2-door cars
- Japan Laptops

description

Used to provide a textual description of the family of products this **ProductClass** represents.

name

Used to specify a short name for the **ProductClass**.

level_type

Specifies the level or category of the **ProductClass** within the hierarchy. The suggested levels are:

- Enterprise - used to define Specification and categories for all **ProductClass** objects in an enterprise with several brands or companies.
- Platform - used to group all product based on the same technical concept.

- Family - used to group all products that have a common base and fixed set of characteristics [SpecificationCategory].
- Type - may be used to group products by marketing range. It is usually at this level that a distinction is made between standard characteristics. For example, number of doors and options that have to be chosen (e.g., color).

12.3.3 ProductClassHierarchy

The **ProductClassHierarchy** object is used to relate **ProductClass** instances into a tree structure in order to allow common specification information to be associated to a **ProductClassSuperClass ProductClass** and inherited by all the descendent **ProductClass** instances.

```
// ProductClassHierarchy Relationship
//   role : ProductClassSuperClass
//     name: 'ProductClassSuperClass'
//     entity : ProductClass
//     cardinality : 0..unbounded
//   role : ProductClassSubClass
//     name : 'ProductClassSubClass'
//     entity : ProductClass
//     cardinality : 0..1

interface ProductClassHierarchy :
  PdmFramework::PdmReferenceRelationship { };

interface ProductClassHierarchyFactory
{
  ProductClassHierarchy create(
        in CosPropertyService::PropertySet property_set,
        in ProductClass product_class_superclass,
        in ProductClass product_class_subclass)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductClassSuperClass :
  PdmFramework::PdmReferencesRole { };

interface ProductClassSubClass :
  PdmFramework::PdmReferencedByRole { };
```

12.3.4 ProductComponent

A **ProductComponent** is a generic representation of a functional component in a product structure. All the physical parts that provide the generic function are associated to the **ProductComponent**. The association of **ProductComponent** to Part can be qualified to indicate the specification condition when the physical part is to be used.

ProductComponents can also be associated together, forming functional decomposition of the Product Structure and a template, which all variations of the product structure most conform to. The top level **ProductComponent** of this structure is associated to a **ProductRootClass**.

Each **ProductComponent** object is associated to a set of **ItemSolution** objects, which identify parts that meet the **ProductComponent**'s functional requirements.

The design of a new product that belongs to an existing **ProductClass** begins by reviewing all the possible **ItemSolution**s for each **ProductComponent** in the decomposition structure and identify the ones that can be used in the new product by adding or changing the various **ProductClassSpecification** relationships.

```
interface ProductComponent : PdmFramework::ManagedEntity,
  PdmFramework::Qualifiable
{
  attribute string name;
  attribute string description;
  attribute boolean instance_required;
};

interface ProductComponentFactory
{
  ProductComponent create(
       in CosPropertyService::PropertySet property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};
```

name

Provides a name for the **ProductComponent** object.

description

Used to provide a textual description for the **ProductComponent** object.

instance_required

Indicates if all instances of the products are required to use at least one of the associated item solutions and related parts.

12.3.5 *ProductClassToComponent*

ProductClassToComponent is a relationship between a **ProductClass** and a **ProductComponent**. It is used to relate product class to its components.

```
// ProductClassToComponent Relationship
//   role : RootProduct
//     name : 'RootProduct'
//     entity : ProductClass
//     cardinality : 0..unbounded
```

```
//   role : ValidProductComponent
//     name : 'ValidProductComponent'
//     entity : ProductComponent
//     cardinality : 0..unbounded

interface ProductClassToComponent :
  PdmFramework::PdmReferenceRelationship { };

interface ProductClassToComponentFactory
{
  ProductClassToComponent create(
        in CosPropertyService::PropertySet property_set,
        in ProductClass the_root_product,
        in ProductComponent the_valid_product_component)
    raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface RootProduct : PdmFramework::PdmReferencesRole {};

interface ValidProductComponent :
  PdmFramework::PdmReferencedByRole{};
```

12.3.6 ComponentHierarchy

ComponentHierarchy is a relationship between two **ProductComponent**s. A **ComponentHierarchy** relationship can be used to create tree like hierarchy of **ProductComponent**s.

```
// ComponentHierarchy Relationship
//   role : ProductComponentParent
//     name: 'ProductComponentParent'
//     entity : ProductComponent
//     cardinality : 0..unbounded
//   role : ProductComponentChild
//     name : 'ProductComponentChild'
//     entity : ProductComponent
//     cardinality : 0..unbounded

interface ComponentHierarchy :
  PdmFramework::PdmReferenceRelationship { };

interface ComponentHierarchyFactory
{
  ComponentHierarchy create(
        in CosPropertyService::PropertySet property_set,
        in ProductComponent product_component_parent,
        in ProductComponent product_component_child )
    raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};
```

```
interface ProductComponentParent :
  PdmFramework::PdmReferencesRole { };

interface ProductComponentChild :
  PdmFramework::PdmReferencedByRole { };
```

12.3.7 ProductFunction

A **ProductFunction** object represents the purpose or function that **ProductComponent**s objects fulfills for the product that contains it. Using the **ProductFuctionHierarchy**, a functional decomposition and classification hierarchy can be built.

The **ProductFunction** and the **ProductFunction** hierarchy allows **ProductComponent**s from different **ProductClass**es to be associated, allowing the user to view parts serving similar purpose not only within a **ProductComponent** but also across **ProductComponent**s and **ProductClass**es.

```
interface ProductFunction : PdmFramework::ManagedEntity,
  PdmFramework::Qualifiable
{
  attribute string name;
  attribute string description;
  attribute PartDiscipline is_relevant_for;
};

interface ProductFunctionFactory
{
  ProductFunction create(
       in CosPropertyService::PropertySet property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};
```

For example, ‘interior illumination’ and ‘exterior illumination’ functions may be related to ‘illumination’ function thus grouping together all the **ProductComponent**s and their related parts that provide some type of illumination.

description

Used to provide a textual description for the function of the **ProductFunction** object.

name

Provides a short name for the **ProductFunction** object.

is_relevant_for

Identifies the context or set of contexts for the **ProductFunction** object.

12.3.8 FunctionHierarchy

FunctionHierarchy is a relationship between two **ProductFunction**s. It is used to establish a hierarchical relationship between **ProductFunction**s.

```
// FunctionHierarchy Relationship
//   role : ProductFunctionParent
//     name: 'ProductFunctionParent'
//     entity : ProductFunction
//     cardinality : 0..unbounded
//   role : ProductFunctionChild
//     name : 'ProductFunctionChild'
//     entity : ProductFunction
//     cardinality : 0..1

interface FunctionHierarchy :
  PdmFramework::PdmReferenceRelationship { };

interface FunctionHierarchyFactory
{
  ProductFunction create(
        in CosPropertyService::PropertySet property_set,
        in ProductFunction product_function_parent,
        in ProductFunction product_function_child )
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductFunctionParent :
  PdmFramework::PdmReferencesRole { };

interface ProductFunctionChild :
  PdmFramework::PdmReferencedByRole { };
```

12.3.9 ComponentFunction

ComponentFunction Relationship is a relationship between a **ProductComponent** and a **ProductFunction**. It indicates a function that a **ProductComponent** performs.

```
// ComponentFunction Relationship
//   role : ProductComponentForFunction
//     name: 'ProductComponentForFunction'
//     entity : ProductComponent
//     cardinality : 0..unbounded
//   role : ProductFunctionForComponent
//     name : 'ProductFunctionForComponent'
//     entity : ProductFunction
//     cardinality : 0..unbounded

interface ComponentFunction :
```

```
  PdmFramework::PdmReferenceRelationship { };

interface ComponentFunctionFactory
{
  ComponentFunction create(
       in CosPropertyService::PropertySet property_set,
       in ProductComponent product_component_for_function,
       in ProductFunction product_function_for_component )
    raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductComponentForFunction :
  PdmFramework::PdmReferencesRole { };

interface ProductFunctionForComponent :
  PdmFramework::PdmReferencedByRole { };
```

12.3.10 ProductClassFunction

A **ProductClassFunction** object is used to associate a **ProductFunction** to a **ProductClass**.

```
// ProductClassFunction Relationship
//  role : ProductUsingFunction
//    name : 'ProductUsingFunction'
//    entity : ProductClass
//    cardinality : 0..unbounded
//  role : ValidProductFunction
//    name : 'ValidProductFunction'
//    entity : ProductFunction
//    cardinality : 0..unbounded

interface ProductClassFunction :
  PdmFramework::PdmReferenceRelationship
{
  attribute string description;
  attribute string relation_type;
};

interface ProductClassFunctionFactory
{
  ProductClassFunction create(
       in CosPropertyService::PropertySet property_set,
       in ProductClass the_product_using_function,
       in ProductFunction the_valid_product_function)
    raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductUsingFunction :
  PdmFramework::PdmReferencesRole {};
```

```
interface ValidProductFunction :
  PdmFramework::PdmReferencedByRole{};
```

description

Used to provide textual description for the specific requirements or reason that the **ProductClass** contains the **ProductFunction** instance.

relation_type

Specifies the meaning of the relationship. Possible values are:

- possibility (the **ProductFunction** has to be considered for at least some products of the **ProductClass**).
- main function (this is one of the primary functions performed by each product of the **ProductClass**).

12.3.11 ItemSolution

An **ItemSolution** defines a solution to the function requirement defined by a **ProductComponent**. An **ItemSolution** may represent a technical solution, such as 'Disc brake' or 'ABS brake' for **ProductComponent** 'front left brake,' or a part solution, which identifies a specific set of **PartMasters** that jointly fulfill the functional requirement (that is, the part number of a tire that may used on a car). Multiple **PartMasters** associated to a single **ItemSolution** represents a kit of parts that jointly fulfill the **ItemSolution**. The **ItemSolution**s for the same **ProductComponent** represent mutually exclusive alternatives.

The association with the **Configuration** object identifies the condition (Specifications) and the effectivity under which the **ItemSolution** is considered valid.

```
interface ItemSolution :  CosLifeCycle::LifeCycleObject,
  CosTransactions::TransactionalObject,
  CosCompoundLifeCycle::Node, PdmFramework::Attributable
{
  attribute string name;
};

  interface ItemSolutionFactory
{
  ItemSolution create(
       in CosPropertyService::PropertySet property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};
```

name

The name attribute for the **ItemSolution** object is used to specify a unique identifier for an instance.

12.3.12 ComponentSolution

The **ComponentSolution** is a relationship between an instance of a **ProductComponent** and an instance of an **ItemSolution** that fulfills its functional requirements. This relationship defines a set of mutually exclusive solutions that describe how a functional requirement of a product may be satisfied or fulfilled.

```
// ComponentSolution Relationship
//   role : ProductComponentForSolution
//     name: 'ProductComponentForSolution'
//     entity : ProductComponent
//     cardinality : 0..unbounded
//   role : ItemSolutionForComponent
//     name : 'ItemSolutionForComponent'
//     entity : ItemSolution
//     cardinality : 0..1

interface ComponentSolution :
  PdmFramework::PdmReferenceRelationship { };

interface ComponentSolutionFactory
{
  ComponentSolution create(
        in CosPropertyService::PropertySet property_set,
        in ProductComponent product_component_for_solution,
        in ItemSolution item_solution_for_component)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductComponentForSolution :
  PdmFramework::PdmReferencesRole { };

interface ItemSolutionForComponent :
  PdmFramework::PdmReferencedByRole { };
```

12.3.12.1 DerivedSolution

DerivedSolution is a relationship between two **ItemSolution**s when one **ItemSolution** is derived from the other.

```
// DerivedSolution Relationship
//   role : OriginalItemSolution
//     name: 'OriginalItemSolution'
//     entity : ItemSolution
//     cardinality : 0..unbounded
//   role : DerivedItemSolution
//     name : 'DerivedItemSolution'
//     entity : ItemSolution
//     cardinality : 0..1
```

```
interface DerivedSolution :
  PdmFramework::PdmReferenceRelationship { };

interface DerivedSolutionFactory
{
  DerivedSolution create(
        in CosPropertyService::PropertySet property_set,
        in ItemSolution original_item_solution,
        in ItemSolution derived_item_solution)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface OriginalItemSolution :
  PdmFramework::PdmReferencesRole { };

interface DerivedItemSolution :
  PdmFramework::PdmReferencedByRole { };
```

12.3.13 SolutionPartMaster

SolutionPartMaster is a relationship between an **ItemSolution** and a **PartMaster**. It is used to indicate the part for which the **ItemSolution** provides a solution.

```
// SolutionPartMaster Relationship
//   role : ItemSolutionForMaster
//     name: 'ItemSolutionForMaster'
//     entity : ItemSolution
//     cardinality : 0..unbounded
//   role : PartMasterForSolution
//     name : 'PartMasterForSolution'
//     entity : PartMaster
//     cardinality : 0..unbounded

interface SolutionPartMaster :
  PdmFramework::PdmReferenceRelationship { };

interface SolutionPartMasterFactory
{
  SolutionPartMaster create(
        in CosPropertyService::PropertySet property_set,
        in ItemSolution item_solution_for_master,
        in PdmProductStructureDefinition::PartMaster
           part_master_for_solution )
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ItemSolutionForMaster :
  PdmFramework::PdmReferencesRole { };

interface PartMasterForSolution :
```

```
PdmFramework::PdmReferencedByRole { };
```

12.3.14 SpecificationCategory

The **SpecificationCategory** defines a set of features that serve the same purpose or function.

```
interface SpecificationCategory : PdmFramework::ManagedEntity
{
  attribute string description;
  attribute boolean mutually_exclusive;
};

interface SpecificationCategoryFactory
{
  SpecificationCategory create(
       in CosPropertyService::PropertySet property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};
```

description

Used to provide a textual description for the **SpecificationCategory** object. Example:

Hard Disk Size (related specifications = 850M, 1.2G, 2G)

mutually_exclusive

Specifies whether the related **Specification** instances are mutually exclusive. For example, an Engine (SpecificationCategory) could be one of two types; say a 4 cylinder engine (Specification) and a 6 cylinder engine (Specification). Both the Specification objects in the Engine category are mutually exclusive as a car can have only one type of engine.

12.3.14.1 SpecificationOperation

The **SpecificationOperation** is an abstract class. The **Specification** class and the **SpecificationExpression** are its sub-classes. Sub-classes of the **SpecificationOperation** may be used for both the **if_condition** and the **included_specification** of the **SpecificationInclusion** object.

```
interface SpecificationOperation :
  CosTransactions::TransactionalObject,
  CosLifeCycle::LifeCycleObject, CosCompoundLifeCycle::Node,
  PdmFramework::Attributable { };
```

12.3.14.2 Specification

A **Specification** is a characteristic, feature, or option of a **ProductClass** that may be used within a product's structure to select among a set of possible Part solutions. A **Specification** may also represent a feature package (See package attribute).

```
interface Specification : SpecificationOperation,
  PdmFoundation::Manageable
{
  attribute string description;
  attribute string name;
  attribute boolean package;
};

interface SpecificationFactory
{
  Specification create(
       in CosPropertyService::PropertySet property_set,
       in SpecificationCategory category_for_specification)
    raises(ITEM_CREATE_EXCEPTIONS);
};
```

Examples:

- 110 volts, 220 volts
- sunroof, t-top, convertible

description

Used to specify a textual description for the **Specification** object.

name

Provides a short name for the **Specification** choice. The name must be unique within a particular **SpecificationCategory**.

package

Specifies whether this Specification represents a package of Specification objects. A package may be defined by the marketing department and combines a set of **Specification** objects that shall be offered to the market as a set.

When set to TRUE, exactly one **SpecificationInclusion** instance shall reference this Specification as it's **if_condition**. The inclusion **SpecificationExpression** shall list all the included **Specifications** for this package.

12.3.15 SpecificationExpression

A **SpecificationExpression** is a combination of specification instances combined by simple Boolean operations.

```
interface SpecificationExpression : SpecificationOperation
{
  attribute string description;
  attribute string operation;
};

interface SpecificationExpressionFactory
{
  SpecificationExpression create(
       in CosPropertyService::PropertySet property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};
```

For example:

if (operation = OR, operands = AC, towing package, 4-wheel drive)
included_specification (operation = AND, operands = 350 HP engine, large battery)

would mean if the AC or towing package or 4-wheel drive feature was selected, then also include the 350HP engine and the large battery feature.

description

Used to provide a textual description of the **SpecificationExpression** object.

operation

Specifies the Boolean relationship between the operands. There are four kinds of operations permitted:

1. AND (all the identified specifications apply)
2. OR (any subset or all the identified specifications apply)
3. ONEOF (exactly one of the identified specifications apply)
4. NOT (the identified specification must not apply)

12.3.16 SpecificationInclusion

A **SpecificationInclusion** is the representation of the statement that the application of a **Specification** or **SpecificationExpression** implies or requires the inclusion of an additional **Specification** or **SpecificationExpression**.

```
interface SpecificationInclusion : CosTransactions::TransactionalObject,
  CosLifeCycle::LifeCycleObject, CosCompoundLifeCycle::Node,
  PdmFramework::Attributable
{
  attribute string description;
};

interface SpecificationInclusionFactory
{
```

```
  SpecificationInclusion create(
        in CosPropertyService::PropertySet property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};
```

description

Used to provide a textual description for the **SpecificationInclusion**.

12.3.17 ProductClassCategory

The **ProductClassCategory** is used to identify the **SpecificationCategories** that apply to a **ProductClass**.

```
// ProductClassCategory Relationship
//   role : ProductUsingCategory
//     name: 'ProductUsingCategory'
//     entity : ProductClass
//     cardinality : 0..unbounded
//   role : ValidCategories
//     name : 'ValidCategories'
//     entity : SpecificationCategory
//     cardinality : 0..unbounded

interface ProductClassCategory :
  PdmFramework::PdmReferenceRelationship
{
  attribute boolean mandatory;
};

interface ProductClassCategoryFactory
{
  ProductClassCategory create(
        in CosPropertyService::PropertySet property_set,
        in ProductClass product_using_category,
        in SpecificationCategory valid_categories)
    raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductUsingCategory :
  PdmFramework::PdmReferencesRole { };

interface ValidCategories :
  PdmFramework::PdmReferencedByRole { };
```

mandatory

Indicates if one or more of the **Specification** objects within the referred category must be used within the products within this **ProductClass**.

For example, the radio within a car is usually optional equipment, but cars are not sold without seats.

12.3.18 ProductClassExpression

The **ProductClassExpression** object associates a **SpecificationExpression** to a **ProductClass** for which it is used. This relationship indicates that the **SpecificationExpression** is valid for all products of the referenced **ProductClass**.

```
// ProductClassExpression Relationship
//   role : ProductUsingExpression
//     name: 'ProductUsingExpression'
//     entity : ProductClass
//     cardinality : 0..unbounded
//   role : ValidProductExpression
//     name : 'ValidProductExpression'
//     entity : SpecificationExpression
//     cardinality : 0..unbounded

interface ProductClassExpression :
  CosCompoundLifeCycle::Node,
PdmFramework::PdmReferenceRelationship
{
  attribute string description;
  attribute string condition_type;
};

interface ProductClassExpressionFactory
{
  ProductClassExpression create(
        in CosPropertyService::PropertySet property_set,
        in ProductClass product_using_expression,
        in SpecificationExpression valid_product_expression)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductUsingExpression :
  PdmFramework::PdmReferencesRole { };

interface ValidProductExpression :
  PdmFramework::PdmReferencedByRole { };
```

description

Used to provide textual description of why this **SpecificationExpression** applies to the specified **ProductClass**.

condition_type

Specifies the meaning of the relationship.

The value of **condition_type** shall be one of the following:

- part usage - specifies a condition for the usage of the components of an **ItemSolution** in the **ProductClass**.
- identification - specifies a condition that enables the associated **ProductClass** to be distinguished from other **ProductClass** objects.
- validity - specifies a condition that is used to verify that a given set of Specification objects for a **ProductClass** are valid.

12.3.19 ProductClassInclusion

A **ProductClassInclusion** is the assignment of a **SpecificationInclusion** instance to a **ProductClass** and applies to all products of that **ProductClass**.

```
// ProductClassInclusion Relationship
//   role : ProductUsingInclusion
//     name: 'ProductUsingInclusion'
//     entity : ProductClass
//     cardinality : 0..unbounded
//   role : 'ValidProductInclusion'
//     name : 'ValidProductInclusion'
//     entity : SpecificationInclusion
//     cardinality : 0..unbounded

interface ProductClassInclusion :
  PdmFramework::PdmReferenceRelationship
{
  attribute string description;
};

interface ProductClassInclusionFactory
{
  ProductClassInclusion create(
        in CosPropertyService::PropertySet property_set,
        in ProductClass product_using_inclusion,
        in SpecificationInclusion valid_product_inclusion)
    raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductUsingInclusion :
  PdmFramework::PdmReferencesRole { };

interface ValidProductInclusion :
  PdmFramework::PdmReferencedByRole { };
```

description

Used to provide textual description of why the **SpecificationInclusion** rule applies to the particular **ProductClass**.

12.3.20 ProductClassSpecification

This object relates a **Specification** object to an instance of a **ProductClass** object. A **SpecificationCategory** may include **Specification**s that do not apply to all instances of a particular **ProductClass**. This relation indicates the specific **Specification** objects that apply to each **ProductClass** and how the specification is used for that **ProductClass**.

```
// ProductClassSpecification Relationship
//   role : ProductUsingSpecification
//     name: 'ProductUsingSpecification'
//     entity : ProductClass
//     cardinality : 0..unbounded
//   role : ValidProductSpecification
//     name : 'ValidProductSpecification'
//     entity : Specification
//     cardinality : 0..unbounded

interface ProductClassSpecification :
  CosCompoundLifeCycle::Node,
  PdmFramework::PdmReferenceRelationship
{
  attribute string association_type;
};

interface ProductClassSpecificationFactory
{
  ProductClassSpecification create(
        in CosPropertyService::PropertySet property_set,
        in ProductClass product_using_specification,
        in Specification valid_product_specification)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductUsingSpecification :
  PdmFramework::PdmReferencesRole { };

interface ValidProductSpecification :
  PdmFramework::PdmReferencedByRole { };
```

association_type

Specifies the availability of a particular specification in a **ProductClass**. The value of **association_type** may be one of the following:

- replaceable standard - the Specification is a default characteristic for the products belonging to the **ProductClass**, as long as no other specification of the same **SpecificationCategory** has not been chosen.
- non-replaceable standard - the Specification is an unconditional characteristic of any product in the **ProductClass**.

- availability - the Specification is a potential characteristic of the **ProductClass**. It is not specified if this is a option or a standard.
- identification - the Specification is a characteristic that allows the associated **ProductClass** to be distinguished from other **ProductClass** objects. This is similar to ‘non replaceable standard.’
- option - the Specification is a characteristic of a product only if explicitly chosen. This Specification can replace a ‘replaceable standard’ Specification of the same category.

12.3.21 InclusionIfCondition

An **InclusionIfCondition** is a relationship between a **SpecificationInclusion** and a **SpecificationOperation**. An **InclusionIfCondition** instance applies when the related **SpecificationOperation** is selected.

```
// InclusionIfCondition Relationship
//   role : InclusionIfOperation
//     name: 'InclusionIfOperation'
//     entity : SpecificationInclusion
//     cardinality : 1..1
//   role : IfOperationForInclusion
//     name : 'IfOperationForInclusion'
//     entity : SpecificationOperation
//     cardinality : 0..unbounded

interface InclusionIfCondition :
  PdmFramework::PdmReferenceRelationship { };

interface InclusionIfConditionFactory
{
  InclusionIfCondition create(
        in CosPropertyService::PropertySet property_set,
        in SpecificationInclusion inclusion_if_operation,
        in SpecificationOperation if_operation_for_inclusion)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface InclusionIfOperation :
  PdmFramework::PdmReferencesRole { };

interface IfOperationForInclusion :
  PdmFramework::PdmReferencedByRole { };
```

12.3.22 IncludedSpecification

An **IncludedSpecification** is a relationship between a **SpecificationInclusion** and a **SpecificationOperation**. It indicates a related **SpecificationOperation** that should be selected when the **InclusionIfCondition** evaluates to true.

```
// IncludedSpecification Relationship
//   role : SpecificationInclusionForOperation
//     name: 'InclusionForOperation'
//     entity : SpecificationInclusion
//     cardinality : 1..1
//   role : SpecificationOperationForInclusion
//     name : 'SpecificationOperationForInclusion'
//     entity : SpecificationOperation
//     cardinality : 0..unbounded

interface IncludedSpecification :
  PdmFramework::PdmReferenceRelationship { };

interface IncludedSpecificationFactory
{
  IncludedSpecification create(
        in CosPropertyService::PropertySet property_set,
        in SpecificationInclusion inclusion_for_operation,
        in SpecificationOperation operation_for_inclusion)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface SpecificationInclusionForOperation :
  PdmFramework::PdmReferencesRole { };

interface SpecificationOperationForInclusion :
  PdmFramework::PdmReferencedByRole {};
```

12.3.23 SpecificationCategoryComposition

A **SpecificationCategory** is the definition of a set of **Specification** objects that serve the same purpose. A **SpecificationCategoryComposition** is a relation between **SpecificationCategory** object and all the Specification objects that fall under that category.

```
// SpecificationCategoryComposition Relationship
//   role : SpecificationCategoryForSpecifications
//     name: 'SpecificationCategoryForSpecifications'
//     entity : SpecificationCategory
//     cardinality : 0..unbounded
//   role : SpecificationForSpecificationCategory
//     name : 'SpecificationForSpecificationCategory'
//     entity : Specification
//     cardinality : 1..1

interface SpecificationCategoryComposition :
  PdmFramework::PdmContainmentRelationship { };

interface SpecificationCategoryForSpecifications :
  PdmFramework::PdmContainsRole { };
```

```
interface SpecificationForSpecificationCategory :
  PdmFramework::PdmContainedInRole { };
```

12.3.24 SpecificationExpressionOperands

A **SpecificationExpression** object is a combination of **SpecificationOperation** objects formed by Boolean operations. A **SpecificationExpressionOperands** object is a relation between a **SpecificationExpression** and **SpecificationOperation** object/objects that are combined using the Boolean operation.

- When the Boolean operation is a NOT operation only one **SpecificationOperation** object acts as the operand.
- When the Boolean operations are of the type OR or AND there are exactly two **SpecificationOperation** objects as operands.
- When the Boolean operation is of the type ONEOF there can be two or more **SpecificationOperation** operands.

```
// SpecificationExpressionOperands Relationship
//   role : ExpressionForOperands
//     name: 'ExpressionForOperands'
//     entity : SpecificationExpression
//     cardinality : 0..unbounded
//   role : OperandForExpression
//     name : 'OperandForExpression'
//     entity : SpecificationOperand
//     cardinality : 0..unbounded

interface SpecificationExpressionOperands :
  PdmFramework::PdmReferenceRelationship { };

interface SpecificationExpressionOperandsFactory
{
  SpecificationExpressionOperands create(
        in CosPropertyService::PropertySet property_set,
        in SpecificationExpression expression_for_operands,
        in SpecificationOperation operand_for_expression)
    raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ExpressionForOperands :
  PdmFramework::PdmReferencesRole { };

interface OperandForExpression :
  PdmFramework::PdmReferencedByRole { };
```

12.3.25 Configuration

A **Configuration** is the association of a **Specification** or **SpecificationExpression** (which identifies an option or set of options selected), an **Effectivity** (valid time range for which the association applies), and a **ProcessOperation** or **ItemSolution**.

Since the **Configuration** can alter the content of the products BOM, it may also be necessary to alter the process operation steps that describe how to assemble the optional part. Therefore, the configuration not only has the ability to qualify the usage of a particular part, but also qualify the use of a particular process operation.

Each **Configuration** may control its usage by time, serial numbers, or lot numbers via effectivity.

```
interface Configuration :  CosLifeCycle::LifeCycleObject,
  CosTransactions::TransactionalObject,
  CosCompoundLifeCycle::Node, PdmFramework::Attributable,
  PdmFramework::Qualifiable
{
  attribute string name;
};

interface ConfigurationFactory
{
  Configuration create(
       in CosPropertyService::PropertySet property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};
```

name

Used to specify a short name for the **Configuration** object.

12.3.26 ConfiguredItemUsage

ConfiguredItemUsage is a relationship between a **Configuration** and an **ItemSolution** or a **ProcessOperation**. It specifies the characteristics (that is, the **ItemSolution** or **ProcessOperation** that is control by the **Configuration**).

Since an instance of an **ItemSolution** or a **ProcessOperation** can take part in the relationship the factory for this relationship has two separate create functions, one for each type of object.

```
// ConfiguredItemUsage Relationship
//   role : ConfigurationForUsage
//     name: 'ConfigurationForUsage'
//     entity : Configuration
//     cardinality : 0..1
//   role : SolutionConfiguration
//     name : 'SolutionConfiguration'
```

```
//    entity : ItemSolution
//    cardinality : 0..unbounded
//  role : OperationConfiguration
//    name : 'OperationConfiguration'
//    entity : ProcessOperation
//    cardinality : 0..unbounded

interface ConfiguredItemUsage :
  PdmFramework::PdmReferenceRelationship { };

interface ConfiguredItemUsageFactory
{
  ConfiguredItemUsage create_using_solution(
       in CosPropertyService::PropertySet property_set,
       in Configuration configuration_for_usage,
       in ItemSolution solution_configuration)
    raises(RELATIONSHIP_CREATE_EXCEPTIONS);
  ConfiguredItemUsage create_using_operation(
       in CosPropertyService::PropertySet property_set,
       in Configuration configuration_for_usage,
       in PdmManufacturingImplementation::ProcessOperation
          operation_configuration)
    raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ConfigurationForUsage :
  PdmFramework::PdmReferencesRole { };

interface SolutionConfiguration :
  PdmFramework::PdmReferencedByRole { };

interface OperationConfiguration :
  PdmFramework::PdmReferencedByRole { };
```

12.3.27 ConfiguredSpecificationSolution

ConfiguredSpecificationSolution is a relationship between a **Configuration** and a **ProductClassExpression** or a **ProductClassSpecification**. This relationship is used to indicate the **Specification** or a valid combination of **Specification**s in the form of a **SpecificationExpression** for which the **ItemSolution** provides a solution or the **ProcessOperation** that is controlled.

Since an instance of a **ProductClassSpecification** or a **ProductClassExpression** can take part in the relationship the factory for this relationship has two separate create functions, one for each type of object.

```
// ConfiguredSpecificationSolution Relationship
//  role : ConfigurationForSolution
//    name: 'ConfigurationForSolution'
//    entity : Configuration
```

```
//     cardinality : 0..1
//   role : ConfigurationExpression
//     name : 'ConfigurationExpression'
//     entity : ProductClassExpression
//     cardinality : 0..unbounded
//   role : ConfigurationSpecification
//     name : 'ConfigurationSpecification'
//     entity : ProductClassSpecification
//     cardinality : 0..unbounded

interface ConfiguredSpecificationSolution :
  PdmFramework::PdmReferenceRelationship { };

interface ConfiguredSpecificationSolutionFactory
{
  ConfiguredSpecificationSolution create_using_specification(
        in CosPropertyService::PropertySet property_set,
        in Configuration configuration_for_solution,
        in ProductClassSpecification configuration_specification)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
  ConfiguredSpecificationSolution create_using_expression(
        in CosPropertyService::PropertySet property_set,
        in Configuration configuration_for_solution,
        in ProductClassExpression configuration_expression)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ConfigurationForSolution :
  PdmFramework::PdmReferencesRole { };

interface ConfigurationExpression :
  PdmFramework::PdmReferencedByRole { };

interface ConfigurationSpecification :
  PdmFramework::PdmReferencedByRole { };
```

12.3.28 ProductComponentSatisfiesSpecification

The **ProductComponentSatisfiesSpecification** is a relationship between a **ProductComponent** and a **ProductClassSpecification**. This relationship is used to indicate the **Specification** within a particular **ProductClass** that is satisfied by the **ProductComponent**.

```
// ProductComponentSatisfiesSpecification Relationship
//   role: ProductComponentForSpecification
//     name: 'ProductComponentForSpecification'
//     entity: ProductComponent
//     cardinality: 0..unbounded
//   role: ProductClassSpecificationForSpecification
//     name: 'ProductClassSpecificationForSpecification'
```

```
//     entity: ProductClassSpecification
//     cardinality: 0..unbounded

interface ProductComponentSatisfiesSpecification :
  PdmFramework::PdmReferenceRelationship { };

interface ProductComponentSatisfiesSpecificationFactory
{
  ProductComponentSatisfiesSpecification create (
        in CosPropertyService::PropertySet property_set,
        in ProductComponent product_component,
        in ProductClassSpecification product_class_specification)
      raises (RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductComponentForSpecification :
  PdmFramework::PdmReferencesRole { };

interface ProductClassSpecificationForSpecification :
  PdmFramework::PdmReferencedByRole { };
```

12.4 PdmConfigurationManagement IDL

```
// PdmConfigurationManagement.idl

#ifndef PDMCONFIGURATIONMANAGEMENT
#define PDMCONFIGURATIONMANAGEMENT

#include <CosTransactions.idl>
#include <CosLifeCycle.idl>
#include <CosCompoundLifeCycle.idl>

#include <PdmFoundation.idl>
#include <PdmFramework.idl>
#include <PdmManufacturingImplementation.idl>
#include <PdmProductStructureDefinition.idl>

module PdmConfigurationManagement
{
interface ProductClassExpression;
interface ProductClassSpecification;
interface SpecificationOperation;

typedef sequence<string> PartDiscipline;

// Exceptions

#define ITEM_CREATE_EXCEPTIONS \
     PdmFramework::PdmError, \
     PdmFramework::PermissionDenied, \
```

```
        PdmFramework::ValidationError, \
        PdmFramework::InvalidProperties, \
        PdmFramework::NotUnique

#define RELATIONSHIP_CREATE_EXCEPTIONS \
        PdmFramework::PdmError, \
        PdmFramework::PermissionDenied, \
        PdmFramework::ValidationError, \
        PdmFramework::InvalidProperties, \
        PdmFramework::NotUnique, \
        PdmFramework::CardinalityExceeded


// Entities

interface SpecificationOperation :
        CosTransactions::TransactionalObject,
        CosLifeCycle::LifeCycleObject, CosCompoundLifeCycle::Node,
        PdmFramework::Attributable { };

interface Configuration : CosLifeCycle::LifeCycleObject,
        CosTransactions::TransactionalObject,
        CosCompoundLifeCycle::Node, PdmFramework::Attributable,
        PdmFramework::Qualifiable
{
  attribute string name;
};

interface ConfigurationFactory
{
  Configuration create(
        in CosPropertyService::PropertySet property_set)
      raises(ITEM_CREATE_EXCEPTIONS);
};

interface ItemSolution : CosLifeCycle::LifeCycleObject,
        CosTransactions::TransactionalObject,
        CosCompoundLifeCycle::Node, PdmFramework::Attributable
{
  attribute string name;
};

interface ItemSolutionFactory
{
  ItemSolution create(
        in CosPropertyService::PropertySet property_set)
      raises(ITEM_CREATE_EXCEPTIONS);
};

interface ProductClass : PdmFramework::ManagedEntity
{
```

```
  attribute string description;
  attribute string name;
  attribute string level_type;
};

interface ProductClassFactory
{
  ProductClass create(in CosPropertyService::PropertySet
      property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};

interface ProductComponent : PdmFramework::ManagedEntity,
    PdmFramework::Qualifiable
{
  attribute string name;
  attribute string description;
  attribute boolean instance_required;
};

interface ProductComponentFactory
{
  ProductComponent create(
      in CosPropertyService::PropertySet property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};

interface ProductFunction : PdmFramework::ManagedEntity,
    PdmFramework::Qualifiable
{
  attribute string name;
  attribute string description;
  attribute PartDiscipline is_relevant_for;
};

interface ProductFunctionFactory
{
  ProductFunction create(
      in CosPropertyService::PropertySet property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};


interface Specification : SpecificationOperation,
    PdmFoundation::Manageable
{
  attribute string description;
  attribute string name;
  attribute boolean package;
};
```

```
interface SpecificationFactory
{
  Specification create(in CosPropertyService::PropertySet
       property_set,
       in SpecificationCategory category_for_specification)
    raises(ITEM_CREATE_EXCEPTIONS);
};

interface SpecificationCategory : PdmFramework::ManagedEntity
{
  attribute string description;
  attribute boolean mutually_exclusive;
};

interface SpecificationCategoryFactory
{
  SpecificationCategory create(in CosPropertyService::PropertySet
       property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};

interface SpecificationExpression : SpecificationOperation
{
  attribute string description;
  attribute string operation;
};

interface SpecificationExpressionFactory
{
  SpecificationExpression create(in CosPropertyService::PropertySet
       property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};

interface SpecificationInclusion :
    CosTransactions::TransactionalObject,
    CosLifeCycle::LifeCycleObject, CosCompoundLifeCycle::Node,
    PdmFramework::Attributable
{
  attribute string description;
};

interface SpecificationInclusionFactory
{
  SpecificationInclusion create(in CosPropertyService::PropertySet
       property_set)
    raises(ITEM_CREATE_EXCEPTIONS);
};

// Relationships
```

```
// ComponentFunction Relationship
//   role : ProductComponentForFunction
//     name: 'ProductComponentForFunction'
//     entity : ProductComponent
//     cardinality : 0..unbounded
//   role : ProductFunctionForComponent
//     name : 'ProductFunctionForComponent'
//     entity : ProductFunction
//     cardinality : 0..unbounded

interface ComponentFunction :
     PdmFramework::PdmReferenceRelationship { };

interface ComponentFunctionFactory
{
  ComponentFunction create(
        in CosPropertyService::PropertySet property_set,
        in ProductComponent product_component_for_function,
        in ProductFunction product_function_for_component)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductComponentForFunction :
   PdmFramework::PdmReferencesRole { };

interface ProductFunctionForComponent :
     PdmFramework::PdmReferencedByRole { };

// ComponentHierarchy Relationship
//   role : ProductComponentParent
//     name: 'ProductComponentParent'
//     entity : ProductComponent
//     cardinality : 0..unbounded
//   role : ProductComponentChild
//     name : 'ProductComponentChild'
//     entity : ProductComponent
//     cardinality : 0..unbounded

interface ComponentHierarchy :
PdmFramework::PdmReferenceRelationship { };

interface ComponentHierarchyFactory
{
ComponentHierarchy create(
     in CosPropertyService::PropertySet property_set,
     in ProductComponent product_component_parent,
     in ProductComponent product_component_child )
  raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};
```

```
interface ProductComponentParent :
  PdmFramework::PdmReferencesRole { };

interface ProductComponentChild :
  PdmFramework::PdmReferencedByRole { };


// ComponentSolution Relationship
//   role : ProductComponentForSolution
//     name: 'ProductComponentForSolution'
//     entity : ProductComponent
//     cardinality : 0..unbounded
//   role : ItemSolutionForComponent
//     name : 'ItemSolutionForComponent'
//     entity : ItemSolution
//     cardinality : 0..1

interface ComponentSolution :
  PdmFramework::PdmReferenceRelationship { };

interface ComponentSolutionFactory
{
  ComponentSolution create(
        in CosPropertyService::PropertySet property_set,
        in ProductComponent product_component_for_solution,
        in ItemSolution item_solution_for_component)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductComponentForSolution :
  PdmFramework::PdmReferencesRole { };

interface ItemSolutionForComponent :
     PdmFramework::PdmReferencedByRole { };

// ConfiguredItemUsage Relationship
//   role : ConfigurationForUsage
//     name: 'ConfigurationForUsage'
//     entity : Configuration
//     cardinality : 0..1
//   role : SolutionConfiguration
//     name : 'SolutionConfiguration'
//     entity : ItemSolution
//     cardinality : 0..unbounded
//   role : OperationConfiguration
//     name : 'OperationConfiguration'
//     entity : ProcessOperation
//     cardinality : 0..unbounded

interface ConfiguredItemUsage :
     PdmFramework::PdmReferenceRelationship { };
```

```
interface ConfiguredItemUsageFactory
{
  ConfiguredItemUsage create_using_solution(
        in CosPropertyService::PropertySet property_set,
        in Configuration configuration_for_usage,
        in ItemSolution solution_configuration)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
  ConfiguredItemUsage create_using_operation(
        in CosPropertyService::PropertySet property_set,
        in Configuration configuration_for_usage,
        in PdmManufacturingImplementation::ProcessOperation
          operation_configuration)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ConfigurationForUsage :
      PdmFramework::PdmReferencesRole { };

interface SolutionConfiguration :
      PdmFramework::PdmReferencedByRole { };

interface OperationConfiguration :
      PdmFramework::PdmReferencedByRole { };

// ConfiguredSpecificationSolution Relationship
//   role : ConfigurationForSolution
//     name: 'ConfigurationForSolution'
//     entity : Configuration
//     cardinality : 0..1
//   role : ConfigurationExpression
//     name : 'ConfigurationExpression'
//     entity : ProductClassExpression
//     cardinality : 0..unbounded
//   role : ConfigurationSpecification
//     name : 'ConfigurationSpecification'
//     entity : ProductClassSpecification
//     cardinality : 0..unbounded

interface ConfiguredSpecificationSolution :
      PdmFramework::PdmReferenceRelationship { };

interface ConfiguredSpecificationSolutionFactory
{
  ConfiguredSpecificationSolution create_using_specification(
        in CosPropertyService::PropertySet property_set,
        in Configuration configuration_for_solution,
        in ProductClassSpecification configuration_specification)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
  ConfiguredSpecificationSolution create_using_expression(
        in CosPropertyService::PropertySet property_set,
```

```
        in Configuration configuration_for_solution,
        in ProductClassExpression configuration_expression)
    raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ConfigurationForSolution :
    PdmFramework::PdmReferencesRole { };

interface ConfigurationExpression :
    PdmFramework::PdmReferencedByRole { };

interface ConfigurationSpecification :
    PdmFramework::PdmReferencedByRole { };

// DerivedSolution Relationship
//   role : OriginalItemSolution
//     name: 'OriginalItemSolution'
//     entity : ItemSolution
//     cardinality : 0..unbounded
//   role : DerivedItemSolution
//     name : 'DerivedItemSolution'
//     entity : ItemSolution
//     cardinality : 0..1

interface DerivedSolution :
    PdmFramework::PdmReferenceRelationship { };

interface DerivedSolutionFactory
{
  DerivedSolution create(
        in CosPropertyService::PropertySet property_set,
        in ItemSolution original_item_solution,
        in ItemSolution derived_item_solution)
    raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface OriginalItemSolution :
    PdmFramework::PdmReferencesRole { };

interface DerivedItemSolution :
    PdmFramework::PdmReferencedByRole { };

// FunctionHierarchy Relationship
//   role : ProductFunctionParent
//     name: 'ProductFunctionParent'
//     entity : ProductFunction
//     cardinality : 0..unbounded
//   role : ProductFunctionChild
//     name : 'ProductFunctionChild'
//     entity : ProductFunction
//     cardinality : 0..1
```

```
interface FunctionHierarchy :
     PdmFramework::PdmReferenceRelationship { };

interface FunctionHierarchyFactory
{
  ProductFunction create(
       in CosPropertyService::PropertySet property_set,
       in ProductFunction product_function_parent,
       in ProductFunction product_function_child )
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductFunctionParent :
     PdmFramework::PdmReferencesRole { };

interface ProductFunctionChild :
     PdmFramework::PdmReferencedByRole { };

// IncludedSpecification Relationship
//   role : SpecificationInclusionForOperation
//     name: 'InclusionForOperation'
//     entity : SpecificationInclusion
//     cardinality : 1..1
//   role : SpecificationOperationForInclusion
//     name : 'SpecificationOperationForInclusion'
//     entity : SpecificationOperation
//     cardinality : 0..unbounded

interface IncludedSpecification :
     PdmFramework::PdmReferenceRelationship { };

interface IncludedSpecificationFactory
{
  IncludedSpecification create(
       in CosPropertyService::PropertySet property_set,
       in SpecificationInclusion inclusion_for_operation,
       in SpecificationOperation operation_for_inclusion)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface SpecificationInclusionForOperation :
     PdmFramework::PdmReferencesRole { };

interface SpecificationOperationForInclusion :
     PdmFramework::PdmReferencedByRole {};

// InclusionIfCondition Relationship
//   role : InclusionIfOperation
//     name: 'InclusionIfOperation'
//     entity : SpecificationInclusion
```

```
//     cardinality : 1..1
//   role : IfOperationForInclusion
//     name : 'IfOperationForInclusion'
//     entity : SpecificationOperation
//     cardinality : 0..unbounded

interface InclusionIfCondition :
     PdmFramework::PdmReferenceRelationship { };

interface InclusionIfConditionFactory
{
  InclusionIfCondition create(
        in CosPropertyService::PropertySet property_set,
        in SpecificationInclusion inclusion_if_operation,
        in SpecificationOperation if_operation_for_inclusion)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface InclusionIfOperation :
     PdmFramework::PdmReferencesRole { };

interface IfOperationForInclusion :
     PdmFramework::PdmReferencedByRole { };

// ProductClassCategory Relationship
//   role : ProductUsingCategory
//     name: 'ProductUsingCategory'
//     entity : ProductClass
//     cardinality : 0..unbounded
//   role : ValidCategories
//     name : 'ValidCategories'
//     entity : SpecificationCategory
//     cardinality : 0..unbounded

interface ProductClassCategory :
     PdmFramework::PdmReferenceRelationship
{
  attribute boolean mandatory;
};

interface ProductClassCategoryFactory
{
  ProductClassCategory create(
        in CosPropertyService::PropertySet property_set,
        in ProductClass product_using_category,
        in SpecificationCategory valid_categories)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductUsingCategory :
     PdmFramework::PdmReferencesRole { };
```

```
interface ValidCategories :
      PdmFramework::PdmReferencedByRole { };

// ProductClassExpression Relationship
//   role : ProductUsingExpression
//     name: 'ProductUsingExpression'
//     entity : ProductClass
//     cardinality : 0..unbounded
//   role : ValidProductExpression
//     name : 'ValidProductExpression'
//     entity : SpecificationExpression
//     cardinality : 0..unbounded

interface ProductClassExpression : CosCompoundLifeCycle::Node,
      PdmFramework::PdmReferenceRelationship
{
   attribute string description;
   attribute string condition_type;
};

interface ProductClassExpressionFactory
{
   ProductClassExpression create(
         in CosPropertyService::PropertySet property_set,
         in ProductClass product_using_expression,
         in SpecificationExpression valid_product_expression)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductUsingExpression :
      PdmFramework::PdmReferencesRole { };

interface ValidProductExpression :
      PdmFramework::PdmReferencedByRole { };

// ProductClassFunction Relationship
//  role : ProductUsingFunction
//    name : 'ProductUsingFunction'
//    entity : ProductClass
//    cardinality : 0..unbounded
//  role : ValidProductFunction
//    name : 'ValidProductFunction'
//    entity : ProductFunction
//    cardinality : 0..unbounded

interface ProductClassFunction :
      PdmFramework::PdmReferenceRelationship
{
   attribute string description;
   attribute string relation_type;
```

```
};

interface ProductClassFunctionFactory
{
  ProductClassFunction create(
        in CosPropertyService::PropertySet property_set,
        in ProductClass the_product_using_function,
        in ProductFunction the_valid_product_function)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductUsingFunction :
      PdmFramework::PdmReferencesRole {};

interface ValidProductFunction :
      PdmFramework::PdmReferencedByRole{};

// ProductClassHierarchy Relationship
//   role : ProductClassSuperClass
//     name: 'ProductClassSuperClass'
//     entity : ProductClass
//     cardinality : 0..unbounded
//   role : ProductClassSubClass
//     name : 'ProductClassSubClass'
//     entity : ProductClass
//     cardinality : 0..1

interface ProductClassHierarchy :
      PdmFramework::PdmReferenceRelationship { };

interface ProductClassHierarchyFactory
{
  ProductClassHierarchy create(
        in CosPropertyService::PropertySet property_set,
        in ProductClass product_class_superclass,
        in ProductClass product_class_subclass)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductClassSuperClass :
       PdmFramework::PdmReferencesRole { };

interface ProductClassSubClass :
      PdmFramework::PdmReferencedByRole { };

// ProductClassInclusion Relationship
//   role : ProductUsingInclusion
//     name: 'ProductUsingInclusion'
//     entity : ProductClass
//     cardinality : 0..unbounded
//   role : ValidProductInclusion
```

```
//     name : 'ValidProductInclusion'
//     entity : SpecificationInclusion
//     cardinality : 0..unbounded

interface ProductClassInclusion :
      PdmFramework::PdmReferenceRelationship
{
  attribute string description;
};

interface ProductClassInclusionFactory
{
  ProductClassInclusion create(
        in CosPropertyService::PropertySet property_set,
        in ProductClass product_using_inclusion,
        in SpecificationInclusion valid_product_inclusion)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductUsingInclusion :
      PdmFramework::PdmReferencesRole { };

interface ValidProductInclusion :
      PdmFramework::PdmReferencedByRole { };

// ProductClassSpecification Relationship
//   role : ProductUsingSpecification
//     name: 'ProductUsingSpecification'
//     entity : ProductClass
//     cardinality : 0..unbounded
//   role : ValidProductSpecification
//     name : 'ValidProductSpecification'
//     entity : Specification
//     cardinality : 0..unbounded

interface ProductClassSpecification :
      CosCompoundLifeCycle::Node,
      PdmFramework::PdmReferenceRelationship
{
  attribute string association_type;
};

interface ProductClassSpecificationFactory
{
  ProductClassSpecification create(
        in CosPropertyService::PropertySet property_set,
        in ProductClass product_using_specification,
        in Specification valid_product_specification)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};
```

```
interface ProductUsingSpecification :
      PdmFramework::PdmReferencesRole { };

interface ValidProductSpecification :
      PdmFramework::PdmReferencedByRole { };

// ProductComponentSatisfiesSpecification Relationship
//   role: ProductComponentForSpecification
//     name: 'ProductComponentForSpecification'
//     entity: ProductComponent
//     cardinality: 0..unbounded
//   role: ProductClassSpecificationForSpecification
//     name: 'ProductClassSpecificationForSpecification'
//     entity: ProductClassSpecification
//     cardinality: 0..unbounded

interface ProductComponentSatisfiesSpecification :
      PdmFramework::PdmReferenceRelationship { };

interface ProductComponentSatisfiesSpecificationFactory
{
      ProductComponentSatisfiesSpecification create (
         in CosPropertyService::PropertySet property_set,
         in ProductComponent product_component,
         in ProductClassSpecification product_class_specification)
        raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ProductComponentForSpecification :
      PdmFramework::PdmReferencesRole { };

interface ProductClassSpecificationForSpecification :
      PdmFramework::PdmReferencedByRole { };

// ProductClassToComponent Relationship
//  role : RootProduct
//    name : 'RootProduct'
//    entity : ProductClass
//    cardinality : 0..unbounded
//  role : ValidProductComponent
//    name : 'ValidProductComponent'
//    entity : ProductComponent
//    cardinality : 0..unbounded

interface ProductClassToComponent :
      PdmFramework::PdmReferenceRelationship { };

interface ProductClassToComponentFactory
{
   ProductClassToComponent create(
         in CosPropertyService::PropertySet property_set,
```

```
        in ProductClass the_root_product,
        in ProductComponent the_valid_product_component)
      raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface RootProduct : PdmFramework::PdmReferencesRole { };

interface ValidProductComponent :
      PdmFramework::PdmReferencedByRole{ };

// SolutionPartMaster Relationship
//   role : ItemSolutionForMaster
//     name: 'ItemSolutionForMaster'
//     entity : ItemSolution
//     cardinality : 0..unbounded
//   role : PartMasterForSolution
//     name : 'PartMasterForSolution'
//     entity : PartMaster
//     cardinality : 0..unbounded

interface SolutionPartMaster :
PdmFramework::PdmReferenceRelationship { };

interface SolutionPartMasterFactory
{
SolutionPartMaster create(
      in CosPropertyService::PropertySet property_set,
      in ItemSolution item_solution_for_master,
      in PdmProductStructureDefinition::PartMaster
         part_master_for_solution )
   raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ItemSolutionForMaster :
   PdmFramework::PdmReferencesRole { };

interface PartMasterForSolution :
   PdmFramework::PdmReferencedByRole { };

// SpecificationCategoryComposition Relationship
//   role : SpecificationCategoryForSpecifications
//     name: 'SpecificationCategoryForSpecifications'
//     entity : SpecificationCategory
//     cardinality : 0..unbounded
//   role : SpecificationForSpecificationCategory
//     name : 'SpecificationForSpecificationCategory'
//     entity : Specification
//     cardinality : 1..1

interface SpecificationCategoryComposition :
      PdmFramework::PdmContainmentRelationship { };
```

```
interface SpecificationCategoryForSpecifications :
     PdmFramework::PdmContainsRole { };

interface SpecificationForSpecificationCategory :
     PdmFramework::PdmContainedInRole { };

// SpecificationExpressionOperands Relationship
//   role : ExpressionForOperands
//     name: 'ExpressionForOperands'
//     entity : SpecificationExpression
//     cardinality : 0..unbounded
//   role : OperandForExpression
//     name : 'OperandForExpression'
//     entity : SpecificationOperand
//     cardinality : 0..unbounded

interface SpecificationExpressionOperands :
     PdmFramework::PdmReferenceRelationship { };

interface SpecificationExpressionOperandsFactory
{
  SpecificationExpressionOperands create(
       in CosPropertyService::PropertySet property_set,
       in SpecificationExpression expression_for_operands,
       in SpecificationOperation operand_for_expression)
     raises(RELATIONSHIP_CREATE_EXCEPTIONS);
};

interface ExpressionForOperands :
     PdmFramework::PdmReferencesRole { };

interface OperandForExpression :
     PdmFramework::PdmReferencedByRole { };
};

#endif
```

PdmStepModule 13

Contents

This chapter contains the following sections.

Section Title	Page
“Overview”	13-1
“PdmStep Model”	13-2
“PdmStep Description”	13-2
“PdmStep IDL”	13-5

13.1 Overview

STEP and PDM are closely related. The Standard for the Exchange of Product data (STEP), and PDM provides enablers to manage product data. The PDM Enablers interface models are guided by STEP. However, the PDM models are not a representation of STEP Application Interpreted Models (AIM). Rather the PDM models represent user-level objects that are analogous (but not identical) to corresponding Application Resource Models (ARM) from various Application Protocols. The idea is that this kind of entity is more suitable for the dynamic runtime inter-operation that may occur between PDM client and PDM server or between PDM servers of different vendors. Moreover, this module specifies STEP as a means for the interchange of PDM data between PDM systems, or between PDM and CAD systems using a STEP Application Protocol such as AP203 or AP214.

All interfaces in this module are mandatory, except for the **export_baseline()** operation in the **StepTranslator** interface, which is optional.

13.2 PdmStep Model

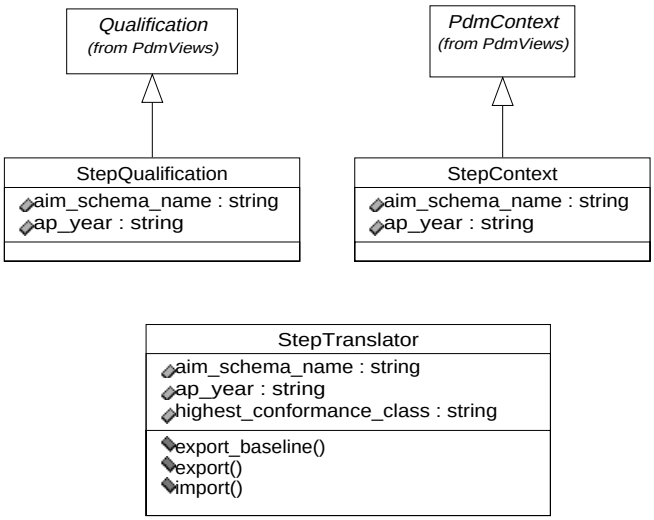

Figure 13-1 PdmStep Model Diagram

13.3 PdmStep Description

13.3.1 StepTranslator

The **StepTranslator** performs two functions: export PDM objects into a **File**, and import PDM objects from a **File**. Each instance of a **StepTranslation** supports a specific STEP application protocol. Given a **ManagedEntity**, the **StepTranslator.export()** operation is able to select related objects for export using the context parameter. While this module defines **StepQualification**, objects need not be explicitly qualified with a **StepQualification** in order to be selected for export. See Section 13.3.3, “StepQualification,” on page 13-4 for more information on how **StepQualification** is used.

```
interface StepTranslator
{
  attribute string aim_schema_name;
  attribute string ap_year;
  attribute string highest_conformance_class;

  PdmDocumentManagement::SecuredFile export_baseline(
```

```
        in PdmBaseline::BaselineIteration base,
        in string conformance_class,
        in string file_format)
    raises(NotImplemented,
        TranslationError,
        PdmFoundation::ValidationError,
        PdmFoundation::PermissionDenied,
        PdmFoundation::PdmError);
  PdmDocumentManagement::SecuredFile export(
        in PdmFramework::ManagedEntity item,
        in PdmViews::PdmContext view_context,
        in string conformance_class,
        in string file_format)
    raises(TranslationError,
        PdmFoundation::ValidationError,
        PdmFoundation::PermissionDenied,
        PdmFoundation::PdmError);
  void import(in PdmDocumentManagement::SecuredFile file)
    raises(TranslationError,
          PdmFoundation::ValidationError,
          PdmFoundation::PermissionDenied,
          PdmFoundation::PdmError);
};
```

aim_schema_name

The ISO EXPRESS schema name of an application interpreted model that is supported by this **StepTranslator**. This defines the Application Protocol supported by this translator. The name value shall be the same as that of the **FILE_SCHEMA** attribute in an ISO-10303-21 file. For example, **CONFIG_CONTROL_DESIGN**. The name is not case-sensitive.

ap_year

The year of approval of the ISO EXPRESS schema **aim_schema_name**. For example, 1994 in the case of AP203.

highest_conformance_class

The highest STEP AP conformance class supported by this **StepTranslator**. For example, CC1. The value is not case sensitive.

export_baseline

Creates a new STEP file that contains the items associated with the **BaselineIteration**. The STEP file shall contain all the items required by the constraints contained in the application protocol supported by this **StepTranslator** and the **conformance_class** parameter. The syntax of the returned STEP file shall be designated by the **file_format** parameter. Currently the valid value is ISO10303-21. This operation is optional. If not implemented, it shall throw a NotImplemented exception.

export

Creates a new STEP file that contains the items described by the **ManagedEntity** and all its children as qualified by the **Context**. The STEP file shall contain all the items required by the constraints contained in the application protocol and **conformance_class** supported by this **StepTranslator**. The syntax of the returned STEP file shall be designated by the **file_format** parameter. Currently the valid value is ISO10303-21.

import

Instantiates new PDM entities corresponding to the entities contained in the **SecuredFile**. The **SecuredFile** shall be encoded according to ISO 10303-21.

13.3.2 StepContext

The **StepContext** is a **PdmContext** which indicates which STEP Application Protocol shall be used as the basis of a particular STEP translation.

```
interface StepContext : PdmViews::PdmContext
{
  attribute string aim_schema_name;
  attribute string ap_year;
};

interface StepContextFactory
{
  StepContext create(in string aim_schema_name, in string ap_year)
    raises (PdmFoundation::ValidationError,
            PdmFoundation::PermissionDenied,
            PdmFoundation::PdmError);
};
```

aim_schema_name

The ISO EXPRESS schema name of an application interpreted model, which supports this qualification.

ap_year

The year when the application protocol, which contains the application interpreted model, was approved.

13.3.3 StepQualification

A **StepQualification** is a type of **Qualification**. It indicates that the object to which it is attached is applicable to a particular type of STEP translation. It is typically used to filter out objects that may be applicable for one STEP translation, but not another.

For example, some documents associated with a part may be applicable for AP214 while others may only be applicable for AP203. A **StepTranslator** for AP203 can use the **StepQualification** to exclude those documents explicitly qualified for AP214.

```
interface StepQualification : PdmViews::Qualification
{
  attribute string aim_schema_name;
  attribute string ap_year;
};

interface StepQualificationFactory
{
  StepQualifiation create(in string aim_schema_name, in string ap_year)
    raises (PdmFoundation::ValidationError,
        PdmFoundation::PermissionDenied,
        PdmFoundation::PdmError);
};
```

aim_schema_name

The ISO EXPRESS schema name of an application interpreted model that supports this qualification.

ap_year

The year when the application protocol, which contains the application interpreted model, was approved.

13.4 PdmStep IDL

```
// PdmStep.idl

#ifndef  PDMSTEP
#define  PDMSTEP

#ifdef   SOM_COMPILE
#include <somobj.idl>                  // SOM COMPILE
#endif

#ifdef  ORBIX_COMPILE
#ifndef  IFR
#define  IFR
#include <ifr.idl>
#endif
#endif

#include <PdmBaseline.idl>
#include <PdmDocumentManagement.idl>
#include <PdmFramework.idl>
#include <PdmViews.idl>
```

```
module PdmStep
{
exception NotImplemented
{
  unsigned long error_code;
  string error_text;
};

exception TranslationError
{
  unsigned long error_code;
  string error_text;
};

interface StepTranslator
{
  attribute string aim_schema_name;
  attribute string ap_year;
  attribute string highest_conformance_class;

  PdmDocumentManagement::SecuredFile export_baseline(
       in PdmBaseline::BaselineIteration base,
       in string conformance_class,
       in string file_format)
    raises(NotImplemented,
           TranslationError,
           PdmFoundation::ValidationError,
           PdmFoundation::PermissionDenied,
           PdmFoundation::PdmError);
  PdmDocumentManagement::SecuredFile export(
       in PdmFramework::ManagedEntity item,
       in PdmViews::PdmContext view_context,
       in string conformance_class,
       in string file_format)
    raises(TranslationError,
           PdmFoundation::ValidationError,
           PdmFoundation::PermissionDenied,
           PdmFoundation::PdmError);
  void import(in PdmDocumentManagement::SecuredFile file)
    raises(TranslationError,
           PdmFoundation::ValidationError,
           PdmFoundation::PermissionDenied,
           PdmFoundation::PdmError);
};

interface StepContext : PdmViews::PdmContext
{
  attribute string aim_schema_name;
  attribute string ap_year;
};
```

```
interface StepContextFactory
{
  StepContext create(in string aim_schema_name, in string ap_year)
    raises (PdmFoundation::ValidationError,
          PdmFoundation::PermissionDenied,
          PdmFoundation::PdmError);
};

interface StepQualification : PdmViews::Qualification
{
  attribute string aim_schema_name;
  attribute string ap_year;
};

interface StepQualificationFactory
{
  StepQualification create(in string aim_schema_name,
       in string ap_year)
    raises (PdmFoundation::ValidationError,
          PdmFoundation::PermissionDenied,
          PdmFoundation::PdmError);
};

};

#endif
```

Mapping the Product Development Process to the PDM Enablers A

A.1 Eleven Sub-processes of Product Development Process

A.1.1 Mapping of PDM enablers to sub-processes

PDM enablers may cover more than one sub-process. Furthermore, more than one PDM enabler may be required to address a specific sub-process. The following represents a mapping of the sub-processes.

- When submitters describe their interfaces for the PDM enablers, their responses shall be placed within a particular sub-process. Submitters may describe how the technology service applies to multiple sub-processes. Submissions should follow the exact pattern of this mapping. That is, a submission does not need to supply all the enablers, but those that are proposed must conform to the mapping. For example, a submitter of a PDM enabler for Request for Engineering Action, number 1 in the enabler mapping, should describe how it maps into the three sub-processes: Develop Strategic Product Plan; Develop Product Definition; and Develop Product Design.
- The MDTF will consider submissions that propose a different mapping but this will require information from the submitter describing why a deviation from the mapping below is preferred.

This section provides a detailed mapping from the Product Development subprocesses to IDL interfaces and operations that are specified in the PDM modules. This shows how the proposed specifications enable product data management in the subprocesses defined by the RFP.

Additional facilities beyond the PDM enabler interfaces will be used to fulfill many of the subprocesses. For example, to find data, the user may use interactive non-CORBA features of the PDM system, may use interactive interfaces that use the Query Service in conjunction with the PDM enablers, or may use other systems like a Parts Classification system with a connection to the PDM enablers.

A.2 Develop Strategic Product Plan

- Develop Strategic Product Plan (1), (4).
- Determine and identify product areas for business.
- Determine market decisions (that is, mapping of manufacturing strategy against new development areas).
- Determine design and manufacturing responsibilities .
- Project cost.
- Project volumes.

The following conditions are assumed:

Abstracts for copyrighted publications and reports are made available in the PDM system using the following service:

PdmDocumentManagement::SecuredFile

Most major on-line market research providers (for example, IDC, Gartner Group) provide electronic files of their research via the Web. These references can be indexed in the PDM system through interfaces provided by using the UnsecuredFile interface with a type attribute which designates URLs.

Customer interviews and surveys are made available as bulk data within the PDM system using the following service:

PdmDocumentManagement::SecuredFile

Sales reports are made available as bulk data within the PDM system using the following service:

PdmDocumentManagement::SecuredFile.

This data may be associated to DocumentIterations through the DocumentFileRelationship interface.

A.2.1 Determine and Identify Product Areas for Business

1. Access market research reports.
   - IdentificationContext, Identifiable
   - DocumentFileRelationship
2. Access customer interviews and surveys.
   - IdentificationContext, Identifiable
   - DocumentFileRelationship
3. Access technology research reports.
   - IdentificationContext, Identifiable
   - DocumentFileRelationship
4. Access sales reports.
   - IdentificationContext, Identifiable
   - DocumentFileRelationship

5. Access trade surveys and publications.
   - IdentificationContext, Identifiable
   - DocumentFileRelationship
6. Access previous Strategic Project Plan Documents.
   - IdentificationContext, Identifiable
   - DocumentFileRelationship
7. Create new Strategic Project Plan Document.
   - SecuredFile to register the files containing the project plan.
   - DocumentFileRelationship to associate the project plan files and source data (for example, customer surveys) with the project plan document.
   - Dependency relationship to associate the project plan document with any other planning documents that it incorporates, as in the development of derivative, hybrid, or enhancement products.
8. Refine Strategic Project Plan.
   - DocumentIteration::create_new_iteration(), Lockable, SecuredFile, Stateable, to iteratively refine the details of the strategic project plan.

A.2.2 Determine Market Decisions

1. Access Manufacturing Strategy Document.
   - IdentificationContext, Identifiable
   - DocumentFileRelationship

A.2.3 Determine Design and Manufacturing Responsibilities

1. Collect research, operational, product, process and competitive position data:
   - Identifiable
   - Relationship navigation with PdmContext, to find qualified documents.
   - DocumentIteration, DocumentFileRelationship and Dependency
2. Create new Design and Manufacturing Plan Documents.
   - SecuredFile to register the file(s) containing the plans.
   - DocumentFileRelationship to associate the plan file(s) and source data (for example, production research, personnel profiles) with the plan documents.
   - Dependency to associate the plan document with any other documents that it incorporates or references.
3. Refine Design and Manufacturing Plans.
   - DocumentIteration::create_new_iteration(), Lockable, SecuredFile, Stateable, to iteratively refine the details of the strategic project plan.

A.2.4 Project Cost

1. Collect ownership and operational costs for manufacturing resources and processes.

- DocumentFilerelationship, SecuredFile, UnsecuredFile
- PartDocument relationship
- DocumentIteration, DocumentFileRelationship and Dependency.

2. Create Cost Projection.
   - DocumentMaster, DocumentRevision, DocumentIteration to create the new document.
   - SecuredFile to register the file(s) containing the forecast.
   - DocumentFileRelationship to associate the cost projection file(s) and source data (for example, cost of inventory, cost of handling) with the new document.
   - Dependency to associate the cost projection document with any other documents which it incorporates or references.
3. Refine Cost Forecast.
   - DocumentIteration, Lockable, SecuredFile, Stateable to iteratively refine the details of the cost projection.

A.2.5 Project Volumes

1. Collect demand, capacity and capacity utilization data.
   - Identifiable, PartRevision, PartDocumentRelationship, DocumentIteration, DocumentFileRelationship and Dependency.
2. Create Volume Projection.
   - DocumentMaster, DocumentRevision, DocumentIteration to create the new documents. SecuredFile to register the file(s) containing the forecast.
   - DocumentFileRelationship to associate the volume projection file(s) and source data (for example, skills inventory, facility plans/layouts) with the new document.
   - Dependency to associate the volume projection document with any other documents which it incorporates or references.
3. Refine Volume Projection.
   - DocumentIteration, Lockable, SecuredFile, Stateable to iteratively refine the details of the volume projection.

A.3 Develop Product Business Plan

- Develop Product Business Plan (1), (4).
- Determine an approach to product design to create product at estimate cost for volume determined.
- Create development plan with schedules and estimates.

A.3.1 Determine an Approach to Product Design to Create Product at Estimated Cost for Volume Determined

1. Identify constraints and associated costs for existing or new manufacturing processes.
   - Identifiable, DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, and DocumentFileRelationship
2. Identify important connections between design choices and manufacturing system performance.
   - Identifiable, DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, and DocumentFileRelationship
3. Establish key dimensions of product architecture and assess impact on the manufacturing system.
   - Identifiable, DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, and DocumentFileRelationship
4. Create product design approach that reflects the above critical relationships to manufacturing processes.
   - DocumentMaster, DocumentRevision, DocumentIteration to create the new documents. SecuredFile to register the files containing the description of the product design approach.
   - DocumentFileRelationship to associate the product design approach document and supporting data (for example, design checklists, procedures for calculating manufacturing costs, specification of recommended/authorized standard part catalogs).

A.3.2 Create Development Plan with Schedules and Estimates

1. Define development project scope and objectives.
   - Identifiable, DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, and DocumentFileRelationship
2. Identify project staff and organization.
   - Identifiable, DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, and DocumentFileRelationship
3. Define project management roles, expectations, and protocols.
   - Identifiable, DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, and DocumentFileRelationship
4. Define project phases, milestones, tasks, task management, and sequencing.
   - Identifiable, DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, and DocumentFileRelationship
5. Schedule project tasks, milestones, and phase reviews using resource assignments and estimates.

- Identifiable, DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, and DocumentFileRelationship

6. Specify product design, testing, and prototyping methodologies.
   - Identifiable, DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, and DocumentFileRelationship
7. Establish senior management reviews and controls.
   - Identifiable, DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, and DocumentFileRelationship
8. Define mechanism for project plan corrections.
   - Identifiable, DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, and DocumentFileRelationship

A.4 Develop Product Definition

- Develop Product Definition (1), (2a), (4), (5a).
- Develop Product General Specification.
- Develop Product Conceptual Layout.
- Develop Mockups and Visuals.
- Identify relationship to existing parts and Manufacturing Capabilities.
- Establish Development Plan and Objectives.
- Estimate Product Costs by major component.
- Manage Library of Existing Modules (parts).

A.4.1 Develop Product General Specification

1. View market research information to determine features needed.
   - Identifiable, DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, and DocumentFileRelationship if you know the names of the relevant documents.
2. Convert marketing feature requirements into engineering specifications.
   - DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, DocumentFileRelationship to register the file containing the specifications and associate them to a document.
   - Derive to record the relationship between the documents.

A.4.2 Develop Product Conceptual Layout

1. Create initial product structure.
   - PartMaster, PartRevision to create nodes in product structure.
   - Usage to add nodes to product structure.

2. Document the specifications for each node in the product structure.
   - DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, DocumentFileRelationship to register the file containing the specifications and associate them to a document.
   - PartDocumentRelationship to associate specification document with part.

A.4.3 Develop Mockups and Visuals

1. Create artists renderings for concepts under consideration.
   - DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, DocumentFileRelationship to register the file containing the specifications and associate them to a document.
   - PartDocumentRelationship to associate the document with a product.

A.4.4 Identify Relationship to Existing Parts and Manufacturing Capabilities

1. Define specifications of a part relative to another part (for example, new part like #7861-3 with larger flange).
   - Including reference in text does not require any PDM capability.
   - Derive to make the relationship between parts known to PDM.
2. Include existing and standard parts in the product structure.
   - Usage. Same method as for adding a new part to a product structure.
3. Note potential sources (including both external vendors and internal facilities) for parts.
   - PartSupplierRelationship, DesignSupplierRelationship

A.4.5 Establish Development Plan and Objectives

1. Estimate and record resources required for development of each part.
   - PartDataIteration, Attributable
2. Roll up resource requirements through the product structure or functional breakdown.
   - Usage to find the components.
   - PartDataIteration, Attributable to get information about components.
   - PartDataIteration, Attributable to record the information for the assembly.

A.4.6 Estimate Product Cost by Major Component

1. Access actual cost information for similar components.
   - PartMaster, PartRevision, PartDataIteration, Attributable in conjunction with a Part Classification system or interactive query to find the similar components.
   - PartDataIteration, Attributable for information that is stored as attributes.

- PartDocumentRelationship, DocumentIteration to get to information that is stored inside documents.

2. Record the estimate.
   - PartDataIteration, Attributable
3. Roll up cost information through the product structure.
   - Usage to find the components.
   - PartDataIteration, Attributable to get information about components.
   - PartDataIteration, Attributable to record the information for the assembly.

A.4.7 Manage Libraries of Existing Modules (Parts)

1. Classify parts according to part type.
   - By a Parts Classification system, possibly integrated with the PDM System.
2. Manage parameters that are specific to the type of part (for example, number of teeth for gears).
   - Parts Classification or PartDataIteration, Attributable
3. Define new part types.
   - By a Parts Classification system.

A.5 Define Product Marketing Configuration and Rules

- Define Product Marketing Configuration and Rules (2a), (4), (5a), (6), (7)
- Define features and options available on product
- Project sales volumes and cost by options

A.5.1 Define Features and Options Available on Product

PdmConfigurationManagement module

1. Associate products with a product family.
   - ProductRootToComponent
2. Organize product families into a hierarchy.
   - ProductClassHierarchy
3. Define features available in a product family or product.
   - ProductClassSpecification
4. List choices available for each feature
   - SpecificationCategoryComposition
5. List restrictions on feature choices (and combinations of feature choices) for product families and products.
   - SpecificationExpression, ProductClassInclusion

A.5.2 Project Sales Volumes and Cost by Options

PdmConfigurationManagement, PdmProductStructureDefinition, PdmDocumentManagement module

A.6 Develop Product Design

- Develop Product Design (2a), (2b), (4), (5a), (6)
- Develop Major Component General Specifications
- Develop Major Component Layout
- Retrieve Reusable Parts and Major components
- Incorporate Reusable parts and major components
- Establish local reusable parts and major components
- Establish Design Plan and record status
- Define Module (sub-assembly) content with optional components
- Develop Part Design
- Select preferred features

A.6.1 Develop Major Component General Specifications

1. Access general specifications of higher level systems in the product structure (or functional breakdown).
   - Usage to get the higher level systems.
   - PartDocumentRelationship to get the specification documents.
   - PdmDocumentManagement module routines to view the specification documents.
2. Provide feedback when general specifications of higher level systems are not feasible or are overly conservative.
   - EngChangeIssue to create an issue object, and EngChangeAffect, Changeable to associate it to the questionable specifications.

A.6.2 Develop Major Component Layout

- PartMaster, PartRevision, PartStructure, Usage.

A.6.3 Retrieve Reusable Parts and Major Components

1. Query over a part class by attributes specific to that part class (for example, piston diameter).
   - By a Part Classification system in conjunction with PartRevision, PartDataIteration.
2. Access functional specifications for candidate parts.

- PartDocumentRelationship to get the specification documents.
- PdmDocumentManagement module interfaces to view the specification documents.

A.6.4 Incorporate Reusable Parts and Major Components

1. Insert a part into the product structure of multiple products.
   - Usage
2. For major components that will be reused with modification, copy the product structure and edit the copy.
   - PartMaster, PartRevision, PartStructure, PartStructureIteration to create the new copy.
   - Usage to get the components of the original part.
   - Usage to add components of original to copy.
   - PartDocumentRelationship to get documents describing original.
   - DocumentMaster, DocumentRevision, DocumentIteration to create documents for the copy.
   - PartDocumentRelationship to add documents to copy.

A.6.5 Establish Local Reusable Parts and Major Components

A.6.6 Establish Design Plan and Record Status

1. Associate a responsible person and a budget with parts and assemblies.
   - PartRevision, PartDataIteration, Manageable, ObjectOwner, Attributable to modify and retrieve these attributes.
2. Record life cycle state of parts and assemblies.
   - PartRevision, Stateable to modify and retrieve these attributes.

A.6.7 Define Module (Subassembly) Content with Optional Components

1. Restrict the applicability of product structure relationships based on arbitrary expressions of options.
   - PdmConfigurationManagement::Specification,
   - PdmConfigurationManagement::SpecificationExpression.

A.6.8 Develop Part Design

1. Associate part definition data with a part. The definition data would be geometry for mechanical parts; schematics and/or behavioral specifications for electrical parts; and part-specific attributes (also called loose attributes) (e.g., orifice_diameter) to any type of part.
   - SecuredFile, DocumentFileRelationship to register the data items.

- DocumentMaster, DocumentRevision, DocumentIteration to create the part definition documents.
- DocumentFileRelationship to associate the data items to the document.
- PartDocumentRelationship to associate the document with the part.
- PartDataIteration, Attributable to modify part specific attributes.

2. Access definition data for related parts such as mating parts.
   - PartStructureIteration, Usage, Dependency to find the related parts.
   - PartDocumentRelationship to get the definition data of the related parts.
3. Provide feedback to other designers when changes to their parts would improve the overall design.
   - EngIssueItem to create an issue object.
   - EngChangeAffect to associate it to the questionable specifications.

A.6.9 Select Preferred Features

This process involves selecting from among several alternative design concepts.

1. Designate one part (or part version) as a competing alternative to another, either in general or in a particular context. Until the PartRevision is approved, this means that the alternative is being explored, not that it is an acceptable substitute in any context.
   - Alternate or Substitute to designate the relationship.
2. Remove competing alternatives from consideration.
   - Alternate or Substitute
3. Select one of the competing alternatives as the new primary design for further consideration.
   - Usage to determine where the primary design was used
   - Usage to put the selected alternative in the product structure in place of the previous design.

PdmConfigurationManagement can be used to identify features.

A.7 Develop Process Design and Procurement Agreements

- Develop Process Design and Procurement Agreements (2a), (2b), (3), (4), (5b), (6)
- Develop assembly process design
- Develop manufacturing facility plan
- Assess manufacturing capabilities
- Determine part sourcing
- Procure machine tools, firm tools and services
- Develop production tooling design

- Develop part fabrication process design
- Develop procurement contracts

A.7.1 Develop assembly process design

Process operations and steps can be defined with the PdmManufacturingImplementation module.

A.7.2 Develop manufacturing facility plan

PdmDocumentManagement

A.7.3 Assess Manufacturing Capabilities

1. Retrieve final product (assembly) design, components and volume projections.
   - PartDocumentRelationship, using the appropriate StateContext, or
   - Identifiable, DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, and DocumentFileRelationship
   - Usage, using the final design ViewContext
2. Identify other products or product families with similar requirements. (Non-PDM activity.) Retrieve process specifications and facility specifications for those products.
   - PartMaster, PartRevision, Identifiable
   - PartDocumentRelationship
3. Identify the major assembly processes needed to make the product. Identify available facilities which house or can house those processes. Determine available capacity in those facilities. (Non-PDM activity.)
4. Create assembly capabilities assessment report:
   - DocumentMaster, DocumentRevision
   - SecuredFile, DocumentFileRelationship to associate the report file to the Document object
   - PartDocumentRelationship to associate the report to the Product object
5. For each component, retrieve the component design and materials specifications:
   - PartDocumentRelationship
6. Identify other Parts with similar processing requirements. (Non-PDM activity.)
   - Retrieve process specifications and facility specifications for those products.
   - Partmaster, PartRevision, Identifiable
   - PartDocumentRelationship
7. Identify the major fabrication processes needed to make the product. Identify available facilities which house or can house those processes. Determine available capacity in those facilities. (Non-PDM activity.)

8. Create fabrication capabilities assessment report:
   - DocumentMaster, DocumentRevision
   - SecuredFile, DocumentFileRelationship to associate the report file to the Document object
   - PartDocumentRelationship to associate the report to the component Part

A.7.4 Determine Part Sourcing

1. Retrieve the component list for the product:
   - Usage
2. For each component part, retrieve the fabrication/assembly capabilities assessment:
   - PartDocumentRelationship
3. For each component, make the Make/Buy decision and record the decision:
   - Set PartDataIteration::make_buy
4. Create the buy list of components and per-product quantities to be purchased:
   - DisciplineQualification to create a Qualification consisting of the buy-list view
   - Usage to copy the buy-list component (and quantity) to the buy-list for the Part
   - Qualifies to associate the buy-list qualification to the buy-list component
5. For the each component to be fabricated, identify the facility (or facilities) and the initial capacity allocation. (Non-PDM activity) Record the decisions:

DocumentMaster, DocumentRevision for the facility selection data

- SecuredFile, DocumentFileRelationship to associate the facility selection data to the Document object
- PartDocumentRelationship to associate the facility selection Document to the component Part

A.7.5 Develop (Part) Procurement Contracts

1. Retrieve the buy list for the Product
   - DisciplineContext to do retrievals with the buy-list view
   - Usage to get the components and quantities on the buy-list
2. For each component, obtain the design specifications for the part:
   - PartDocumentRelationship using the appropriate StateContext
3. Create a draft technical specification:
   - DocumentMaster, DocumentRevision to create the Technical Specification Package
   - SecuredFile, DocumentFileRelationship to associate the specification documents/files to the Tech Spec Package

4. Create a draft RFQ, and associate the standard boilerplate and the technical specifications. The RFQ has several sections, each of which can consist of several standard files. The section that is the Technical Specification can consist of several subsections which are the various forms of Part specification. For each of these, the document_type is, in effect, the name of the subsection.
   - DocumentMaster, DocumentRevision to create the RFQ document
   - DocumentFileRelationship, to associate each boilerplate file to the RFQ. Some boilerplate files may be templates which are edited into a version specifically for this contract. A contract-specific file is captured in the PDM via SecuredFile before attaching it.
   - DocumentFileRelationship to associate the technical specification to the RFQ
   - PartDocumentRelationship to associate the RFQ to the component part
5. Determine a list of possible suppliers, issue RFQ, etc. (Non-PDM activity.)
6. When a contract is firm, make a new business item, copying all the items from the draft contract, and attaching supplier-specific riders, etc.
   - DocumentMaster, DocumentRevision to create the Contract object
   - Derive to relate the actual contract to the RFQ

SecuredFile or UnsecuredFile DocumentFileRelationship and DocumentFileRelationship, to associate rider files and signed paper copies to the Contract object

A.7.6 Develop Part Fabrication Process Design

1. Retrieve part model (geometry and topology, tolerances, etc.), plus market projections, cost and time constraints:
   - PartDocumentRelationship
2. Retrieve the target fabrication facilities (from Part sourcing, above):
   - PartDocumentRelationship
3. Create the Manufacturing Specification Package, to contain various process specification documents:
   - DocumentMaster, DocumentRevision of type Manufacturing Specification Package
   - PartDocumentRelationship to associate the Manufacturing Specification Package to the component part
4. If necessary, create one Manufacturing Specification Package per target manufacturing facility, and construct the necessary qualification:
   - LocationQualification
5. Identify the specific stock materials to be used and create initial Bill of Materials:
   - PartStructure
6. Identify manufacturing features, choose major fabrication processes, and store initial Process Plan in the Manufacturing Specification Package:

- DocumentMaster, DocumentRevision of type Process Plan
- SecuredFile, DocumentFileRelationship to associate the process plan definition file to the Document object
- SecuredFile, DocumentFileRelationship to associate the Process Plan Document to the Manufacturing Specification Package.

7. If necessary, initiate Request for Engineering Action to engineer a new process and associate Process requirements object and relevant Part Model and manufacturing features
   - DocumentMaster, DocumentRevision of type Process Requirements
   - SecuredFile, DocumentFileRelationship to associate the requirements definition file to the Document object
   - EngChangeRequest to create the Request for Engineering Action, of type Process Design
   - EngChangeAffect to associate the requirements definition and relevant Part model documents
8. Determine what features are to be measured or otherwise inspected and what the qualifying criteria are. Create preliminary Inspection Plan
   - Document, SecuredFile, DocumentFileRelationship as in step 6, to associate the Inspection Plan Document to the Manufacturing Specification Package.
9. Choose the (type of) machine and operator skills to be used for each major process (fabrication or inspection). Create initial Resource Requirements:
   - Document, SecuredFile, DocumentFileRelationship, as in step 6, to associate the Resource Requirements Document to the Manufacturing Specification Package.
10. For each major process, specify the component operations and parameters. Replace the initial Process Plan:
    - Identifiable, ProcessRevision to get the initial Process Plan
    - or PartDocumentRelationship, to get the appropriate Manufacturing Specification Package, and then PdmManufacturingImplementation::Documentation to get the initial Process Plan
    - Detach the initial specification file
    - SecuredFile, DocumentFileRelationship to associate the new (complete) specification file to the Document object
    - Create the initial Routing: ProcessRevision, Documentation to associate the Routing Document to the Manufacturing Specification Package.
11. From the detailed process plans, create preliminary Cost Estimates:
    - ProcessRevision, DocumentRevision, SecuredFile, DocumentFileRelationship, to associate the Cost Estimate document to the Manufacturing Specification Package
12. Review and evaluate the plans. This may result in revisions of any of the Manufacturing Specifications, initial Bill of Materials, or Cost Estimates.
    - Identifiable, ProcessRevision, Documentation, DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, and DocumentFileRelationship, to get the previous version of the specification

- or PartDocumentRelationship, to get the appropriate Manufacturing Specification Package, and then get the previous version of the specification
- ProcessRevision to make a new version of the specification
- Detach the previous specification file from the new Document
- SecuredFile, DocumentFileRelationship to associate the new specification file to the Document object

13. Record initial signoff and release preliminary cost estimates and resource requirements to production planners. For each of Bill of Materials, Cost Estimates, Manufacturing Specification Package:
    - Identifiable, ProcessRevision, Docmentation, DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, and DocumentFileRelationship, or PartDocumentRelationship, to get the appropriate Manufacturing Specification Package
    - Stateable to change the state of the specification
14. When necessary, initiate Design Change negotiations
    - DocumentMaster, DocumentRevision to contain the marked-up design,
    - SecuredFile, DocumentFileRelationship to associate the markup file to the Document object
    - EngChangeIssue to create the negotiation object, type Manufacturability Markup
    - EngChangeDocument to associate the markup documents
15. Develop the per-workstation (major resource) processing specifications. Retrieve the Part model, Bill of materials, Process plans:
    - PartDocumentRelationship, to get the Part model, and the Manufacturing Specification Package for the target facility
    - Identifiable, ProcessRevision to get the Process
16. For each Workstation, create a ProcessStep
    - ProcessStep named for the target workstation (type)
    - ProcessRevisionDocumentation to associate it to the Manufacturing Specification Package.
17. For those processes to be performed at the Workstation, create tooling requirements. When these requirements cannot be met by standard tooling, create WorkOrders for tooling design.
    - DocumentMaster, DocumentRevision of type Tooling Requirements
    - SecuredFile, DocumentFileRelationship to associate the requirements definition file to the Document object
    - EngChangeRequest to create the request for engineering action, of type Tooling Design
    - EngChangeDocument to associate the requirements definition and relevant Part model and Manufacturing Specification documents
18. For those processes to be performed at the Workstation, create in-process workpiece geometries, operator instruction sheets, and machine control programs

- DocumentMaster, DocumentRevision (as many as needed) of types in-process geometry, operation sheet, control program
- SecuredFile, DocumentFileRelationship to associate the specification file to the corresponding Document object
- ProcessOperationDocumentation to associate the specification Document to the Workstation Package.

19. For each Workstation, create the Tooling list. Tools have exactly the same properties as Parts, and are assumed to be a subtype of Part, for PDM purposes. A Tooling list looks exactly like an Assembly a list of quantities of tool objects which may have versions.
    - PartMaster, PartRevision, to create the Tooling list for Workstation <name> object.
    - This PartRevision represents a Tooling list object that contains Usage information and is subject to revision independent of the real Part. The alternative is to render it into an (opaque) document and lose the PDM where-used relationships.
    - ProcessStepUsesTool to add the Tooling list to the ProcessStep.
    - Usage to put an entry for one Tool in the tooling list (a Tool is a subtype of Part)
20. Identify requirements for consumables and lossage and modify Bill of Materials
    - ProcessStepConsumesPart to put an entry for each Consumable in the ProcessStep.
    - Usage to put one Consumable on the BoM for the Part Usage
21. Validate workcell package test operator instructions, control programs, and tooling in the target facility or a prototyping facility. Retrieve Workstation Package:
    - PartDocumentRelationship, to get the appropriate Manufacturing Specification Package,
    - ProcessStepProducesPartRelationship to get the ProcessOperation
    - ProcessOperationDocumentation to get the operator instructions and control program Documents
    - DocumentFileRelationship to get the associated specification files from the Documents
22. Modify workstation specifications to resolve problems discovered
    - Identifiable, DocumentRevision, SecuredFile, DocumentFileRelationship to modify the appropriate document
    - PartRevision to produce a formal revision of the tooling list, if necessary
    - Usage, to find an entry in the tooling list object (a PartRevision), to delete an existing Tooling list entry, to add a new tooling list entry
23. Produce Validation report for each station and associate to the ProcessStep
    - DocumentRevision, SecuredFile, DocumentFileRelationship, ProcessOperationDocumentation to associate the validation report to the ProcessStep.
24. Sign off the ProcessStep for each station

- PartDocumentRelationship, to get the appropriate Manufacturing Specification Package,
- Stateable to change the state of the Workstation specification package

25. Roll up the Workstation Tooling lists into the overall Bill of Materials
    - PartDocumentRelationship, to get the appropriate Manufacturing Specification Package,
    - PartDocumentRelationship to find the Bill of Materials in the Manufacturing Specification Package.
    - ProcessStepProducesPartRelationship to get the ProcessStep
    - ProcessStepUsesTool to find the Tooling list
    - Usage to copy one Tooling list entry to the Manufacturing BoM for the Part
26. Update overall Resource Requirements time estimates
    - PartDocumentRelationship, to get the appropriate Manufacturing Specification Package,
    - ProcessStepProducesPart to get the ProcessStep
    - From each package, ProcessOperationDocumentation to get the Validation report
    - Identifiable, DocumentRevision, DocumentFileRelationship, to modify the Resource Requirements
27. Develop final Cost Estimate document get preliminary cost documents, updated machine time and materials requirements and tooling costs
    - PartDocumentRelationship, to get the appropriate Manufacturing Specification Package,
    - RevisionRelationship to get the Cost Estimates, Resource Requirements, Tooling Cost Estimates Identifiable, DocumentRevision, DocumentFileRelationship, to modify the Cost Estimates
28. Review and signoff the final Manufacturing Specification Package, as in Steps 12 and 13.

A.7.7 Develop Assembly Process Design

1. The operations and use of the PDM are essentially the same as the 28 steps in the previous task, except that the original Bill of Materials is generated by the designer and consists of the component parts (so Step 5 is omitted). Many of the other activities may be somewhat simplified and in particular, the per-workstation breakdown may or may not occur, depending on the design of the line.

A.7.8 Develop Manufacturing Facility Plan

A.7.9 Develop Production Tooling Design

Similar to Product Design

A.7.10 Procure Machine Tools, Firm Tools and Services

A.8 Coordinate Design Change

- Coordinate Design Change (2b), (2c), (3), (4), (5a), (5b), (6), (7), (8)
- Identify design and process changes
- Process external design changes
- Identify implementation dependencies
- Notify required team members
- notify design change across all products (design control)

The steps of Notify required team members and Notify design change across all products are addressed by the EngChangeAffectedParty relationship.

A.9 Evaluate Product Design

- Evaluate Product Design (2a), (2b), (4), (8)
- Develop Model or Prototype of Proposed Design
- Develop Test Plans and Prototype Configurations
- Run Tests and Report Results
- Analyze Test Results

The emphasis is on CAE evaluation of proposed designs. Product definition and models are stored in data files attached to design definition documents.

A.9.1 Develop Model or Prototype of Proposed Design

1. Access design definition documents.
   - PartDocumentRelationship.
2. Access previously built models of the part or assembly.
   - PartDocumentRelationship, PartRevision, BaselineRevision
3. Access previously built models of components.
   - Usage to find the components.
   - PartDocumentRelationship, PartRevision, BaselineRevision
4. Register a newly created model.
   - DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, DocumentFileRelationship to register the files containing the model and associate them to a document.
   - PartDocumentRelationship to associate the model with the part or assembly that it represents.
   - Derive to associate the model with any other model from which it is derived.

- Dependency to associate the model with any other model that it incorporates, such as a model of a component.

A.9.2 Develop Test Plans and Prototype Configurations

1. Register newly created test plans.
   - DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, DocumentFileRelationship to register the files containing the test plans and associate them to a document.
   - PartDocumentRelationship to associate the model with the part or assembly that it represents.

A.9.3 Run Tests and Report Results

1. Access other test results that provide loading information.
   - Usage to navigate the product structure. (The required analysis may be performed on a higher or lower level assembly.)
   - PartDocumentRelationship.
2. Register the results.
   - DocumentMaster, DocumentRevision, DocumentIteration, SecuredFile, DocumentFileRelationship to register the file containing the specifications and associate them to a document.
   - Derive to associate the results with the model.

A.9.4 Analyze Test Results

1. Access specifications to determine evaluation criteria.
   - PartDocumentRelationship.

A.10 Implement Production Changes

- Implement Production Changes (2c), (3), (4), (5b), (6), (7), (8)
- Plan for future manufacturing
- Plan Product Change implementation
- Generate Material Requirements for prototype
- Alert change and restrict inventory/investment
- Prototype build using Production Facilities
- Distribute Prototype Build Documentation
- Schedule tool tryout
- Establish Change Implementation Schedules (decision)
- Generate Material requirements for pilot

- Notify and distribute update manufacturing documentation
- The step Alert change is addressed by the EngChangeAffectedParty relationship (section 2.9.3.17).

A.11 Develop Product Service Methods

- Develop Product Service Methods (3), (4), (5b), (5c), (6), (7), (8)
- Develop service assembly procedures
- Develop diagnostic procedures

A.12 Develop Service Distribution Plan

- Develop Service Distribution Plan (3), (4), (6), (7)
- Create parts catalogs
- Create service inventory plans.

OMG IDL *B*

Please refer to OMG document formal/2000-10-65. This is a zipped file that contains compilable IDL for the PDM modules. To access this file, point to:

http://cgi.omg.org/cgi-bin/doc?formal/2000-10-65

References C

The following documents provide supporting material or are referenced by this proposal.

Product Data Mangement Enablers Request For Proposal, OMG document mfg/96-08-01.

Richard Soley (ed.), *Object Management Architecture Guide*, Third Edition, Wiley, June 1995.

The Common Object Request Broker: Architecture and Specification, Revision 2.0, July 1995.

CORBAservices: Common Object Services Specification, Revised Edition, March 31, 1995.

CORBAfacilities Architecture, Revision 4.0, November 1995.

Common Business Objects and Business Object Facility RFP, OMG document: cf/96-01-04.

Product Data Interchange using STEP (PDES), US Product Data Association.

Part 21 - Clear Text Encoding of the Physical File Exchange Structure, ISO 10303-21: 1994(E)

Part 41 - Integrated Generic Resources: Fundamentals of Product Description and Support, ISO 10303-41: 1994(E).

Part 44 - Integrated Generic Resources: Product Structure Configuration, ISO 10303-44: 1994(E).

Part 203 - Application Protocol: Configuration Controlled Design, ISO 10303-203: 1994(E).

Part 214 - Application Protocol: Core Data for Automotive Mechanical Design Processes, ISO 10303-214 Committee Draft 2, 1997.

PDM Schema 1.0, PDES Inc, ProSTEP and JSTEP.

Workflow Facility DRAFT Request For Proposal, V.1.0, OMG Document: cf/97-03-14.

The Workflow Reference Model (WFMC-TC-1003, 29-Nov-94, 1.1), http://www.aiai.ed.ac.uk:80/WfMC

The Computer Integrated Manufacturing (CIM) Application Framework Specification 1.3, SEMATECH, January 31, 1996.

Relationship to Other Services D

D.1 Relationships to Other OMG Specifications

D.1.1 CORBAservices

This specification accommodates CORBAservices specifications and avoids specifying competing interfaces. In some cases, it specifies the use of existing CORBAservices interfaces. In other cases, it specifies interfaces whose implementations may use CORBAservices. In other cases, CORBAservices may be used in conjunction with the PDM enabler interfaces to contribute to a total product data management solution.

LifeCycle Service

The LifeCycle Service describes a standard scheme for creating and finding objects using a Factory object, and for moving and copying objects using LifeCycle operations.

The **ManagedEntity** object (and thus all of its subclasses) are **LifeCycleObject**s. They inherit the **CosLifeCycle::LifeCycleObject** interface, which includes the copy, move, and remove operations.

Each PDM enabler module defines Factory interfaces for each of the **LifeCycleObject**s that are introduced by the module.

The **PdmSystem** object of the **PdmFramework** module acts as a **FactoryFinder** to locate an appropriate factory for a copied or moved object.

Relationship Service

Perhaps the most important function of a PDM system is to manage relationships between different objects used to specify product information.

The CosRelationship Service specifies a robust, standard set of interfaces that can be used to create, navigate, and manipulate relationships. The interfaces specified by the Relationship Service provide the primary way to establish and navigate relationships in the PDM enablers.

This specification uses the Relationship Service to define associations between objects, aggregates of objects, and object composition. The **PdmFramework** module subtypes the COS containment and reference relationships to add specialized behavior. The **PdmResponsibility**, **PdmFoundation**, and **PdmFramework** modules are the only ones which directly reference the COS Relationship Service.

Query Service

The CosQuery Service specifies a standard set of interfaces that can be used to query the persistent database for collections of objects that meet specified criteria.

A PDM system that implements the PDM enabler modules, or the underlying database management systems of the PDM system, may support the standard Query interfaces, but such use is not required in order to be compatible with the PDM enabler interface.

The Identifiable interface specifies operations that allow client programs to obtain objects by their key identification attributes in a flexible manner. Many interoperating clients can use these operations to perform their function, without relying on the Query Service.

However, the ability to flexibly query information stored in the PDM system is an essential part of a total PDM solution. Vendors of PDM systems are encouraged to provide access to a Query Service.

Object Property Service

As defined by the CosProperty Service, properties are typed, named values dynamically associated with an object and outside of the type system.

A typical PDM system is highly customizable. Classes and attributes are extended uniquely for each customer site and for each application interface. In addition, classes and attributes are changed frequently during the life of a PDM system. Some PDM systems also support ad-hoc attributes that are not specified in the schema or object model of the PDM system.

This specification uses the Object Property Service to allow client programs to communicate to the PDM system with regard to extended classes and attributes without requiring that the IDL and all clients have knowledge of the entire detailed schema of the PDM system.

The **Attributable** class in the **PdmFramework** module uses a non-persistent **PropertySet** in the signature of the **set_info** and **get_info** operations to pass attribute information between clients and the PDM System.

A **PropertySet** is used in the signature of PDM enabler create and copy operations to initialize attributes of new objects.

Transaction Service

In this specification all persistent objects optionally inherit the **CosTransactions::TransactionalObject** interface. If the implementation has a transaction service then transactional behavior is provided by that service. This specification does not require the presence or use of a transaction service. In implementations that do not support transaction services, the persistent objects will not inherit from CosTransactions. In this case each operation is atomic and must leave the object in a consistent state.

Security Service

The CosSecurity Service specifies a standard set of interfaces that can be used to implement security. The security service can be applied to objects without requiring any security specific constructs in their IDL.

Therefore, a PDM system that implements the PDM enabler modules may transparently support the standard security interfaces, but such use is not required in order to be compatible with the PDM enabler interface.

There is no requirement that the PDM Enabler interfaces implement any security policies, procedures or methods.

If secure interfaces are required in a given implementation environment, then all objects should be security sensitive and all operations should be subject to security policy control. The objects will need to be security aware and to pass on security credentials when issuing requests. Default policies for security sensitive objects are not defined and there are no special security considerations introduced.

From CORBA Services Security:

> An active entity must establish its rights to access objects in the system. It must either be a principle, or a client acting on behalf of a principle. A principle is a human user or system entity that is registered in and authentic to the system.

To meet this security requirement the any object that uses or accesses a PDM Enabler object must be a principle or a client acting on behalf of a principle to form the basis for security delegation considerations and to achieve the access control needed.

Relative to CORBA Services Security Section, Access Control Model, there are two important parts to the Access Control Model, the object invocation access policy and the application access policy.

The object invocation access policy is outside the scope of this specification. However, for the object invocation access policy to work there must be control attributes on the object. One extreme is to allow all access. For most situations some minimum level of granularity of object operations should be supported for interoperability.

The application access policy could be inside the scope of this specification, and no policy is specified or required. The various implementations of this standard should provide whatever application access policy is needed for their customers and market.

Relative to CORBA Services Security, this specification does not list any required application security relevant events. The various implementations of this standard should provide whatever security relevant events are needed for their customers and market.

Relative to CORBA Services Security, Delegation, the PDM Enabler objects will often be intermediate objects in the Security Specification vocabulary. As intermediate objects the PDM Enabler objects should be able to support at least combined privilege delegation and in higher accountability situations the PDM Enabler objects should support traced delegation. This specification does not require PDM Enabler objects to support any delegation.

Relative to CORBA Services Security, Non-repudiation, as long as a PDM Enabler objects reside with in a single security domain, the need for non-repudiation is usually low. When access to PDM enabler objects happen across security domains (which is also across business domains in most cases), then non-repudiation services could be required. The Security Service Specification states that the non-repudiation services specified here are under control of the applications. The need for non-repudiation services is defined by the specific application situation, therefore this specification does not mandate the use of any non-repudiation services.

Relative to CORBA Services Security, Domains, PDM Enabler objects may have several hierarchical levels. For practical security purposes the PDM Enabler objects (at least all objects in the same module) should be within the same security policy domain. This specification does not mandate that all the objects involved within a PDM Enabler implementation exist within the same security policy domain.

Time Service

The specification uses UtcT data type from the time service to represent date and time.

Currency Service

This specification uses the Currency data type.

D.1.2 Common Business Objects

The PDM enabler specification defines certain framework concepts which may in the future be accommodated by a future OMG specification together with common business objects. Many objects are core or common objects in other business domains as well as manufacturing and PDM. These objects are currently specified in the PdmResponsibility and PdmFoundation modules. Many of them may be superseded by adopted common business objects in the future. For example:

- Person
- Organization

The following objects may be used widely in the business, but are core objects of the manufacturing domain and PDM enablers in particular. This specification defines interfaces for these classes, for example:

- Part
- Document
- Engineering Change Order
- Effectivity

D.1.3 Workflow

Workflow services are an important aspect of a fully functional PDM solution. The OMG Workflow specification and the Pdm Enablers are complementary. The PDM Enablers provides the repository for information (even multiple versions) and Workflow provides the Process to dynamically change the information. One possible connection is to tie the PDM Enablers and Workflow together by having an interface inherit from **Wf_requester** and **EngChangeRequest**. The engineering change process would use **EngChangeIssue** as input and produce at various steps in the process **EngChangeOrder**, **EcoDeliverable**, **EngChangeNotice**, and part or document revisions.

A good way to view the interaction of Workflow and PDM Enablers to carry out an Engineering Change is the following:

- Consider the Workflow as an Actor in the PDM Enabler sense. The Workflow then takes responsibility for a set of ECIs, Parts and Documents. At various stages of the Change Process and ECR could be created, ECOs could be created, Items or revision or new parts or documents could be created. The PDM Enablers provides the information organization and control aspects, while the Workflow encapsulates the ordering of the steps needed to change the information. The key is that Workflows can work with objects that support **ManagedEntity** interface.

When activities involve changes to a **ManagedEntity** in the PDM system, the changes are accomplished by the workflow management system via its Invoked Application Interface, which uses the PDM enabler CORBA interfaces to accomplish the change. Such activities might be:

- Updating attributes.
- Promoting the item to its next state.
- Transferring the item to a different PdmSystem.
- Establishing or changing relationships.

When the activities involve operations on a File, they are typically accomplished by the workflow management system via its Invoked Application Interface, which typically uses an interface to the data native application program. Such an application interface can be invoked either directly, through the facilities of the application, or indirectly, through facilities of the PDM system. The specification of such application interfaces is outside of the scope of the PDM enablers specification. Such activities might be:

- Reviewing the File.
- Changing the File.

- Translating, analyzing, moving, copying, or printing the File.

The **PdmFoundation** module, **Stateable** interface carries the concept of state and operations to change state.

The PDM enablers do not provide support for approval or signoff. Approval and signoff are not meaningful except in the context of a business process or workflow definition that defines for what purpose or stage the item is approved or signed off. Historical tracking information relating the persons, stage, and disposition of a item in an approval process is the responsibility of a workflow service, which can operate in a manner that includes history for other actions in addition to approval.

D.1.4 Relationships to Other Standards

RM-ODP

Although this specification was not developed under a RM-ODP methodology or orientation, its structure does map to RM-ODP concepts.

Enterprise Viewpoint

The Manufacturing DTFs Manufacturing Enterprise Systems White Paper mfg/96-01-02 outlines the scope of the Enterprise Viewpoint of the whole manufacturing domain.

The PDM enablers RFP mfg/96-08-01 itself is also an expression of the Enterprise Viewpoint, especially section 6.5 Manufacturing Domain, in which the business processes of a manufacturing enterprise are outlined at a high level. Section 6.5.2 of the RFP outlines the sub-processes of product development which the PDM enablers are designed to address.

This information is expanded and mapped to the PDM enablers in the previous section, Mapping the Product Development Process to the PDM Enablers.

Information Viewpoint

In the PDM Enablers context, the information viewpoint is specified and held privately by the vendors of the PDM systems. In some sense, ISO 10303 (STEP), especially the STEP PDM Schema defines an information viewpoint specification of a static model of a vendor neutral schema which is appropriate for data interchange.

Computational Viewpoint

This Specification presents a specification in the Computational Viewpoint. APIs for interoperability are specified in IDL and are described in UML diagrams and natural language descriptions.

Engineering Viewpoint and Technology Viewpoint

The PDM enablers specification does not express the Engineering and Technology viewpoints. They are the province of CORBA, the ORB vendors, and the PDM vendors.

ISO 10303 (STEP)

ISO 10303, Standard for the Exchange of Product Model Data (STEP) and the PDM Enablers are different in purpose, scope, abstraction level, and operational characteristics. Despite these differences, and because of these differences, each provides an essential piece in the process of defining and managing product data.

The OMG and STEP communities have co-authored a comprehensive whitepaper outlining the relationships and synergies between these two important PDM standards: “STEP and OMG Product Data Management Specifications: A Guide for Decision Makers” (available at http://www.omg.org/cgi-bin/doc?mfg/1999-10-04).

D.2 Compliance

D.2.1 Summary of Optional vs. Mandatory Interfaces

All interfaces defined in each module are mandatory, unless indicated as optional in this section.

Modules

A PDM system implementation of these PDM enabler modules is not required to provide all modules. In a sense, each module as a whole is optional. However, some modules depend on other modules.

Qualifications

The Qualification interface defined in the **PdmViews** module is mandatory. However, the inheritance of the Qualifiable interface by each of the relationships and interfaces it affects is optional. If a Qualifiable inheritance is implemented, the implementation is not required to implement all subtypes of Qualification.

If they are implemented, they must be implemented as described.

For example, a **PartMaster** implementation might support **DisciplineQualification** but not support **EffectivityQualification**.

Transactions

In this specification all persistent objects optionally inherit the **CosTransactions::TransactionalObject** interface.

Documentations

The Documentation relationship, defined in the **PdmDocumentManagement** module, between a **Documentable** object and a **DocumentMaster** object is optional. If a Documentation relationship is implemented, the implementation is not required to implement this relationship for all **ManagedEntity** objects. For example, a **PartMaster** object implementation might support Documentation relationship but a File object may not support this relationship.

PdmSTEP

The **StepTranslator::export_baseline** operation is optional.

CORBAservices

This specification specifies interfaces inherited from CORBAservices specifications. Compliant PDM enablers implementations shall provide all operations and attributes inherited by PDM enabler interfaces.

It is not required that a PDM Enablers implementation provide the services of CORBAservices interfaces which are not inherited by PDM Enablers interfaces.

Index

A
AssemblyComponentUsage 8-5
Attributable 4-4

B
Baselineable 4-6, 5-2
Baselined 5-5
BaselineIteration 5-4
BaselineMaster 5-3
BaselineMasterComposition 5-4
BaselineRevision 5-3
BaselineRevisionComposition 5-4

C
Changeable 4-6
Compliance 1-3, 11
ComponentSolution 12-13
CompoundContext 6-7
CompoundQualification 6-7
Configuration 12-25
ConfiguredItemUsage 12-25
ConfiguredSpecificationSolution 12-26
ControlledProcessStep 11-11
CORBA
  contributors 3
  documentation set 2

D
Data Transfer 7-12
Dependency 4-13
Derive 4-12
DerivedSolution 12-13
DesignSupplierRelationship 8-6
DisciplineContext 6-7
DisciplineQualification 6-6
Document Management IDL 7-14
Documentable 4-6, 7-5
Documentation Relationship 7-6
DocumentFileRelationship Relationship 7-7
DocumentIteration 7-4
DocumentMaster 7-3
DocumentMasterComposition Relationship 7-4
DocumentRevision 7-3
DocumentRevisionComposition Relationship 7-5

F
Factory Operations 3-15
Factory operations 3-10
File 7-6
FileStorage Relationship 7-12

I
Identifiable 3-7
Identification 3-16
IdentificationContext 3-8
IdentifierSeq 3-7
ImportedProcessStep 11-11
ImportedProcessStepUsedAtOrganization Relationship 11-11
IncludedSpecification 12-22
InclusionIfCondition 12-22
ItemIteration 4-7
ItemMaster 4-6
ItemRevision 4-7
ItemSolution 12-12
IterationRelationship 4-11

L
LifeCycleContext 6-6
LifeCycleQualification 6-5
LocationContext 6-6
LocationQualification 6-6
Lockable 3-13
LockOwner 3-17

M
Make From Usage 8-16
Manageable 3-14
ManagedEntity 4-6
ManufacturingOrganization 11-12
MasterRelationship 4-10

N
NextAssemblyUsageOccurrence 8-7

O
Object Management Group 1
  address of 2
ObjectCreator 3-18
ObjectOwner 3-18
ObjectSecurityClassification 3-19

P
PartData 8-8
PartDataIteration 8-8
PartDataIterationRelationship 8-8
PartDataRelationship 8-9
PartDocumentRelationship 8-6
PartMaster 8-9
PartMasterComposition 8-10
PartProducedByProcessRevision Relationship 11-6
PartRevision 8-11
PartStructure 8-11
PartStructureIteration 8-11
PartStructureIterationRelationship 8-12
PartStructureRelationship 8-12
PartSupplierRelationship 8-13
PDM Enablers 1-1
PdmBaseline IDL 5-6
PdmBaseline Model 5-2
PdmChangeManagement
  EngChangeRequest 10-6
PdmChangeManagement Description
  Addressing 10-7
  ChangeDescription 10-7
  Deliverable 10-8
  EciInitiation 10-8
  EcnCoRequirement 10-9
  EcnPreRequirement 10-10
  EcoCoRequirement 10-10
  EcoDeliverable 10-4
  EcoPreRequirement 10-11
  EcrInitiation 10-11
  EnccoRequirement 10-9
  EngChangeAffecteData 10-12

Index

EngChangeAffectedParty 10-13
EngChangeIssue 10-5
EngChangeItem 10-3
EngChangeNotice 10-5
EngChangeOrder 10-5
ObjectChange 10-14
ObjectChangeNotification 10-14
Responsibility 10-15
PdmChangeManagement IDL 10-15
PdmChangeManagement Model 10-3
PdmConfigurationManagement Description
ComponenetHierarchy 12-8
ComponentFunction 12-10
FunctionHierarchy 12-10
PartDiscipline 12-5
ProductClass 12-5
ProductClassHierarchy 12-6
ProductClassToComponent 12-7
ProductComponent 12-6
ProductFunction 12-9
PdmConfigurationManagement IDL 12-28
PdmConfigurationManagement Model 12-3
PdmContainmentRelationship 4-8
PdmContext 6-3
PdmDocumentManagement Model 7-2
PdmEffectivity Description
ConfigurationDesign Relationship 9-3
ConfigurationItem 9-2
DatedContext 9-6
DatedEffectivity 9-4
Effectivity 9-3
Effectivity Context 9-6
EffectivityItem Relationship 9-4
LotContext 9-7
LotEffectivity 9-5
SerialNumberedEffectivity 9-6
PdmEffectivity IDL 9-8
PdmEffectivity Model 9-2
PdmEffectivyt Description
SerialNumberedContext 9-7
PdmFoundation IDL 3-20
PdmFoundation Model 3-2
PdmFramework Description 4-4
PdmFramework Entity Model 4-2
PdmFramework IDL 4-15
PdmFramework Relationship Model 4-3
PdmManufacturingImplementation Description
ProcessMaster 11-2
PdmManufacturingImplementation IDL 11-12
PdmManufacturingImplementation Model 11-2
PdmProductStructureDefinition Description 8-4
PdmProductStructureDefinition IDL 8-17
PdmProductStructureDefinition Model 8-3
PdmReferenceRelationship 4-9
PdmResponsibility IDL 2-8
PdmResponsibility Model 2-2
PdmStep Description
StepContext 13-4
StepQualification 13-4
StepTranslator 13-2
PdmStep IDL 13-5
PdmStep Model 13-2
PdmSystem 4-4
PdmTraversalCriteriaFactory 6-4
PdmTypedRelationship 4-9
PdmViews IDL 6-9
PdmViews Model 6-2
ProcessMasterComposition Relationship 11-3
ProcessMasterControlledByOrganization Relationship 11-3
ProcessMasterUsedAtOrganization Relationship 11-4
ProcessOperation 11-7
ProcessOperationDocumentation Relationship 11-7
ProcessOperationSequence Relationship 11-8
ProcessRevision 11-5
ProcessRevisionComposition Relationship 11-5
ProcessRevisionDocumentation Relationship 11-5
ProcessStep 11-9
ProcessStepConsumesPart 11-10
ProcessStepUsesTool 11-9
ProductClassCategory 12-18
ProductClassExpression 12-19
ProductClassFunction 12-11
ProductClassInclusion 12-20
ProductClassSpecification 12-21
ProductComponentSatisfiesSpecification 12-27
ProgramOwner 2-7
PromissoryUsageOccurrence 8-13

Q
Qualifiable 4-6, 6-3
Qualification 6-2
Qualifies Relationship 6-3

R
RevisionMasterRelationship 4-14
RevisionRelationship 4-11

S
SecuredFile 7-8
SecurityClassifiable 3-14
SecurityClassification 3-14
SecurityClassification Attributes 3-15
SolutionPartMaster 12-14
Specification 12-16
Specification Characteristics 1-2
Specification Overview 1-2
Specification Structure 1-3
SpecificationCategory 12-15
SpecificationCategoryComposition 12-23
SpecificationExpression 12-16
SpecificationExpressionOperands 12-24
SpecificationInclusion 12-17
SpecificationOperation 12-15
Stateable 3-15
StateContext 6-8
Substitute 8-14
Supersedes 4-14

U
UnsecuredFile 7-8
Usage 8-15
Using Qualifications and PdmContexts 6-8

V
Vaults 7-11
ViewContext 6-5
ViewQualification 6-5

Index