MEF Standard

MEF 78

MEF Core Model (MCM)

January 2019

Disclaimer

The information in this publication is freely available for reproduction and use by any recipient and is believed to be accurate as of its publication date. Such information is subject to change without notice and MEF Forum (MEF) is not responsible for any errors. MEF does not assume responsibility to update or correct any information in this publication. No representation or warranty, expressed or implied, is made by MEF concerning the completeness, accuracy, or applicability of any information contained herein and no liability of any kind shall be assumed by MEF as a result of reliance upon such information.

The information contained herein is intended to be used without modification by the recipient or user of this document. MEF is not responsible or liable for any modifications to this document made by any other party.

The receipt or any use of this document or its contents does not in any way create, by implication or otherwise:

a) any express or implied license or right to or under any patent, copyright, trademark or trade secret rights held or claimed by any MEF member which are or may be associated with the ideas, techniques, concepts or expressions contained herein; nor

b) any warranty or representation that any MEF members will announce any product(s) and/or service(s) related thereto, or if such announcements are made, that such announced product(s) and/or service(s) embody any or all of the ideas, technologies, or concepts contained herein; nor

c) any form of relationship between any MEF member and the recipient or user of this document.

Implementation or use of specific MEF standards, specifications, or recommendations will be voluntary, and no Member shall be obliged to implement them by virtue of participation in MEF Forum. MEF is a non-profit international organization to enable the development and worldwide adoption of agile, assured and orchestrated network services. MEF does not, expressly or otherwise, endorse or promote any specific products or services.

Table of Contents

List of Figures

List of Tables

1 List of Contributing Members

The following members of the MEF participated in the development of this document and have requested to be included in this list.

Amdocs	Huawei Technologies
Coriant (as Infinera)	PCCW Global Limited
Ericsson AB	Verizon

2 Abstract

This specification defines the MEF Core Information Model (MCM), which is an information model describing the base set of object definitions and relationships supporting the concepts defined in the MEF Lifecycle Service Orchestration (LSO) Reference Architecture (RA). The MCM formalizes these diverse concepts into a coherent, object-oriented information model that can serve the needs of multiple MEF projects by defining key concepts and functions that can be reused or refined as necessary.

This specification uses UML (Unified Modeling Language) to describe the salient characteristics and behavior of entities that are important to the managed environment. This does not mean that the MCM will try and model "everything"; rather, it means that it will represent key entities that various MEF projects need. For example, the Sonata Ordering project needs the concept of an Order. MCM provides a basic set of model elements to represent this (see section 7.7) as a reusable pattern, so that other similar concepts (e.g., TroubleTicket) can use the same pattern (adjusted as necessary to suit the differences between TroubleTicket and Order). As another example, ONF TAPI is used to model lower-level resources in NRM and NRP. A higher-level representation of resources is required in order to join this lower-level model to other entities (e.g., Products and Offers). The MCM provides the basis for this higher-level representation.

These entities, and the relationships between them, describe concepts used by different functional components (e.g., the Service Orchestration Functionality (SOF) and Infrastructure Control and Manager (ICM), as well as different actors (e.g., business applications, as well as Customers, Application Developers, and Administrators) that are designing, implementing, and deploying LSO functionality. The model elements (e.g., classes, attributes, relationships, and operations) defined in this model are not specific to Carrier Ethernet, and are intended to define a comprehensive abstract model from which more specific models can be extended.

The MCM is built on modeling best practices (e.g., [5][6][8]), and uses a number of software patterns (e.g., [2][3][4]) to provide an extensible framework that can support model-driven engineering [9] as well as the needs of DevOps-inspired automation. It defines concepts and functions that can be represented to define data exchanged at all seven of the Interface Reference Points defined in [1].

Put another way, the MCM serves as a common lexicon for all MEF models. It defines a set of concepts and terms, and relationships between them, in an object-oriented information model. This makes it independent of any specific architectural paradigm (e.g., resource- or service-oriented architectures).

As MEF models evolve, and define new concepts, those concepts will be added to the MCM if they can be used by multiple teams.

This document normatively includes the content of the following Papyrus UML files as if they were contained within this document from the MEF GitHub Repository (https://github.com/MEF-GIT/MEF-Common-Model): MCM.di, MCM.notation, and MCM.uml.

3 Terminology and Abbreviations

This section defines the terms used in this document. In many cases, the normative definitions to terms are found in other documents. In these cases, the third column is used to provide the reference that is controlling, in other MEF or external documents.

Term	Definition	Reference
Abstract Class	An abstract class is a class that cannot be directly instantiated. It can have abstract or concrete subclasses.	THIS DOCUMENT
Abstraction	Abstraction is the process of focusing on the important characteristics and behavior of a concept, and ignoring less important characteristics and behavior.	THIS DOCUMENT
Class	A class is a template for defining a specific type of object that exhibits a common set of characteristics and behavior.	THIS DOCUMENT
Classification Theory	The principles that govern the organization of objects into groups according to their similarities and differences or their relation to a set of criteria.	THIS DOCUMENT
Concrete Class	A concrete class is a class that can be directly instantiated. Once a class has been defined as concrete in the hierarchy, all of its subclasses are required to be concrete.	THIS DOCUMENT
Customer	A Customer is the organization purchasing, managing, and/or using Connectivity Services from a Service Provider. This may be an end user business organization, mobile operator, or a partner network operator.	[14]
Data Model	A data model is a representation of concepts of interest to an environment in a form that is dependent on data repository, data definition language, query language, implementation language, and/or protocol (typically, but not necessarily, all five).	THIS DOCUMENT
Information Model	An information model is a representation of concepts of interest to an environment in a form that is independent of data repository, data definition language, query language, implementation language, and protocol.	THIS DOCUMENT

Term	Definition	Reference
LSO (Lifecycle Service Orchestration)	Open and interoperable automation of management operations over the entire lifecycle of Layer 2 and Layer 3 Connectivity Services. This includes fulfillment, control, performance, assurance, usage, security, analytics and policy capabilities, over all the network domains that require coordinated management and control in order to deliver the service.	MEF 55 [1]
LSO RA (LSO Reference Architecture)	A layered abstraction architecture that characterizes the management and control domains and entities, and the interfaces among them, to enable cooperative orchestration of Connectivity Services. Note that in this document, cooperative orchestration is NOT limited to only Connectivity Services, and may include other services as well.	MEF 55 [1]
Metadata	Metadata is a class that contains prescriptive and/or descriptive information about the object(s) to which it is attached. While metadata can be attached to any information model element, this document only considers metadata object instances attached to class instances and relationships.	THIS DOCUMENT
Model Element	An element of a model. For the purposes of this document, this refers to a set of classes, attributes, operations, constraints, and/or relationships.	THIS DOCUMENT
Object	An instance of a (concrete) class.	THIS DOCUMENT
Pattern	A pattern describes a named, generic, reusable solution to a problem that applies to a particular context. A pattern is not a finished design, but rather, is a reusable template that defines a set of objects, and their interactions, that can be adapted to meet the context-specific needs required to solve a problem.	[2] [12]
Relationship	For the purposes of this document, a relationship can be any type of association, aggregation, or composition.	THIS DOCUMENT

Term	Definition	Reference
Role	The Role-Object pattern enables an object to adapt to the needs of different applications and contexts by transparently attaching and/or removing Role Objects. Each Role Object defines a set of responsibilities that the object has to play in that client's context. Each context may be its own application, which therefore gets decoupled from other applications. The Role-Object pattern is implemented in the MCM by aggregating Role objects, which are defined as a type of Metadata, to other objects (to enforce the separation of defining an object vs. defining responsibilities that the object has to play).	[3]
Service Provider	The organization providing Ethernet Service(s). Note that in this document, as well as in [1], the (Service Provider) organization is NOT limited to providing only Ethernet Services.	MEF 10.3 [13]
Unified Modeling Language (UML)	The objective of UML is to provide system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes.	OMG UML 2.5 [11]
Whole-Part Relationship	A whole-part relationship is one in which one set of entities aggregates another set of entities. In such a relationship, three objects are created (the entity doing the aggregation, the set aggregated entities, and the combination of the aggregating entity and its aggregated entities). More formally, a whole-part relationship is a *partial ordering* that is reflexive, transitive, and anti-symmetric (i.e., everything is a part of itself, any part of any part of an entity is itself a part of that entity, and two distinct entities cannot be part of each other).	Various; see for example Stanford Encyclopedia of Philosophy

Table 1. Terminology and Abbreviations

4 Compliance Levels

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 (RFC 2119 [7], RFC 8174 [18]) when, and only when, they appear in all capitals, as shown here. All key words must be in bold text.

Items that are **REQUIRED** (contain the words **MUST** or **MUST NOT**) are labeled as **[Rx]** for required. Items that are **RECOMMENDED** (contain the words **SHOULD** or **SHOULD NOT**) are labeled as **[Dx]** for desirable. Items that are **OPTIONAL** (contain the words **MAY** or **OPTIONAL**) are labeled as **[Ox]** for optional**.**

5 Numerical Prefix Conventions

This document uses the prefix notation to indicate multiplier values as shown in Table 2.

Decimal		**Binary**	
Symbol	**Value**	**Symbol**	**Value**
k	10^3	Ki	2^{10}
M	10^6	Mi	2^{20}
G	10^9	Gi	2^{30}
T	10^{12}	Ti	2^{40}
P	10^{15}	Pi	2^{50}
E	10^{18}	Ei	2^{60}
Z	10^{21}	Zi	2^{70}
Y	10^{24}	Yi	2^{80}

Table 2. Numerical Prefix Conventions

6 Introduction

The Lifecycle Service Orchestration Reference Architecture (LSO RA) [1] describes the control and management domains, and the main functional management entities contained in those domains, that enable cooperative LSO capabilities. The architecture also defines the Interface Reference Points (IRPs), which are the logical points of interaction between specific functional management entities. These IRPs are specified in part by Interface Profiles and implemented by APIs. The High Level LSO Reference Architecture is shown in Figure 1. This is a functional architecture, and hence, does not describe how the functional management entities are implemented (e.g., single vs. multiple instances). Rather, it identifies functional management entities that provide logical functionality as well as the points of interaction among them.

This specification uses UML (Unified Modeling Language) to describe the salient characteristics and behavior of entities that are important to the managed environment. These entities, and the relationships between them, describe concepts used by different functional components, such as the Service Orchestration Functionality (SOF) and the Infrastructure and Control Manager (ICM), as well as different actors (e.g., business applications, as well as Customers, Application Developers, and Administrators) that are designing, implementing, and deploying LSO functionality. Figure 1 shows three different domains (Service Provider, Partner, and Customer).

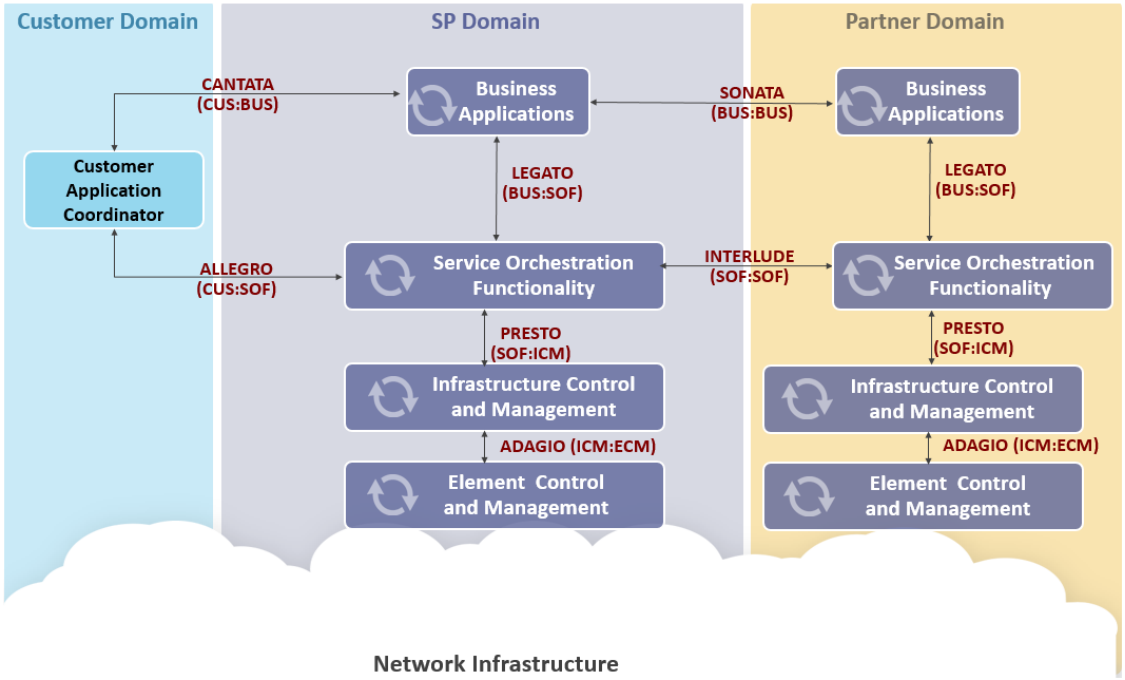

Figure 1. The Lifecycle Service Orchestration Reference Architecture

The scope of the MCM is to model concepts and functions as seen from the Service Provider's point-of-view. This includes interactions between the Service Provider and its Partners, as well as interactions between the Service Provider and its Customers. Hence, the MCM is potentially relevant for all seven IRPs defined in MEF 55.

This document is intended for developers and users that need the formalism that an information model provides. An information represents concepts, along with their relationships and semantics, to help specify an extensible and structured, shareable, information repository.

The remainder of this document defines the MCM. First, a high-level Overview of the MCM is provided in section 7.1. This section also includes brief, informative text to enable the reader to understand important design decisions that were taken in the development of the MCM. Then, section 7.2 defines the top of the MCM class hierarchy (including how the three main hierarchies of the MCM interact with each other), while sections 7.3 - 7.9 define the rest of the MCMEntity class hierarchy. Finally, sections 7.10 and 7.11 define the MCMInformationResource and MCMMetaData class hierarchies, respectively.

7 MEF Core Model (MCM)

The MCM is a UML object-oriented information model that represents key functions and concepts in the Service Provider and Partner Domains of MEF 55 [1]. As such, it also includes key concepts and functions from other domains that are manipulated by the Service Provider and Partner domains (e.g., "Customer").

7.1 Overview of the MCM

The design of the MCM is explained by summarizing the purpose and semantics of the top-level classes of the MCM. This results in three sub-hierarchies, one for each subclass of the MCMRootEntity class, which is the top of the model. Subsequent subsections will then describe each sub-hierarchy in more detail.

The MCM uses the following rules to define the names of its model elements:

- Naming rules are as follows:

  [R1] Class names **MUST** be in UpperCamelCase (i.e., the first letter is capitalized). Class names **MUST NOT** begin with any non-alphabetic character, and no spaces are allowed.

  [R2] Attribute names **MUST** be in lowerCamelCase (i.e., the first letter is lower case); attribute names **MUST NOT** begin with any non-alphabetic character except for the underscore, and no spaces are allowed. Note that attribute names that begin with an underscore are private attributes that reference an end of an association.

  [R3] Relationship names **MUST** be in UpperCamelCase (i.e., the first letter is capitalized). Relationship names **MUST NOT** begin with any non-alphabetic character, and no spaces are allowed.

  [R4] Each class **MUST** be prefixed with "MCM". For example, RootEntity is named "MCMRootEntity". This serves two purposes. First, it helps provide context to textual descriptions of these model elements. Second, it enables MCM model elements, patterns, and approaches to be compared to those of other SDOs and consortia unambiguously.

  [R5] Each attribute **MUST** be prefixed with "mcm". For example, the attribute "commonName" in the MCMRootEntity class is named "mcmCommonName". If an attribute starts with an underscore, then "mcm" immediately follows the underscore (e.g., _mcmARef).

  [R6] Each relationship **MUST** be prefixed with "MCM". For example, the aggregation "EntityHasMetaData" is named "MCMEntityHasMCMMetaData".

[R7] All association classes **MUST** be suffixed with the word "Detail". For example, the association class for the above example is named "MCMEntityHasMCMMetaDataDetail". This makes it obvious that a class is an association class.

- Regarding interoperability with concepts from other SDOs:

[R8] All classes that model a concept from another SDO and *change* the model of that SDO (e.g., to be able to be used in the MCM) **MUST** be prefixed with "MCMMEF". For example, the concept of a Descriptor from ETSI NFV is named "MCMMEFDescriptor".

[R9] All classes that model a concept from another SDO *exactly as it is defined* in that SDO **MUST** be prefixed with "MCM", followed by the name of the SDO, followed by the class name. For example, if an SDO named Foo defined a class named Bar, and MCM imported this concept with no changes, it would be named MCMFooBar.

A note about associations, aggregations, compositions, and their multiplicity. The UML guidelines do not specify in detail what valid multiplicities are. In the MCM, multiplicities are important, in order to provide a robust foundation for code generation, as well as to accommodate the future incorporation of ontologies Therefore:

[O1] Association relationships **MAY** have a 0..* - 0..* multiplicity. This is because they represent a generic dependency, and one end of the association may not be instantiated yet.

[D1] Aggregation and composition relationships **SHOULD NOT** have a 0..* - 0…*

[D2] multiplicity. This is because both aggregations and compositions are a type of whole-part relationship. Ontologically, it is impossible to talk about a "whole" when no "parts" exist (or vice-versa). If there is the possibility of not instantiating the relationship, then the cardinality of the aggregate (or composite) part **SHOULD** be 0..1, where the 0 signifies that the relationship has not yet been instantiated.

[D3] Relationships whose owner (i.e., the source of the relationship) is a value greater than 0 (e.g., 1 or 1..* or 3..7) **SHOULD** have a part multiplicity of at least 1. This is because one side of the relationship **MUST** exist, and it makes no sense to have one side of a relationship exist while the other side doesn't.

7.1.1 The Top Portion of the MCM

Figure 2 shows the top of the MCM class hierarchy (MCMRootEntity), the first level of inheritance (consisting of three subclasses), and relationships with their association classes.

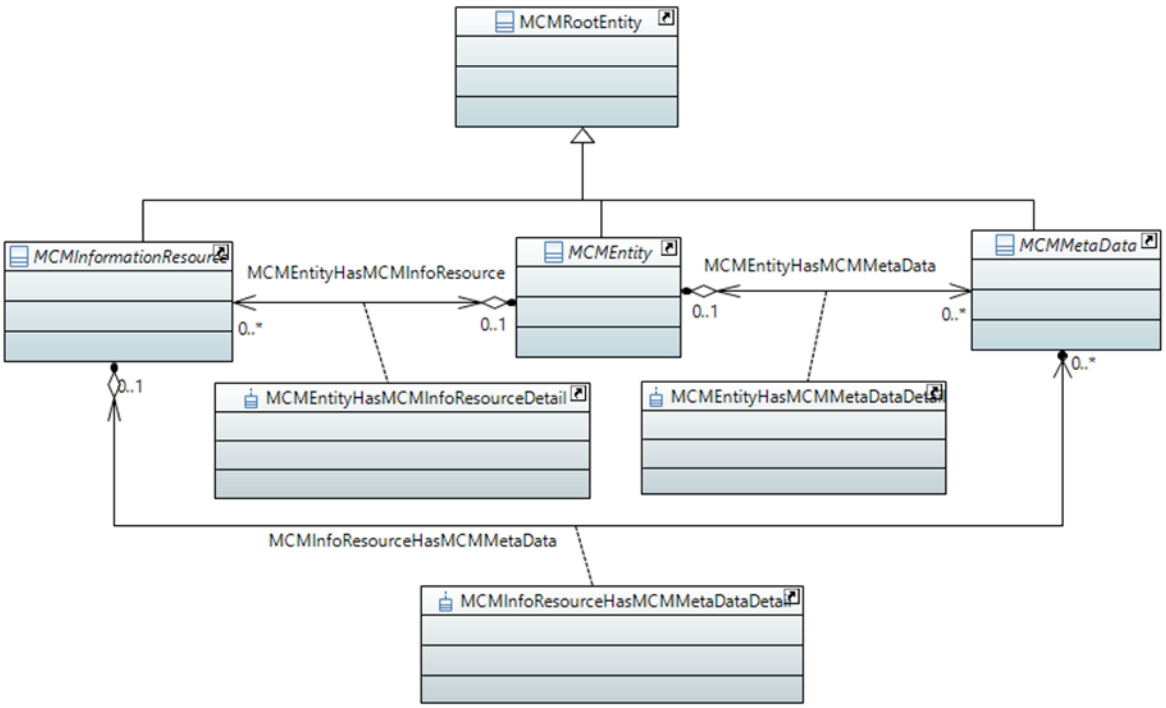

Figure 2. The Top Portion of the MCM Class Hierarchy

MCMRootEntity defines the top of the MCM class hierarchy. Its characteristics and behavior are thus inherited by all MCM classes. MCMRootEntity defines a set of attributes that enable all objects to be unambiguously named, described, and identified in a managed environment. Note that multiple inheritance is *disallowed* in MEF models. Its full definition is defined in Section 7.2.

Figure 2 shows the three subclasses of MCMRootEntity: MCMEntity (see section 7.3), MCMInformationResource (see section 7.10), and MCMMetaData (see section 7.11). The limit of three subclasses simplifies the understanding of the model, and uses classification theory to ensure that objects are organized into groups according to a set of criteria (e.g., their similarities and/or differences).

The three subclasses create three parallel class hierarchies that can interact with each other. For example, object instances from the MCMMetaData class hierarchy are designed to be attached to object instances from the other two class hierarchies. In addition, classes from the MCMInfoResource class hierarchy are inherently related to classes from the MCMEntity class hierarchy.

The three class hierarchies are described as follows:

1) **MCMEntity**, which is the superclass for objects of interest that are important to the managed environment, and which have a separate and distinct existence. These objects can play one or more business functions, and can be managed or unmanaged (using digital mechanisms). Examples include Chassis (unmanaged) and Product, Service, and Resource (all three are managed).

2) **MCMInformationResource**, which is information that is required to describe concepts owned by other Entities, but which is not an inherent part of the Entity being described. For example, an IPAddress is an important piece of data, but it does not control its own lifecycle; rather, its lifecycle is controlled by another Resource (e.g., a DHCPServer). The use of MCMInformationResource enables the IPAddress (in this example) to be represented and associated with the correct Resource responsible for its lifecycle.

3) **MCMMetaData**, which is an object that defines descriptive and/or prescriptive information about the MCMEntity or MCMInformationResource objects that it is attached to. Examples include versioning information of an object, as well as best common practice information and context-specific usage guidelines.

Figure 2 also shows three aggregations, called MCMEntityHasMCMInfoResource (see section 7.4), MCMEntityHasMCMMetaData (see section 7.4.1), and MCMInfoResourceHasMCMMetaData (see section 7.10.1).

The first aggregation defines the set of MCMInformationResource objects that are associated with a given set of MCMEntities. The second and third aggregations define the set of MCMMetaData objects that can be attached to a particular MCMEntity and a given MCMInformationResource, respectively. All three of these aggregations are implemented as association classes; this enables the Policy Pattern (see Figure 3) to be used to define policy rules that constrain which part objects (i.e., MCMInformationResource for the first aggregation, and MCMMetaData for the second and third) are attached to which MCMEntity (first or second aggregation) or MCMInformationResource (third aggregation). An example of the Policy Pattern is shown in Figure 3. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

All MCM association classes are rooted from a single superclass, called MCMRelationshipParent (which in turn is subclasses from MCMEntity); this simplifies both the design of the association classes and their implementation. The MCMPolicyStructure, which is a subclass of MCMPolicyObject (see section 7.8.3), is the superclass of all policies defined in the MEF Policy Driven Orchestration project (i.e., imperative, declarative, and intent policies). The above diagram shows that an object instance of the appropriate concrete subclass of MCMPolicyStructure is related to class-level attributes and operations of an object instance of the MCMEntityHasMCMMetaDataDetail association class.

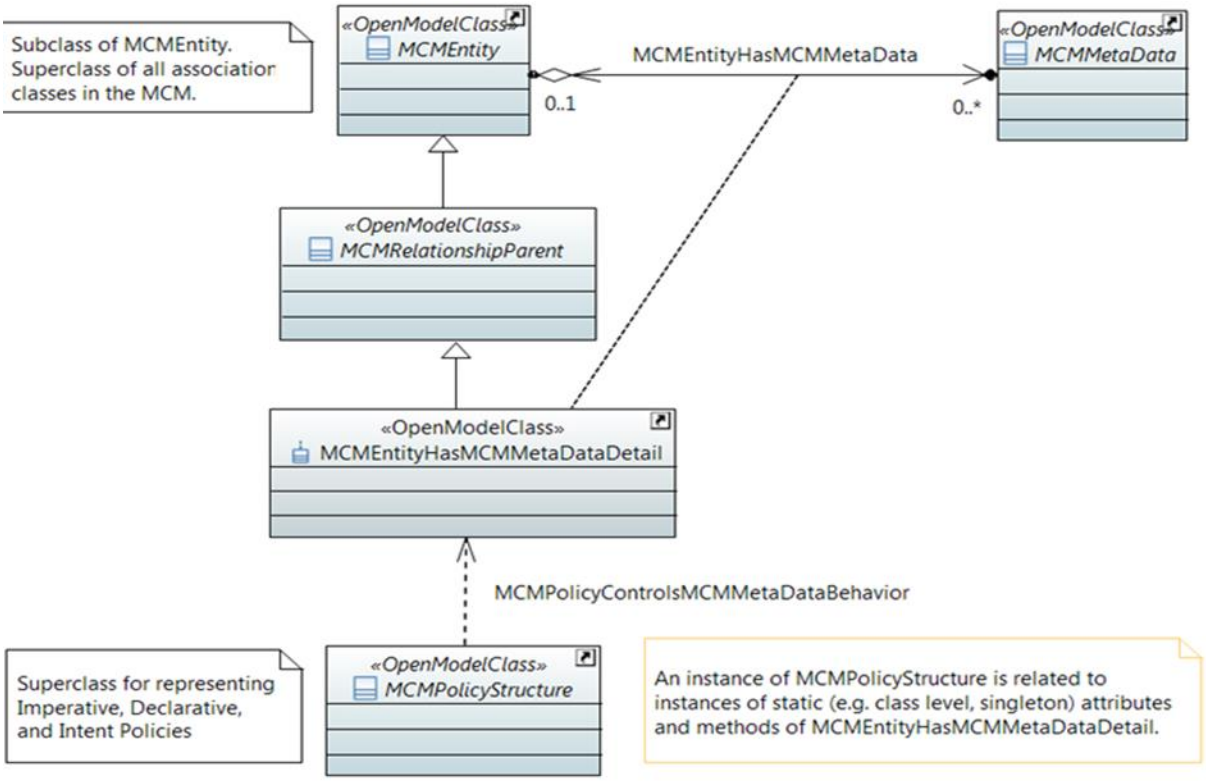

Figure 3. The Policy Pattern Applied to MCMEntityHasMCMMetaDataDetail

7.1.2 MCMEntity Hierarchy

The purpose of the MCMEntity is to represent the characteristics and behavior of concepts that are important to the managed environment. An MCMEntity defines a key concept in the managed environment, and has a separate and distinct existence (i.e., an MCMEntity is not just a collection of attributes or an abstraction of behavior).

The MCMEntity hierarchy is the set of subclasses of the MCMEntity class that define the externally visible characteristics and behavior of the system in more detail. The MCMEntity class is defined in Section 7.4. The main classes in this hierarchy include MCMUnManagedEntity, MCMDomain, MCMBusinessObject, MCMManagedEntity, and MCMParty. See Sections 7.5.1, 7.6, 7.7, 7.8, and 7.9, respectively, for more information.

7.1.3 MCMInformationResource Hierarchy

The purpose of the MCMInformationResource hierarchy is to represent information and concepts needed by one or more managed entities that are not inherent to those managed entities. It is shown in Figure 4.

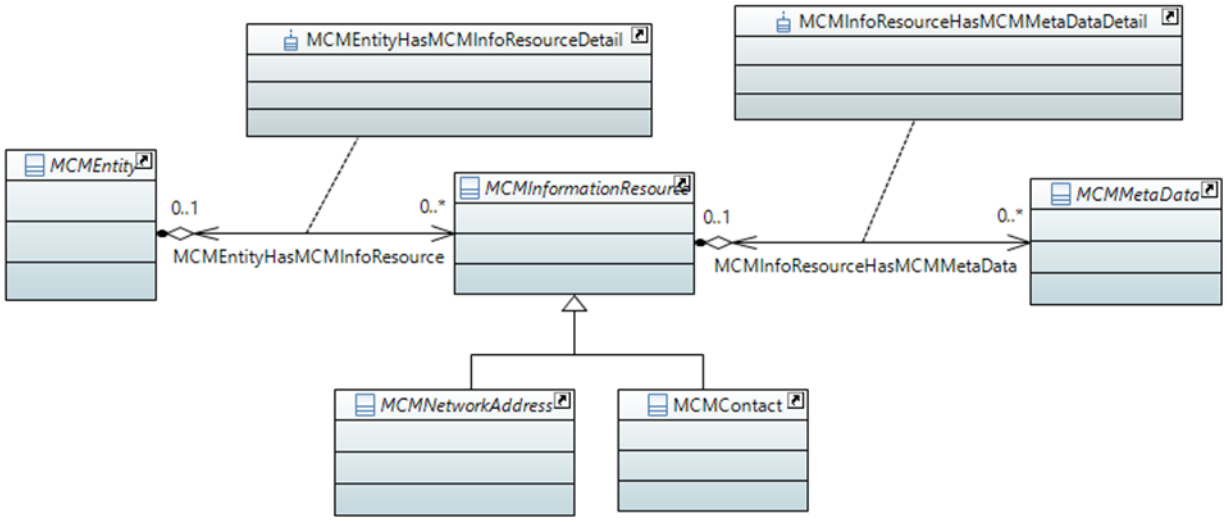

Figure 4. The Top Portion of the MCMInformationResource Hierarchy

Consider the concepts of a networking device and an IP Address. The networking device may be modeled in different abstraction levels, ranging from a black box to a detailed model that shows its constituent manageable components. In either case, an IP Address may be assigned to the networking device (or a component of the networking device). While the IP Address represents important information that is managed, the IP Address is *not* an inherent part of the networking device. IP Addresses are generated by a different component in the system being managed, and then assigned to the networking device.

The MCMInformationResource hierarchy defines concepts owned by a set of MCMEntities that is also needed by a management system, but which is *not* an inherent part of the MCMEntity being modeled. Hence, it must be treated as a separate object. In the above example, the IP Address is defined as a subclass of MCMNetworkAddress, which in turn is a subclass of MCMInformationResource, and attached to the networking device using the MCMInformationResourceHasMetaData aggregation.

Note that Figure 4 shows two aggregations, called MCMInformationResourceHasMetaData and MCMEntityHasMCMInfoResource. The first enables an MCMInformationResource to optionally aggregate MCMMetaData. The second enables an MCMEntity to be associated with a set of MCMInformationResources. They are discussed in sections 7.11 and 7.10, respectively

7.1.4 Top Portion of the MCMMetaData Hierarchy

The purpose of MCMMetaData is to describe and/or prescribe information about MCMEntity and MCMInformationResource objects. Examples include describing best current practices of using an object, instructing which version(s) of an object to use for a given situation, and to define how to manage the behavior of the system and its constituent components. This makes MCMMetaData objects different than both MCMEntities (whose purpose is to describe the constituent components

of a managed system) as well as MCMInformationResource (whose purpose is to describe information that is not an inherent part of a managed entity, but which nevertheless is important information for the system being managed and is governed by an MCMEntity).

More formally, in the MCM, metadata may describe and/or prescribe information about the object(s) to which it is attached. This is done by "attaching" the metadata object to another object using a relationship, which is typically an aggregation (i.e., a type of "whole-part" relationship). This can be thought of as augmenting the description of that object, and/or attaching management and control information, to that object. Multiple metadata objects may be attached to any single object.

There is often debate as to whether something is metadata or not. In the MCM, a very simple rule is used to make this decision:

[D4] *Metadata **SHOULD** be used to describe a concept that is not part of the inherent characteristics or behavior of an object.*

For example, suppose we were designing a class to represent a Person. An attribute called birthdate would be reasonable, since it is a characteristic of all People. In contrast, an attribute called hairColor is not, since a Person may not have any hair; this could instead be conveyed using metadata. Finally, an attribute called socialSecurityNumber is a poor design for a number of reasons, including (1) social security numbers are typically used only in the US, and (2) there are a number of complex geo-political reasons involving whether a person living in the US even has a social security number.

A much better design is to realize that a social security number is one way to identify a person in a given context. Hence, a more scalable approach would be to define an association between Person and another class, called (for example) PersonalIdentifier. Note that this enables different types of identifiers (e.g., driverLicense, nameAndPassword, biometricData) to be defined a subclasses of PersonalIdentifier. Since each of these have different metadata (e.g., when they should be used), metadata could be attached to each type of identifier.

Figure 5 shows that zero or more metadata objects may be attached to zero or more Entity or InformationResource objects. These are separate aggregations, because the semantics of these relationships are different in nature. Note that an aggregation defines a *whole-part* relationship; this means that three objects are created (the entity that is aggregating metadata, the metadata, and the combination of the entity and its metadata). These relationships are discussed in sections 7.10.1 and 7.11.1, respectively.

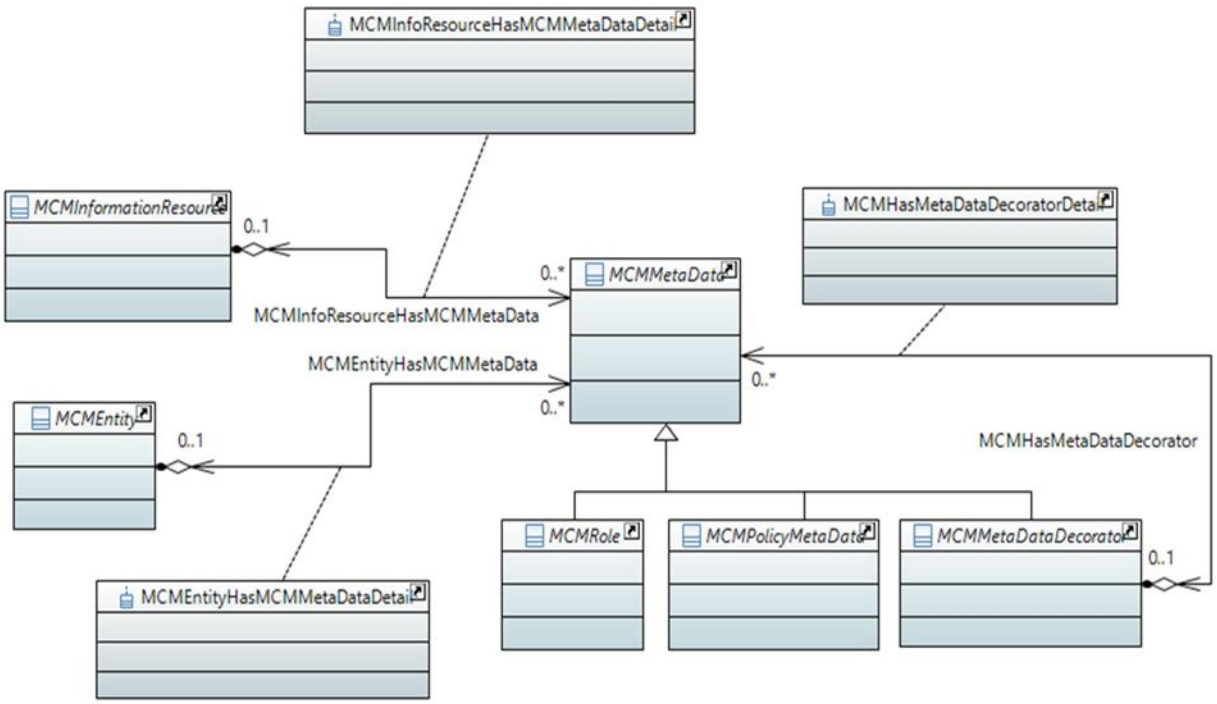

Figure 5. The Top Portion of the MCMMetaData Hierarchy

Metadata is crucial to designing and implementing model-driven software. Most information models either do not specify a metadata hierarchy, or define metadata as embedded within a class. The MCM has chosen to define a separate metadata hierarchy, because:

1) Metadata that is defined within a class makes that metadata available *only* to that class; hence, if the same concept (e.g., versioning, or periods of time within which something is applicable) pertains to other classes, the metadata *is captured as duplicate model elements* (e.g., classes, attributes, operations, constraints, and/or relationships). This creates maintenance issues, as each metadata model object needs to be separately managed.
2) Creating a metadata hierarchy enables a family of objects to be reused to represent common information and behavior that apply to other objects. For example, if the concept of a software version is needed, then defining version as metadata enables any object in the entire model to use a consistent definition of software version.

[O2] Metadata **SHOULD** be optional, since it is used to describe or prescribe the behavior and semantics of another object.

In the MCM, a separate class hierarchy supports attaching a set of metadata objects that can be optionally attached to other objects as needed (e.g., depending on context).

Referring to Figure 5:

- MCMRole is an abstract class, and specializes MCMMetaData. It represents a set of characteristics and behaviors that an object takes on in a particular context. This enables an object to adapt to the needs of different clients through transparently attached role objects. Please see section 7.11.2.1.
- MCMPartyRole is an abstract class, and specializes MCMRole. It represents a set of unique behaviors played by an MCMParty in a given context. Please see section 7.11.2.2.
- MCMPolicyRole is an abstract class, and specializes MCMRole. It represents a set of unique behaviors played by an MCMPolicyObject in a given context. Please see section 7.11.3.

The following classes are not shown in Figure 5 in order to keep the figure simple. Please see the appropriate sections for each class for more detail.

- MCMCustomer is a concrete class, and specializes MCMPartyRole. It represents a particular type of MCMPartyRole that defines a set of people and/or organizations that buy, manage, or use MCMProducts from an MCMServiceProvider. The MCMCustomer is financially responsible for purchasing an MCMProduct. The MCMCustomer is the MCMPartyRole that is purchasing, managing, and/or using Services from an MCMServiceProvider. This definition is based on the definition from [15]. Please see section 7.11.2.3.
- MCMServiceProvider is a concrete class, and specializes MCMPartyRole. It represents a particular type of MCMPartyRole that provides MCMProducts. This specifically includes MCMServices. This definition is based on the definition from [1]. Please see section 7.11.2.4.
- MCMAccessProvider is a concrete class, and specializes MCMPartyRole. It represents a particular type of MCMPartyRole that enables MCMPartyRoles (typically MCMCustomers) to gain entrance to a network (e.g., the Internet), by using an MCMProduct. This specifically includes MCMServices. Please see section 7.11.2.5.
- MCMPartner is a concrete class, and specializes MCMPartyRole. It represents a particular type of MCMPartyRole that provides MCMProducts and MCMServices to the MCMServiceProvider in order to instantiate and manage MCMService elements, such as MCMServiceComponents, external to the Service Provider's Domain. This definition is based on the definition from [1]. Please see section 7.11.2.6.
- MCMCapability is an abstract class, and specializes MCMMetaData. It represents a set of features that are available to be used from an Entity. Each feature may, but does not have to, be used. Please see section 7.11.6.1.
- MCMMEFNetworkFunction is a concrete class, and specializes MCMCapability. It generalizes the concept of an ETSI NFV NetworkFunction, and represents the features and behavior of an MCMManagedEntity that may be used for a given set of external interfaces while in a particular state. It may specify attributes and operations, as well as define nested MEFNetworkFunctions. It may also enumerate the actors that use it. Please see section 7.11.6.2.

- MCMMEFDescriptor is an abstract class, and specializes MCMCapability. It generalizes the concept of an ETSI NFV Descriptor. For example, metadata-driven technologies do not use metadata at design time only; they depend on changing metadata to change behavior. The ETSI NFV Descriptor is a static, design time collection of metadata. In contrast, the MCMMEFDescriptor is metadata that can be used at design time as well as runtime. Please see section 7.11.6.3.
- MCMPolicyMetaData is an abstract class, and specializes MCMMetaData. It represents a set of features and/or behavior that apply to a particular type of MCMPolicyObject (see section 7.11.4).
- MCMGeospatialMetaData is an abstract class, and specializes MCMMetaData. It defines a type of metadata that provides explicit or implicit geographic information. It is defined in ISO 19115:2013 "Geographic Information – Metadata" [14]. Please see section 7.11.5.

7.1.5 MCM Compliance

The MCM defines all common concepts that other models can use.

[D5] In principle, users of a model **SHOULD** be able to find the basic definitions of all concepts that their project needs defined in the MCM.

[D6] If a required concept is not defined in the MCM, then that concept **SHOULD** be added to either the MCM (if it is generally applicable to other models), or to a model derived from the MCM; this enables the MCM, and its derived models, to continually grow and serve the common needs of the MEF modeling community.

[D7] New concepts that are added to the MCM **SHOULD** be in the form of a small number of key model elements. Entire models **SHOULD NOT** be imported into the MCM, as they will likely not be generally applicable to other projects.

For example, if Policy was *not* defined in the MCM, and a project needed to use Policy, then that project should request that Policy be added to the MCM. This does *not* mean that the entire Policy model is added to the MCM; rather, a small set of model elements are added to the MCM hierarchy so that a common Policy model can be built. This is how Policy is currently defined in the MCM.

Note that most projects will need to reference multiple model elements. For example, the Sonata Ordering project will need to use classes, attributes, and relationships from at least the MCMUnManagedEntity hierarchy (e.g., locations and physical entities), MCMManagedEntity hierarchy (e.g., Product, and possible Service, as well as their associated Definitions), MCMParty hierarchy (e.g., people and organizations), MCMBusinessObject hierarchy (e.g., orders and order items), and MCMMetaData hierarchy.

[D8] If a project needs to add model elements (e.g., classes, attributes, relationships, operations, constraints) to the MCM, it **SHOULD** conform to the principles in this section.

The following sections define MCM model elements. Classes are not individually designated as mandatory or optional, because the set of classes that are implemented depends on the application being realized.

[R10] If a class is implemented, then any mandatory model elements defined by that class **MUST** also be implemented.

[R11] Requirement [R10] means that any inherited model elements defined by a class **MUST** also be implemented. In particular, overriding attributes or operations **MUST NOT** be done.

Care should be taken in defining relationships. Relationships are inherited by the classes participating in a relationship.

[D9] Subclasses that inherit relationships from their parent classes **SHOULD NOT** define a relationship that has the same behavior as inherited relationships. While this also applies to attributes and operations, it is much more common in practice to see this requirement not followed.

7.1.6 Alignment With Other SDOs

The ONF TAPI model currently has a very well developed resource model. MCM is committed to using this model (perhaps with suitable modifications) over time. This effort will be mostly complete by version 2 of this model.

Ideally, an object-oriented information model can model a domain regardless of how it is structured technologically (e.g., using a resource- or service-oriented view). The MCM addresses this through the use of established design patterns; this enables the modeler to focus on what is being represented, as opposed to how it is represented (e.g., client-server vs other mechanisms).

At this time, the ONF TAPI model is the only information model being considered for alignment. Alignment with the TMF API data model is slated for work in a future MCM release.

7.1.7 Alignment with Existing MEF Work

The MEF is currently proceeding with multiple modeling projects. Some of these predate the MCM. An overarching goal of the MCM is to incorporate these models without invalidating them. There are three cases to consider: (1) existing models have no superclasses, (2) existing models have superclasses defined in an external model, such as ONF TAPI, and (3) an existing modeling project does not use an MCM pattern, and hence, contains objects that do not directly map to MCM.

The first case is straightforward. MCM, or a model derived from MCM, will define a superclass for all classes in existing MEF models that have no superclasses. This ensures that all MEF models share a common namespace, and can inherit key attributes, such as an objectID, a name, and a description. Note that this case also covers the case of an existing model defining its own “root class”, since that “root class” will inherit from one of MCM’s three subclasses (MCMEntity, MCMInformationResource, or MCMMetaData).

The second case is more complex, since the existing superclass lives in another model. The current ONF TAPI model does *not* have a single superclass, which means that many of its classes do not have superclasses. Hence, for ONF TAPI specifically, this means ensuring that all classes used from the ONF TAPI model by a MEF project have a superclass defined either in the MCM, or in a model derived from the MCM.

The third case is the most complex. For example, ONF TAPI does not use the composite pattern, and instead, uses recursive relationships. This either requires a model mapping (i.e., the ONF TAPI class with a recursive relationship is mapped to an MCM composite pattern) or, for special cases, ignoring the MEF pattern and simply ensuring that the ONF TAPI class inherits from the MCM (or a model derived from the MCM). That being said, the default approach of MCM is to use the composite pattern.

Future alignment with the TMF API data model is for further analysis and work in a future release (note, the TMF API data model is different than a data model produced by the TMF SID; only the TMF API data model is being currently considered). TMF alignment is harder than TAPI alignment due to significant structural differences between the TMF API data model and the MCM – this causes significant semantic differences to be taken into account. Examples include:

- There is no single root class, so some classes have no superclass
- There is no common use of inheritance for key attributes, such as id (rather, they are defined in a class-specific basis)
- There are significant differences in the inheritance hierarchies
- There are significant differences in patterns used
- There is no metadata class, let alone a metadata class hierarchy, in the TMF models

7.2 MCMRootEntity Class Definition

MCMRootEntity is an abstract class. It is the top of the MEF Core Model (MCM) class hierarchy, and specifies a set of attributes and relationships that are common to all other classes in the MCM. The attributes of MCMRootEntity define a common name, a description, and an objectID for all MEF classes. The objectID is defined modularly, so different namespaces can be defined and interoperate. The composite object ID is defined using two class attributes: mcmObjectIDContent and mcmObjectIDFormat. This enables all instances of all objects to be uniquely identified. In the MCM, all classes are rooted. This simplifies implementation.

Table 3 defines the attributes of the MCMRootEntity class.

Attribute Name	Mandatory?	Description
mcmCommonName : String[0..1]	No	This is a string, and represents a user-friendly identifier of an object. It is a name by which the object is commonly known in some limited scope (such as an organization) and conforms to the naming conventions of the scope in which it is used. **[R12]** The mcmCommonName attribute **MUST NOT** be used as a naming attribute (i.e., to uniquely identify an instance of the object). **[D10]** If an object does not have a value for the mcmCommonName attribute, then an empty string **SHOULD** be used.
mcmDescription: String[0..1]	No	This is a string, and defines a textual free-form description of the object. **[D11]** If an object does not have an mcmDescription attribute, then an empty string **SHOULD** be returned.
mcmObjectIDContent: String[1..1]	Yes	The mcmObjectIDContent attribute is a string, and contains the value of the objectID. The mcmObjectIDFormat attribute defines the type of identification that is being used for this object (e.g., URI, GUID, key, or FQDN). The combination of mcmObjectIDContent and mcmObjectIDFormat enables the data model developer to define their own format and content to represent a unique ID of an object. **[R13]** The value of this attribute **MUST NOT** be a NULL or EMPTY string.

mcmObjectIDFormat: String[1..1]	Yes	The mcmObjectIDFormat attribute is a string, and contains the format used by the objectIDContent attribute (e.g., URI, GUID, key, or FQDN). The mcmObjectIDContent attribute is a string, and contains the value of the objectID. The combination of mcmObjectIDContent and mcmObjectIDFormat enables the data model developer to define their own format and content to represent a unique ID of an object. **[R14]** The value of this attribute **MUST NOT** be a NULL or EMPTY string.

Table 3. Attributes of the MCMRootEntity Class

Table 4 defines the operations for the MCMRootEntity class. Note that there are no individual getters and setters for the mcmObjectIDContent and mcmObjectIDFormat attributes, since they are used together as a tuple.

Operation Name	Description
getMCMCommonName() : String[1..1]	This operation returns this object's mcmCommonName attribute as at String. It takes no input parameters. **[D12]** If the mcmCommonName attribute does not have a value, then the getMCMCommonName operation **SHOULD** return an empty String.
setMCMCommonName(in inputString : String[1..1])	This operation sets the current value of the mcmCommonName attribute of this object. It takes a single String parameter, which contains the new value of the mcmCommonName attribute. **[D13]** An empty string **SHOULD** be used to define an empty value for the mcmCommonName attribute.
getMCMObjectID() : String[2..2]	This operation returns this object's mcmObjectID attribute as a String of multiplicity [2] The first element contains the mcmObjectIDContent attribute, and the second contains the mcmObjectIDFormat attribute. This operation takes no input parameters. **[R15]** If either returned parameter is NULL or an empty string, then an exception **MUST** be thrown.

setMCMObjectID(in objectContent : String[1..1], in objectFormat: String[1..1])	This operation sets the current value of the value of the mcmObjectID. The first input parameter is a String, and defines the new value of the mcmObjectIDContent attribute. The second input parameter is a String, and defines the new value of the mcmObjectIDFormat attribute. **[R16]** Both parameters **MUST NOT** be NULL or EMPTY strings.
getMCMDescription() : String[1..1]	This operation returns this object's mcmDescription attribute as at String. It takes no input parameters. **[D14]** If the mcmDescription attribute does not have a value, then the getMCMDescription operation **SHOULD** return an empty String.
setMCMDescription(in inputString : String[1..1])	This operation sets the current value of the mcmDescription attribute of this object. It takes a single String parameter, which contains the new value of the mcmDescription attribute. **[D15]** An empty string **SHOULD** be used to define an empty value for the mcmCommonName attribute.

Table 4. Operations of the MCMRootEntity Class

Note that there are no relationships (i.e., associations, aggregations, or compositions) defined that involve RootEntity. This is because any such relationships would apply to the rest of the MCM classes, and in doing so, would violate many software architecture principles.

7.3 The MCMEntity Hierarchy

MCMEntities represent the characteristics and behavior of the system being managed, and have a separate and distinct existence.
The MCMEntity class has five abstract subclasses, as shown in Figure 6.

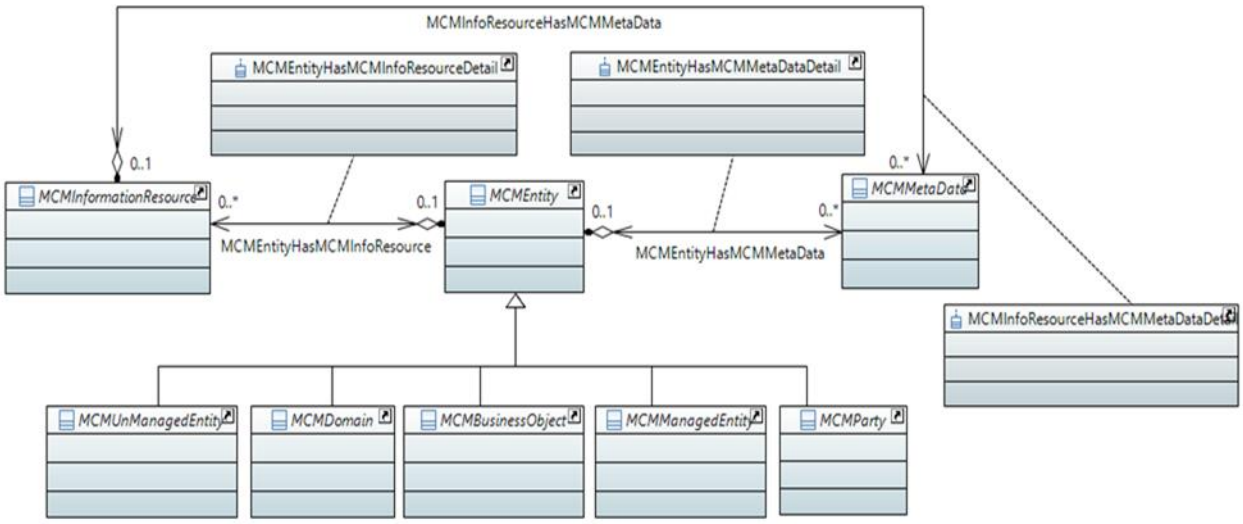

Figure 6. MCMEntity Subclasses

Table 5 defines the purpose of this hierarchy, and aligns them to MEF 55 [1]. The purpose of the MCMEntity hierarchy is to model the major different types of MCMEntities that are of interest to the managed environment. From a classification theory point-of-view, this set of subclasses represent the next level of detail in categorizing what an MCMEntity is.

Name of Class	Function	Relation to MEF 55
MCMEntity	Defines the set of objects that are important to the managed environment	Any object that is monitored or configured is typically a subclass of MCMEntity.
MCMUnManagedEntity	Represents objects that are important to the managed environment, but which have no inherent ability to digitally communicate and be managed. Examples include chassis, location, and cable duct.	Not mentioned; needed for inventory and planning
MCMDomain	A collection of MCMEntities that share a common purpose. In addition, each constituent MCMEntity in an MCMDomain is both uniquely addressable and uniquely identifiable within that MCMDomain.	MCMDomain represents scope of control. It is the superclass of MCMManagementDomain, which is used to apply policy to MCMManagementEntities. Applicable to all MEF55 IRPs.
MCMManagedEntity	Represents objects that have the following common semantics: (1) each has the potential to be managed; (2) each can be associated with at least one ManagementDomain; (3) each is related to Products, Resources, and/or Services of the system being managed.	Superclass of Product, Resource, and Service, as well as templates for their creation and management. Applicable to all MEF55 IRPs.
MCMBusinessObject	Represents business concepts, such as Orders and OrderItems	Enables Business Applications to communicate with other functional components of the LSO RA to order Products, Services, and Resources
MCMParty	Represents either an individual person or a group of people that have a set of (possibly changeable) responsibilities and functions.	Superclass of Person and Organization. Applicable to representing roles that people or organizations play.

Table 5. Functions of the MCMEntity Class and its Subclasses

7.4 MCMEntity Class Definition

This is an abstract class, and specializes MCMRootEntity. It represents objects that are important to the managed environment. Entities represent the characteristics and behavior of the system being managed, and have a separate and distinct existence. An MCMEntity is not just a collection of attributes or an abstraction of behavior. The subclasses of MCMEntity may play one or more business functions, and may be managed or unmanaged (using digital mechanisms). Examples include Chassis, Rack, and CableDuct (unmanaged) and Product, Service, and Resource (managed).

This class does not currently define any attributes. Its significance is from an ontological perspective, as it defines a type of class that is different than its two sibling classes. This is realized by the presence of relationships.

Table 6 defines the operations for this class.

Operation Name	Description
getMCMMetaDataList() : MCMMetaData[1..*]	This operation returns the set of MCMMetaData objects that are currently attached to this particular MCMEntity object. The return value is an array of one or more objects of type MCMMetaData. This operation follows all instances of the MCMEntityHasMCMMetaData aggregation (i.e., from this MCMEntity object to each MCMMetaData object attached to it), and returns the associated MCMMetaData objects as an array. **[D16]** If this object does not have any attached MCMMetaData, then a NULL MCMMetaData object **SHOULD** be returned by the getMCMMetaDataList operation.
setMCMMetaDataList(in attachedMetaDataList : MCMMetaData[1..*])	This operation defines the complete set of MCMMetaData objects that will be attached to this particular MCMEntity object. This operation takes a single input parameter, called attachedMetaDataList, which is an array of one or more MCMMetaData objects. This operation creates a set of aggregations between this particular MCMEntity object and the set of MCMMetaData objects identified in the input parameter (i.e., if there is an array of 5 MCMMetaData objects, then 5 aggregations will be created, where the source for each aggregation is the MCMEntity object and the destination is the appropriate MCMMetaData object in the input parameter list). Note that this operation first deletes any existing attached MCMMetaData objects (and their aggregations and association classes), and then instantiates a new set of MCMMetaData objects; in doing so, each

	MCMMetaData object is attached to this particular MCMEntity object by first, creating an instance of the MCMEntityHasMCMMetaData aggregation, and second, realizing that aggregation instance as an association class. **[D17]** Each aggregation created by the setMCMMetaDataList operation **SHOULD** have an association class (i.e., an instance of the MCMEnttyHasMCM-MetaDataDetail class).
setMCMMetaDataPartialList(in attachedPartialMetaDataList : MCMMetaData[1..*])	This operation defines a set of one or more MCMMetaData objects that will be attached to this particular MCMEntity object WITHOUT affecting any other existing contained MCMMetaData objects or the objects that are contained in them. This operation takes a single input parameter, called attachedPartialMetaDataList, which is an array of one or more MCMMetaData objects. This operation creates a set of aggregations between this particular MCMEntity object and the set of MCMMetaData objects identified in the input parameter. **[D18]** Each aggregation created by the setMCMMetaDataPartialList operation **SHOULD** have an association class (i.e., an instance of the MCMEntityHasMCMMetaDataDetail class).
delMCMMetaDataList()	This operation deletes ALL instances of attached MCMMetaData for this particular MCMEntity. This operation first removes the association class, and second, removes the aggregation, between this MCMEntity object and each MCMMetaData object that is attached to this MCMEntity object. This operation has no input parameters.
delMCMMetaDataPartialList(in attachedPartialMetaDataList : MCMMetaData[1..*])	This operation deletes a set of MCMMetaData objects from this particular MCMEntity. This operation takes a single input parameter, called attachedPartialMetaDataList, which is an array of one or more MCMMetaData objects. This operation first, removes the association class and second, removes the aggregation, between each MCMMetaData object specified in the input parameter and this MCMEntity. Note that all other aggregations between this MCMEntity and other MCMMetaData objects that are not specified in the input parameter are NOT affected.

getMCMInfoResourceList() : MCMInformationResource[1..*]	This operation returns the set of MCMInformationResource objects that are currently attached to this particular MCMEntity object. The return value is an array of one or more objects, of type MCMInformationResource. This operation follows all instances of the MCMEntityHasMCMInfoResource aggregation from this MCMEntity object to each MCMInformationResource object attached to it, and returns the associated MCMInformationResource objects as an array. **[D19]** If this object does not have any attached MCMInformationResource objects, then a NULL MCMInformationResource object **SHOULD** be returned by the getMCMInfoResourceList operation.
setMCMInfoResourceList(in attachedInfoResourceList : MCMInfoResource[1..*])	This operation defines the complete set of MCMInformationResource objects that will be attached to this particular MCMEntity object. This operation takes a single input parameter, called attachedInfoResourceList, which is an array of one or more MCMInformationResource objects. This operation creates a set of aggregations between this particular MCMEntity object and the set of MCMInformationResource objects identified in the input parameter. Note that this operation first deletes any existing attached MCMInformationResource objects (and their aggregations and association classes), and then instantiates a new set of MCMInformationResource objects; in doing so, each MCMInformationResource object is attached to this particular MCMEntity object by first, creating an instance of the MCMEntityHasMCMInfoResource aggregation, and second, realizing that aggregation instance as an association class. **[D20]** Each aggregation created by the setMCMInfoResourceList operation **SHOULD** have an association class (i.e., an instance of the MCMEntityHasMCMInfoResourceDetail class).
setMCMInfoResourcePartialList(in attachedInfoResourcePartialList : MCMInfoResource[1..*])	This operation defines a set of one or more MCMInformationResource objects that will be attached to this particular MCMEntity object WITHOUT affecting any other existing contained MCMInformationResource objects or the objects that are contained in them. This operation takes a single input parameter, called attachedPartialInfoResourceList, which is an array of one or more MCMInformationResource objects. This operation creates a set of aggregations between this particular

	MCMEntity object and the set of MCMInformationResource objects identified in the input parameter. **[D21]** Each created created by the setMCMInfoResourcePartialList aggregation **SHOULD** have an association class (i.e., an instance of the MCMEntityHasMCMInfoResourceDetail class).
delMCMInfoResourceList()	This operation deletes ALL instances of attached MCMInformationResource objects for this particular MCMEntity. This operation first, removes the association class, and second, removes the aggregation, between this MCMEntity object and each MCMInformationResource object that is attached to this MCMEntity object. This operation has no input parameters.
delMCMInfoResourcePartialList(in attachedPartialMetaData : MCMMetaData[1..*])	This operation deletes a set of MCMInformationResource objects from this particular MCMEntity. This operation takes a single input parameter, called attachedpartialInfoResourceList, which is an array of one or more MCMInformationResource objects. This operation first, removes the association class and second, removes the aggregation, between each MCMInformationResource object specified in the input parameter and this MCMEntity. Note that all other aggregations between this MCMEntity and other MCMInformationResource objects that are not identified in the input parameter are NOT affected.

Table 6. Operations of the MCMEntity Class

MCMEntity defines two relationships, called MCMEntityHasMCMInfoResource and MCMEntityHasMCMMetaData, as shown in Figure 6.

MCMEntityHasMCMInfoResource is an aggregation, and defines the set of MCMInformationResource objects that are associated with this particular set of MCMEntity objects. Its multiplicity is defined to be 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMInformationResource objects can be aggregated by this particular MCMEntity object. Note that the cardinality on the part side (MCMInformationResource) is 0..*; this enables an MCMEntity object to be defined without having to define an association MCMInformationResource object. The semantics of this aggregation are defined by the MCMEntityHasMCMInfoResourceDetail association class. This enables the semantics of the aggregation to be defined using the attributes and behavior of this association class. For example, it can be used to define which MCMInformationResource objects are allowed to be associated with which MCMEntity objects.

MCMEntityHasMCMMetaData is an aggregation, and defines the set of MCMMetaData objects that are associated with this particular set of MCMEntity objects. Its multiplicity is defined to be 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If

this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMMetaData objects can be aggregated by this particular MCMEntity object. Note that the cardinality on the part side (MCMMetaData) is 0..*; this enables an MCMEntity object to be defined without having to define an MCMMetaData object for it to aggregate. The semantics of this aggregation are defined by the MCMEntityHasMCMMetaDataDetail association class. This enables the semantics of the aggregation to be defined using the attributes and behavior of this association class. For example, it can be used to define which MCMMetaData objects are allowed to be associated with which MCMEntity objects.

Both of the above association classes can be further enhanced by using the Policy Pattern (see Figure 3) to define policy rules that constrain which part objects (i.e., MCMMetaData) are attached to which object. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.4.1 MCMEntityHasMCMMetaDataDetail Class Definition

This is an association class. Its purpose is to define descriptive and/or prescriptive characteristics and behavior of the MCMEntity object that this MCMMetaData object is aggregated by. Table 7 defines the attributes for this class.

Attribute Name	Mandatory?	Description
mcmEntityEnableStatus : MCMEntityEnable[0..1]	NO	This enumeration defines whether the MCMEntity object that this MCMMetaData object refers to is enabled for normal operation or not. The values are defined in the MCMEntityEnable enumeration, and include: ERROR INIT ENABLED_FOR_ALL ENABLED_FOR_TEST_ONLY DISABLED UNKNOWN **[D22]** The default value for the mcmEntityEnableStatus attribute **SHOULD** be 1.
mcmEntityValidEndTime: TimeAndDate[0..1]	NO	This is a TimeAndDate attribute; it contains a datestamp and a timestamp. It defines the date and time that the MCMEntity to which this MCMMetaData is attached is no longer valid and available to be used. **[D23]** This attribute **SHOULD** have a complete and valid time and/or date. **[O3]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.

mcmEntityValidStartTime: TimeAndDate[0..1]	NO	This is a TimeAndDate attribute; it contains a date-stamp and a timestamp. It defines the date and time that the MCMEntity to which this MCMMetaData is attached is valid and available to be used. **[D24]** This attribute **SHOULD** have a complete and valid time and/or date. **[O4]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.

Table 7. Attributes of the MCMEntityHasMCMMetaDataDetail Association Class

Operation Name	Description
getMCMEntityEnableStatus : MCMEntityEnable[1..1]	This operation returns the mcmEntityEnableStatus of this set of MCMEntity and MCMMetaData objects. The return value is one of the literals defined by the MCMEntityEnable enumeration, and signifies whether the MCMMetaData applied to this MCMEntity enables it to be used or not.
setMCMEntityEnableStatus(in inputStatus : MCMEntityEnable [1..1])	This operation sets the current value of the mcmEntityEnableStatus of this set of MCMEntity and MCMMetaData objects. It takes a single input parameter, of type MCMEntityEnable, which is an enumeration that defines whether the MCMMetaData applied to this MCMEntity enables it to be used or not.
getMCMEntityValidEndTime : TimeAndDate[1..1]	This operation returns the date and time at which this Entity is no longer valid and hence, not able to be used. It returns a datatype of type TimeAndDate. **[D25]** This attribute **SHOULD** have a complete and valid time and/or date. **[O5]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
setMCMEntityValidEndTime(in endTime : TimeAndDate[1..1])	This operation sets the current value of the date and time at which this Entity is no longer valid and able to be used. It takes a single input parameter, which is of type TimeAndDate; this is used to set the mcmEntityValidityEndTime to a new value.

getMCMEntityValidStartTime : TimeAndDate[1..1]	This operation sets the current value of the date and time at which this Entity is valid and able to be used. It takes a single input parameter, which is of type TimeAndDate; this is used to set the mcmEntityValidityStartTime to a new value. **[D26]** This attribute **SHOULD** have a complete and valid time and/or date. **[O6]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
setMCMEntityValidStartTime(in startTime : TimeAndDate[1..1])	This operation sets the current value of the date and time at which this Entity is first valid and able to be used. It takes a single input parameter, which is of type TimeAndDate; this is used to set the mcmEntityValidityEndTime to a new value.

Table 8. Operations of the MCMEntityHasMCMMetaDataDetail Association Class

7.5 MCMUnManagedEntity Class Hierarchy

The MCMUnManagedEntity class has two subclasses, as shown in Figure 7

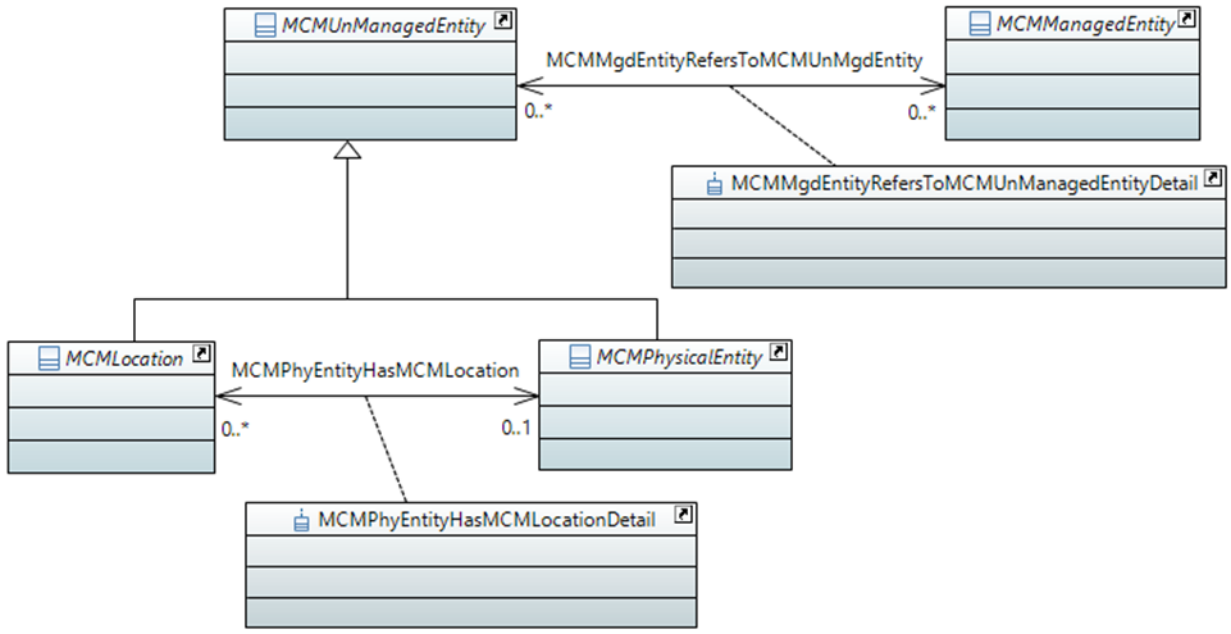

Figure 7. MCMUnManagedEntity Subclasses

The purpose of the MCMUnManagedEntity hierarchy is to model the major different types of MCMEntities that cannot be intrinsically managed, yet are of interest to the managed environment. Note: in the MCM, any purely physical object is defined as unmanageable. Examples include geographic areas, building, Racks, Chassis, and other purely physical Entities. Management capabilities are provided by the logical objects that are attached to a physical object. MCMUnMangedEntity objects are important to the managed environment because they provide context (e.g., where a customer premise equipment is located) and a point of reference (e.g., ensure that cell coverage covers this geographic area).

Table 9 defines the purpose of this hierarchy, and aligns them to MEF 55 [1].

Name of Class	Function	Relation to MEF 55
MCMUnManagedEntity	Represents Entities that are important to the managed environment that have no inherent ability to digitally communicate and be managed.	Not mentioned, but clearly needed for Inventory, Order, and other functions
MCMLocation	Represents points or areas that contain physical objects that are important to the managed environment.	Some types of locations (e.g., Sites) are mentioned, but needs a more general model.
MCMLocationAtomic	A subclass of MCMLocation that represents stand-alone Locations.	Not mentioned, but clearly needed. Examples include a stand-alone structure or area.
MCMLocationComposite	A subclass of MCMLocation that represents a set of Locations that form a tree-like hierarchy.	Not mentioned, but clearly needed. Examples include nested Locations (e.g., a rack within a wiring closet within a floor within a building…).
MCMPhysicalEntity	Represents physical Entities,that are important to the managed environment that cannot be managed electronically.	Examples include Rack, Chassis, Slot, Port, Card, Cable Duct, Shelf.

Table 9. Functions of the MCMUnManagedEntity Class and its Subclasses

7.5.1 MCMUnManagedEntity Class Definition

This is an abstract class, and specializes MCMEntity. It represents MCMEntities that are important to the managed environment, but which have no inherent ability to digitally communicate with other MCMEntities. Hence, they cannot be managed by digital mechanisms.

The current version of this CfCB defines two main subclasses of MCMUnManagedEntity, called MCMLocation and MCMPhysicalEntity. They are described further in sections 7.5.2 and 7.5.6 below.

This class defines a single attribute, which is defined in Table 10.

Attribute Name	Manda-tory?	Description
mcmIsTopo-nym[0..1]	NO	This is a Boolean attribute. If the value of this attribute is TRUE, then this MCMUnManagedEntity is a toponym (i.e., a name of a place). Examples include “CustomerSiteLocation” and “ArchivalFacility”. The value of the toponym MAY be contained in the mcmCommonName attribute, or in a cus-tom attribute added to this class.

Table 10. Attributes of the MCMUnManagedEntity Class

Table 11 defines the operations for the MCMUnManagedEntity Class.

Operation Name	Description
getMCMIsToponym : Boolean[1..1]	This operation returns the value of the mcmIsToponym attribute. If the value of this attribute is TRUE, then this MCMUnManagedEntity is a toponym (i.e., a name of a place). Examples include “CustomerSiteLocation” and “ArchivalFacility”.
setMCMIsToponym(in isAToponym : Boolean[1..1])	This operation sets the current value of the mcmIsTopo-nym attribute. It contains a single input parameter, of type Boolean. If the value of this attribute is TRUE, then this MCMUnManagedEntity is a toponym (i.e., a name of a place). Examples include “CustomerSiteLocation” and “ArchivalFacility”.

Table 11. Operations of the MCMUnManagedEntity Class

At this time, no relationships are defined for the MCMUnManagedEntity class. It does participate in one relationship, called MCMMgdEntityRefersToMCMUnManagedEntity; see section 7.8.1.

7.5.2 MCMLocation Class Design

This section provides background information that describes the design of the MCMLocation class hierarchy. Location design can be very complex, as multiple different factors (e.g., local conventions describing geographic areas, the coordinate system used, and internationalization factors) must be considered. The current MCM design provides a simplified approach that can include these and other factors later.

7.5.2.1 Requirements

The design of the MCMLocation class hierarchy meets the following requirements:

[O7] Any physical entity **MAY** have an associated physical location; this is met by defining an aggregation, called MCMPhyEntityHasMCMLocation, which is shown in Figure 7.

[O8] Any managed entity **MAY** have an associated physical location; this is met by defining an association, called MCMMgdEntityRefersToMCMUnManaged-Entity, which is shown Figure 7.

[O9] Any managed entity **MAY** have an associated physical location; this is met by defining an association, called MCMMgdEntityRefersToMCMUnManaged-Entity, which is shown in Figure 7.

[O10] Locations **MAY** be defined as stand-alone or hierarchical structures (e.g., a single location, such as a postal address, or the location of a room on a floor in a building at a site); this is met by using the composite pattern to define atomic and composite locations (i.e., MCMLocationAtomic and MCMLocationComposite – see sections 7.5.4 and 7.5.5, respectively).

[R17] Location data **MUST** be specified as either a geocode or a set of points that bound an area (e.g., a polygon).

[D27] Geocode data **SHOULD** be provided in text, and **SHOULD** be defined as either relative or absolute.

[R18] Relative geocodes are textual descriptions of a location that, by itself, cannot provide an exact location. A relative geocode **MUST** be specified using one or more absolute geocodes as a reference. For example, “The nearest building northwest of building A3” is a relative geocode that uses the location of building A3 as its reference. In contrast, absolute geocodes are textual descriptions of a location that, by itself, can provide an exact location. For example, a USPS ZIP code (or even a USPS ZIP+4 code) is considered an absolute geocode. However, it is thought of as a polygon. Different geocoding systems use different computation mechanisms (e.g., a centroid) to define the “center” of such an area.

[O11] Note: there is a surprising amount of variability in expressing an address. The geocoding process **MAY** use additional mechanisms, such as address normalization, to reduce this variability. Some geocoders also provide a degree of precision or confidence in their result.

7.5.2.2 Design

One approach to enabling a location to reference another location is to define an attribute for referencing the name of the class that it refers to. This is a poor choice for at least two reasons. First, a location is a Class, and hence, using an attribute to refer to the name of a Class is better accomplished by using an association. Second, what if there are multiple references (e.g., a street address might not correspond to a known street address in the geocoding database, so it is common practice to use two addresses and interpolate).

Given that we need an association, the next decision is, between which objects? This depends on how location is represented. For geocodes, the typical practice is to provide a set of input data, and use a geocode service to turn those data into a geocode. This is complicated by the fact that the actual location (e.g., a postal address, or even a land parcel) is owned by a different administrative authority (e.g., the government). Hence, in this version of the MCM, a geocode is modeled as a subclass of InformationResource (see section 7.10). Since an aggregation already exists between MCMEntity and MCMInformationResource (see section 7.10.1), all that is needed is to define a new subclass of MCMInformationResource, called MCMGeocode; this is shown in Figure 8.

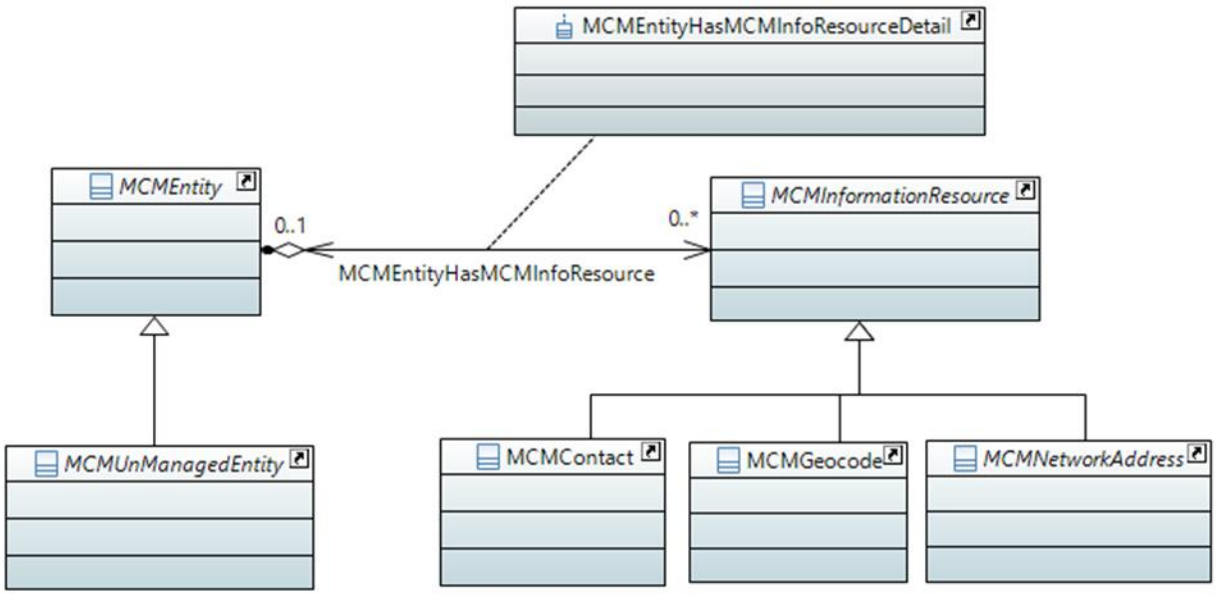

Figure 8. Representing Geocodes in MCM

Note that any subclass of MCMUnManagedEntity may have any of the three subclasses of MCMInformationResource; the particular set of MCMInformationResources are restricted by the ue of the MCMEntityHasMCMInfoResourceDetail association class.

Figure 9 shows both the MCMLocation and MCMPhysicalEntity class hierarchies and their relevant relationships from the previous discussions.

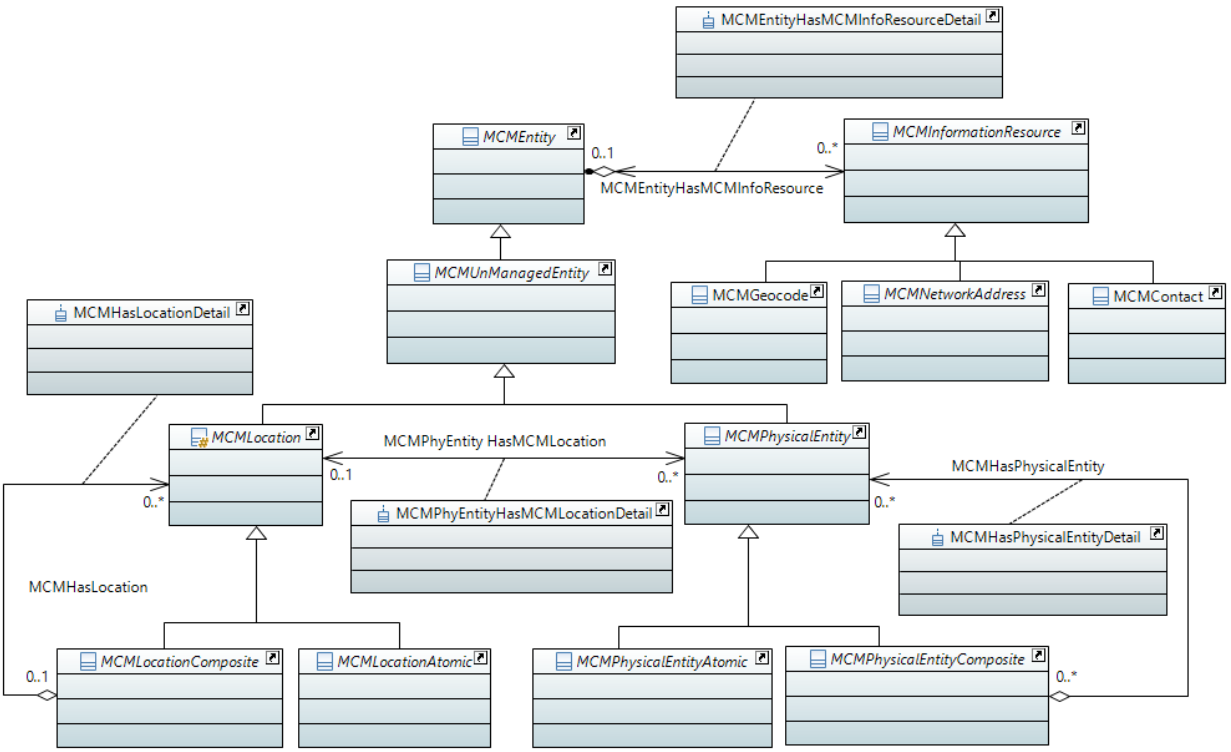

Figure 9. MCMLocation and MCMPhysicalEntity Hierarchies

7.5.3 MCMLocation Class Definition

This is an abstract class, and specializes MCMUnManagedEntity. It represents a point or area that an Entity may occupy. An MCMLocation can be one of two things: (1) a unique estimated or actual geolocation, or (2) the coordinates of an enclosing container (e.g., a polyhedron) that defines the perimeter of the location. In either case, MCMGeospatialMetaData can be used to provide additional descriptive and/or prescriptive data as required (e.g., building colors and entrance instructions for different times, respectively). It has two subclasses, MCMLocationAtomic and MCMLocationComposite, which are described in Sections 7.5.4 and 7.5.5, respectively.

Metadata information is a key part of any geolocation. Several standards exist on defining geospatial metadata information. Hence, the MCM provides a subclass of MCMMetaData, called MCMGeoSpatialMetaData, to represent such information. Note that an explicit relationship between MCMLocation and MCMMetaData is *not* required, since MCMLocation is a subclass of MCMEntity, and MCMEntityHasMCMMetaData already exists to aggregate MCMMetaData to MCMEntities.

Another example of metadata is to provide generic information, such as information that categorizes the business role that a particular MCMLocation plays (e.g., "Customer Premise", "UNI Site", or "Billing Address"). This is implemented using the MCMEntityHasMCMMetaData aggregation (note that this aggregation is inherited from MCMEntity). Another use of this approach is to define information describing or prescribing characteristics and behavior of the location. For example, metadata could be used to provide off-hour entry instructions to a building.

Note that [15] defines four different types of address formatting options (i.e., fielded address, formatted address, address reference, and geographic point). Since each of these options are really complex data structures, this is implemented in the MCM using metadata to represent each of these addresses. Once again, the MCMEntityHasMCMMetaData is used to attach the appropriate subclasses of MCMMetaData to the appropriate subclasses of MCMLocation. Table 12 defines the attributes of the MCMLocation class.

Attribute Name	Mandatory?	Description
mcmIsAbsoluteData : Boolean[1..1]	YES	This is a Boolean attribute. If the value of this attribute is TRUE, then the mcmLocationData class attribute contains absolute input data. Otherwise, the mcmLocationData class attribute contains relative data. Note that relative input defines a relative geocode, which is dependent on (and geographically relative to) other geocode locations.
mcmIsEstimatedLocation : Boolean[0..1]	NO	This is a Boolean attribute. If the value of this attribute is TRUE, then this location is an estimated value. Otherwise, this location is a precise value.
mcmIsFixedBoundary : Boolean[0..1]	NO	This is a Boolean attribute. If the value of this attribute is TRUE, then this MCMlLocation has well-defined boundaries. Otherwise, one or more boundaries of this MCMLocation are ambiguous and/or can change.
mcmIsGeocodeLocation : Boolean[1..1]	YES	This is a Boolean attribute. If the value of this attribute is TRUE, then this location is an actual or estimated geocode. Otherwise, this location is described by an enclosing container (e.g., a polyhedron). In both cases, the location (i.e., the geocode or the points defining the polyhedron) are defined by the mcmLocationDataList class attribute.
mcmLocationDataList : String[1..*]	YES	This is an array of string attributes. Each string in this attribute contains input data to determine the location. If the mcmIsGeocodeLocation class attribute is TRUE, then the data contained in this attribute is the input to a geocoding process. This may consist of one or more attributes. Otherwise, the data contained in this attribute contain the coordinates of an enclosing container of this

		MCMLocation; this is typically defined as one attribute per coordinate. **[D28]** If the value of the mcmLocationDataList attribute is not known, then an empty string **SHOULD** be returned.

Table 12. Attributes of the MCMLocation Class

Table 13 defines the operations for the MCMLocation Class.

Operation Name	Description
getMCMIsAbsoluteData() : Boolean[1..1]	This operation returns the value of the mcmIsAbsoluteData attribute. If the value of this attribute is TRUE, then this MCMLocation represents absolute location data. Otherwise, this MCMLocation object represents relative location data (i.e., the location is relative to another location).
setMCMIsAbsoluteData(isAbsData : Boolean[1..1])	This operation sets the current value of the mcmIsAbsoluteData attribute. This operation takes a single input parameter, of type Boolean, which is used to change the value of the mcmIsAbsoluteData class attribute. A value of TRUE means that this MCMLocation object is defined using absolute data; otherwise, this MCMLocation object is defined relative to another MCMLocation object.
getMCMIsEstimatedLocation() : Boolean[1..1]	This operation returns the value of the mcmIsEstimatedLocation attribute. If the value of this attribute is TRUE, then this MCMLocation represents an estimated location. Otherwise, this MCMLocation object represents a precise location.
setMCMIsEstimatedLocation(in isEstimate : Boolean[1..1])	This operation sets the current value of the mcmIsEstimatedLocation attribute. This operation takes a single input parameter, of type Boolean, which is used to change the value of the mcmIsEstimatedLocation class attribute. A value of TRUE means that it is an estimated location, while a value of FALSE means that it is a precise location.
getMCMIsFixedBoundary() : Boolean[1..1]	This operation returns the value of the mcmIsFixedBoundary attribute. If the value of this attribute is TRUE, then this MCMLocation has a fixed boundary. Otherwise, this MCMLocation object has one or more boundaries that can change.

setMCMIsFixedBoundary(in isFixed : Boolean[1..1])	This operation sets the current value of the mcmIsFixedBoundary attribute. This operation takes a single input parameter, of type Boolean, which is used to change the value of the mcmIsFixedBoundary class attribute. If the input variable is TRUE, then this MCMLocation has a fixed boundary; otherwise, it contains one or more boundaries that can change.
getMCMIsGeocodeLocation() : Boolean[1..1]	This operation returns the value of the mcmIsGeocodeLocation attribute. If the value of this attribute is TRUE, then this MCMLocation represents an actual or estimated geocode (which is defined by the mcmLocationDataList class attribute). Otherwise, this MCMLocation object represents a location that is described by an enclosing contaiiner (e.g., a polyhedron).
setMCMIsGeocodeLocation(in isGeocode : Boolean[1..1])	This operation sets the current value of the mcmIsGeocodeLocation attribute. This operation takes a single input parameter, of type Boolean, which is used to change the value of the mcmIsGeocodeLocation class attribute. If the input variable is TRUE, then this MCMLocation is defined by a geocode; otherwise, it is defined by a polyhedron.
getMCMLocationDataList() : String[1..*]	This operation returns the value of the mcmLocationDataList attribute. The return value is an array of Strings that collectively define either the geocode or each point in a surrounding polyhedron that defines this MCMLocation. **[D29]** If this object does not have a value for the mcmLocationDataList attribute, then a NULL string **SHOULD** be returned by the getMCMLocationDataList operation.
setMCMLocationDataList(in locationDataList : String[1..*])	This operation sets the current value of the mcmLocationDataList attribute. This operation takes a single input parameter, of type String[1..*], which is used to change the value of the mcmLocationDataList class attribute. The mcmLocationDataList class attribute defines the data describing the boundary of the MCMLocation either as a geocode or as a polyhedron.
getMCMLocationParent() : MCMLocationComposite[1..1]	This operation returns the parent of this MCMLocation object. **[D30]** If this MCMLocation object has no parent, then a NULL MCMLocationObject **SHOULD** be returned.
setMCMLocationParent(in newParent : MCMLocationComposite[1..1])	This operation defines the parent of this MCMLocation object. **[R19]** If this MCMLocation object already has a parent, then an exception **MUST** be raised.

getMCMPhyEntityListAtLocation() : MCMPhysicalEntity[1..*]	This operation returns the set of MCMPhysicalEntity objects that are at this particular MCMLocation. This is done by following each instance of the MCMPhyEntityHasMCMLocation association, and taking into effect any semantics defined by the MCMPhyEntityHasMCMLocationDetail association class. This operation takes no input parameters. **[D31]** If no MCMPhysicalEntity objects are associated with this particular MCMLocation, then a NULL MCMPhysicalEntity object **SHOULD** be returned**.**
setMCMPhyEntityListAtLocation (in phyEntityList : MCMPhysicalEntity[1..*])	This operation defines a set of MCMPhysicalEntity objects that are associated with this particular MCMLocation. This operation takes a single input parameter, called phyEntityList, which is an array of one or more MCMPhysicalEntity objects. This operation creates a set of aggregations between this particular MCMLocation object and the set of MCMPhysicalEntity objects identified in the input parameter This is done by instantiating an instance of the MCMPhyEntityHasMCMLocation association for each MCMPhysicalEntity in the input parameter, and then realizing that association with an instance of the MCMPhyEntityHasMCMLocationDetail association class. Note that this operation first deletes any existing associated MCMPhysicalEntity objects (and their aggregations and association classes), and then instantiates a new set of MCMPhysicalEntity objects; in doing so, each MCMPhysicalEntity object is attached to this particular MCMLocation object by first, creating an instance of the MCMPhyEntityHasMCMLocation aggregation, and second, realizing that aggregation instance as an association class. **[D32]** When the setMCMPhyEntityListAtLocation operation is executed, each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMPhyEntityHasMCMLocationDetail class).
setMCMPhyEntityPartialListAtLocation (in phyEntityPartialList: MCMPhysicalElement[1..*])	This operation defines a set of one or more MCMPhysicalEntity objects that should be associated with this particular MCMLocation object WITHOUT affecting any other existing contained MCMLocation objects or the objects that are contained in them. This operation takes a single input parameter, called phyEntityPartialList, which is an array of one or more MCMPhysicalEntity objects. This operation creates a set of aggregations between this particular MCMLocation object and the set of MCMPhysicalEntity objects identified in the input parameter. **[D33]** When the setMCMPhyEntityPartialListAtLocation operation is executed, each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMPhyEntityHasMCMLocation-Detail class).

delMCMPhyEntityAtLocation()	This operation deletes ALL instances of MCMPhysicalEntity objects that are related to this particular MCMLocation object. This operation first removes the association class, and second, removes the association, between this MCMLocation object and each MCMPhysicalEntity object that is attached to this MCMLocation object. This operation has no input parameters.
delMCMPhyEntityPartialList-AtLocation(in phyEntityList : MCMPhysicalEntity[1..*])	This operation deletes the set of instances of MCMPhysicalEntity objects that are specified in the phyEntityList parameter that are related to this particular MCMLocation object. This operation first removes the association class, and second, removes the association, between this MCMLocation object and each MCMPhysicalEntity object that is specified in the phyEntityList (that is attached to this MCMLocation) parameter. All other associations between this particular MCMLocation object and other MCMPhysicalEntity objects that are not specified in the phyEntityList parameter are NOT affected.

Table 13. Operations of the MCMLocation Class

At this time, no relationships are defined for the MCMLocation class, although it participates in two relationships, MCMHasLocation and MCMPhyEntityHasMCMLocation (see sections 7.5.5 and 7.5.6, respectively).

7.5.4 MCMLocationAtomic Class Definition

This is an abstract class, and specializes MCMLocation. This class represents stand-alone MCMLocation objects. In addition, each MCMLocationAtomic has characteristics and behavior that is externally visible. Examples include a single building that is not related to other buildings, or the location of a cable duct (remember, an MCMLocation can be a polyhedron).

[R20] This class **MUST NOT** contain another MCMLocation object.

At this time, no attributes are defined for the MCMLocationAtomic class.

At this time, no operations are defined for the MCMLocationAtomic class.

At this time, no relationships are defined for the MCMLocationAtomic class.

7.5.5 MCMLocationComposite Class Definition

This is an abstract class, and specializes MCMLocation. This class represents a set of related MCMLocation objects that are organized into a tree structure. Its primary use is to collect other types of MCMLocation objects.

[O12] Each MCMLocationComposite object **MAY** contain zero or more MCMLocationAtomic and/or zero or more MCMLocationComposite objects.

For example, a Building may contain floors, floors may contain rooms, rooms may contain wiring closets, and wiring closets may contain other physical entities (e.g., racks and chassis) that in turn contain equipment (e.g., Computers, Routers, and Switches) that are of interest to the managed environment. In this example:

- A building is a type of MCMLocationComposite, since its purpose is to contain other MCMEntities
- A floor is a type of MCMLocationComposite; while it does not “contain” anything, other physical entities may be put “on” a floor (which yields the same result).
- A room is a type of MCMLocationComposite, since the purpose of the room is to contain things.
- Similarly, a Wiring Closet is a type of Room that contains physical equipment and electrical connections; hence, it is also an MCMLocationComposite.
- A Rack is a standardized enclosure for mounting multiple electronic equipment modules; hence, a Rack is an MCMLocationComposite.
- A Chassis is a standardized enclosure that contains the components that make up a type of equipment (e.g., a computer or router); hence, a Chassis is an MCMLocationComposite.
- Equipment frames, such as a Computer or Router or Switch, are all examples of an MCMLocationComposite, since they contain other physical components.
- Physical port is an example of an MCMLocationAtomic.

[O13] Each of these physical entities **MAY** have an associated location, which is defined using the MCMPhyEntityHasMCMLocation aggregation in and discussed in section 7.5.6.

At this time, no attributes are defined for the MCMLocationComposite class. Most attributes will likely be realized using relationships and/or operations. For example, a query to an instance of the MCMLocationComposite class to provide its set of contained MCMLocations (e.g., the floor(s) of a building in a site) will be done by using class operations; the MCMLocationComposite instance will query each of its contained MCMLocations (which will in turn call their operations to acquire their MCMLocations), aggregate and organize the information, and provide that information in its operation response.

Table 14 defines the operations for the MCMLocationComposite class.

Operation Name	Description
getMCMLocationChild-List() : MCMLocation[1..*]	This operation returns the set of all MCMLocation objects that are contained in this specific MCMLocationComposite object. There are no input parameters to this operation. This operation returns a list of one or more MCMLocation objects (i.e., the list is made up of MCMLocationAtomic and/or MCMLocationComposite objects). **[D34]** If no MCMLocation objects are found, then this operation **SHOULD** return a NULL MCMLocation object).
setMCMLocationChildList(childObjectList : MCMLocation[1..*])	This operation defines a set of MCMLocation objects that will be contained by this particular MCMLocationComposite object. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMLocation objects (i.e., one or more MCMLocationAtomic and/or MCMLocationComposite objects). This has the effect of creating an instance of the MCMHasLocation aggregation between each MCMLocation object in the childObjectList and this particular MCMLocationComposite object. Note that this operation first deletes any existing contained MCMLocation objects (and their aggregations and association classes), and then instantiates a new set of MCMLocation objects; in doing so, each MCMLocation object is contained within this particular MCMLocationComposite object by first, creating an instance of the MCMHasLocation aggregation, and second, realizing that aggregation instance as an association class.

	[D35] When this operation is executed, each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasMCMLocationDetail class).
setMCMLocationPartial-ChildList(in childObject-List : MCMLocation[1..*])	This operation defines a set of one or more MCMLocation objects that should be contained within this particular MCMLocationComposite object WITHOUT affecting any other existing contained MCMLocation objects or the objects that are contained in them. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMLocation objects. This has the effect of creating a set of aggregations between this particular MCMLocationComposite object and each of the MCMLocation objects identified in the childObjectList. **[D36]** When this operation is executed, each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasMCMLocationDetail class).
delMCMLocationChild-List()	This operation deletes ALL contained MCMLocation objects of this particular MCMLocationComposite object. This has the effect of first, removing the association class, and second, removing the aggregation, between this MCMLocationComposite object and each MCMLocation object that is contained in this MCMLocationComposite object. This operation has no input parameters.
delMCMLocationPartial-ChildList (in childObjectList : MCMLocation[1..*])	This operation deletes a set of MCMLocation objects from this particular MCMLocationComposite object WITHOUT affecting any other existing contained MCMLocation objects or the objects that are contained in them. This operation takes a single input parameter, called childLocationList, which is an array of one or more MCMLocation objects. This has the effect of first, removing the association class and second, removing the aggregation, between each MCMLocation object specified in the input parameter and this MCMLocationComposite object. Note that all other aggregations between this MCMLocationComposite and other MCMLocation objects that are not identified in the input parameter are NOT affected.

Table 14. Operations for the MCMLocationComposite Class

The MCMLocationComposite class defines a single aggregation, called MCMHasLocation. This aggregation is used to define the set of MCMLocation objects that are contained within this particular MCMLocationComposite object. Its multiplicity is defined to be 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is

instantiated (e.g., the “1” part of the 0..1 cardinality), then zero or more MCMLocation objects can be associated with this particular MCMLocationComposite object. Note that the cardinality on the part side (MCMLocation) is 0..*; this enables an MCMLocationComposite object to be defined without having to define an MCMLocation object for it to aggregate.

The semantics of the MCMHasLocation aggregation is realized using an association class, called MCMHasLocationDetail. This enables the semantics of the MCMHasLocation aggregation to be realized using the attributes, operations, and relationships of the MCMHasLocationDetail association class. The Policy Pattern may be used to control which specific MCMLocation objects are contained within a given MCMLocationComposite object for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

The MCMLocation class also participates in a second association, called MCMPhyEntityHasMCMLocation. Please see section 7.5.6.

7.5.6 MCMPhysicalEntity Class Definition

This is an abstract class, and specializes MCMUnManagedEntity. It represents MCMEntities that are important to the managed environment that have a physical form. They cannot be managed electronically. Examples include Rack, Chassis, CableDuct, and Card. The composite pattern is applied to MCMPhysicalEntity to enable stand-alone as well as hierarchies of MCMPhysicalEntities to be represented. This is described in sections 7.5.7 and 7.5.8, respectively.

Note that some attributes, such as the revision number of a hardware component, are defined by MCMMetada classes. This differs from other implementations, which typically define such attributes in the equivalent of this class (note that other implementations typically do not have a formal metadata class hierarchy, and hence, have no alternative). This was done in the MCM in order to accommodate more use cases and provide flexibility in defining MCMPhysicalEntities.

Table 15 defines the attributes of the MCMPhysicalEntity class.

Attribute Name	Mandatory?	Description
mcmAssetID : String[0..1]	NO	This is a string attribute. It contains a user-assigned asset tracking identifier for the component. **[R21]** The mcmAssetID attribute **MUST NOT** be used as an objectID, since one is inherited from MCMRootEntity. **[D37]** If an mcmAssetID attribute is not assigned, then the value of this attribute **SHOULD** be set to an empty string.
mcmManufactureDate : TimeAndDate[1..1]	YES	This is a TimeAndDate attribute, and contains the date and time of the manufacturing of this object. **[D38]** This attribute **SHOULD** have a complete and valid time and/or date. **[O14]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
mcmManufacturer : String[0..1]	NO	This is a string attribute. It contains the name of the manufacturer of this object. **[D39]** If the Manufacturer is not known, then the value of the mcmManufacturer attribute **SHOULD** be set to an empty string.
mcmSerialNumber : String[1..1]	YES	This is a string attribute. It contains the serial number of this object. **[D40]** If an mcmSerialNumber attribute is not assigned, then the value of this attribute **SHOULD** be set to an empty string.

Table 15. Attributes of the MCMPhysicalEntity Class

Table 16 defines the following operations for the MCMPhysicalEntity class.

Operation Name	Description
getMCMAssetID() : String[1..1]	This operation returns the mcmAssetID attribute of this MCMPhysicalEntity object. There are no input parameters to this operation. **[D41]** If an mcmAssetID attribute is not assigned, then the returned value of the getMCMAssetID operation **SHOULD** be set to an empty string.
setMCMAssetID(in newAssetID : String[1..1])	This operation defines a new mcmAssetID attribute for this MCMPhysicalEntity object. A single input parameter, called newAssetID (of type String), is defined. **[D42]** If an mcmAssetID attribute is not known, then the value of this attribute **SHOULD** be set to an empty string.
getMCMManufacture-Date() : Time-AndDate[1..1]	This operation returns the mcmManufactureDate attribute, in the form of a TimeAndDate datatype, of this MCMPhysicalEntity object. There are no input parameters to this operation. **[D43]** This attribute **SHOULD** have a complete and valid time and/or date. **[O15]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
setMCMManufacture-Date(manufacturerDate : TimeAndDate[1..1])	This operation defines a new mcmManufactureDate attribute, in the form of a TimeAndDate datatype, for this MCMPhysicalEntity object. A single input parameter, called manufacturerDate (of type TimeAndDate) is defined for this operation.
getMCMManufacturer() : String[1..1]	This operation returns the mcmManufacturer attribute of this MCMPhysicalEntity object. There are no input parameters to this operation. **[D44]** If the mcmManufacturer is not known or does not exist, then an empty string **SHOULD** be returned.
setMCMManufacturer(manufacturerName : String[1..1])	This operation defines a new mcmManufacturer attribute for this MCMPhysicalEntity object. A single string attribute, named manufacturerName, is defined. **[D45]** If the mcmManufacturer is not known or does not exist, then the value of this attribute **SHOULD** be set to an empty string.
getMCMSerialNumber() : String[1..1]	This operation returns the mcmSerialNumber attribute of this MCMPhysicalEntity object. There are no input parameters to this operation.

	[D46] If the serial number is not known or does not exist, then an empty string **SHOULD** be returned.
setMCMSerialNumber(in newSerialNumber : String[1..1])	This operation defines a new mcmSerialNumber attribute for this MCMPhysicalEntity object. A single string attribute, named newSerialNumber (of type String), is defined. **[D47]** If the serial number is not known, then the value of this attribute **SHOULD** be set to an empty string.
getMCMPhysicalEntityParent() : MCMPhysicalEntityComposite[1..1]	This operation returns the parent of this MCMPhysicalEntity object. **[D48]** If this MCMPhysicalEntity object has no parent, then a NULL MCMPhysicalEntityComposite **SHOULD** be returned.
setMCMPhysicalEntityParent(in newParent : MCMPhysicalEntityComposite[1..1])	This operation defines the parent of this MCMPhysicalEntity object. **[D49]** If this MCMPhysicalEntity object already has a parent, then an exception **SHOULD** be raised.
getMCMLocationListForPhyEntity() : MCMLocation[1..*]	This operation returns the set of MCMLocation objects that are associated with this particular MCMPhysicalEntity. This is done by following each instance of the MCMPhyEntityHasMCMLocation association, and taking into effect any semantics defined by the MCMPhyEntityHasMCMLocationDetail association class. This operation takes no input parameters. **[D50]** If no MCMLocation objects are associated with this particular MCMPhysicalEntity, then a NULL MCMLocation object **SHOULD** be returned.
setMCMLocationListForPhyEntity(in locationList : MCMLocation[1..*])	This operation defines a set of MCMLocation objects that are associated with this particular MCMPhysicalEntity. This operation takes a single input parameter, called locationList, which contains an array of one or more MCMLocation objects. This is done by instantiating an instance of the MCMPhyEntityHasMCMLocation association for each MCMLocation object in the input parameter, and then realizing that association with an instance of the MCMPhyEntityHasMCMLocationDetail association class. Note that this operation first deletes any existing associated MCMLocation objects (and their aggregations and association classes), and then instantiates a new set of MCMPhysicalEntity objects; in doing so, each MCMPhysicalEntity object is attached to this particular MCMLocation object by first, creating an instance of the MCMPhyEntityHasMCMLocation aggregation, and second, realizing that aggregation instance as an association class.

	[D51] Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMPhyEntityHasMCMLocationDetail class).
setMCMLocationPartialListForPhyEntity(in locPartialList : MCMLocation[1..*])	This operation defines a set of one or more MCMLocation objects that should be associated with this particular MCMPhysicalEntity object WITHOUT affecting any other existing contained MCMLocation objects or the objects that are contained in them. This operation takes a single input parameter, called locPartialList, which is an array of one or more MCMLocation objects. This operation creates a set of aggregations between this particular MCMPhysicalEntity object and the set of MCMLocation objects identified in the input parameter. **[D52]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMPhyEntityHasMCMLocationDetail class).
delMCMLocationForPhyEntity()	This operation deletes ALL instances of MCMLocation objects that are related to this particular MCMPhysicalEntity object. This operation first removes the association class, and second, removes the association, between this MCMPhysicalEntity object and each MCMLocation object that is attached to this MCMLocation object. This operation has no input parameters.
delMCMLocationPartialListForPhyEntity (in locationList : MCMLocation[1..*])	This operation deletes the set of instances of MCMLocation objects that are specified in the locationList parameter that are related to this particular MCMPhysicalEntity object. This operation first removes the association class, and second, removes the association, between this MCMPhysicalEntity object and each MCMLocation object that is specified in the locationList parameter. All other associations between this particular MCMPhysicalEntity object and other MCMLocation objects that are not specified in the locationList parameter are NOT affected.

Table 16. Operations for the MCMPhysicalEntity Class

At this time, the MCMPhysicalEntity class defines a single association that defines zero or more MCMLocations for a given MCMPhysicalEntity, called MCMPhyEntityHasMCMLocation. The multiplicity of this association is defined as 0..1 – 0..*. This means that this association is optional (i.e., the "0" part of the 0..1 cardinality). If this association is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMLocation objects can be associated with this particular MCMPhysicalEntity object. Note that the cardinality on the part side (MCMLocation) is 0..*; this enables an MCMPhysicalEntity object to be defined without having to define an associated MCMLocation object . The semantics of the MCMPhyEntityHasMCMLocation association is realized using an association class, called MCMPhyEntityHasMCMLocationDetail. This controls

the set of which MCMLocation objects can be associated with this particular MCMPhysicalEntity object.

The Policy Pattern may be used to control which specific MCMLocation objects are associated with a given MCMPhysicalEntity object for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

The MCMPhysicalEntity class also participates in a second aggregation, called MCMHasPhysicalEntity; see section 7.5.8.

7.5.7 MCMPhysicalEntityAtomic Class Definition

This is an abstract class, and specializes MCMPhysicalEntity. Each MCMPhysicalEntityAtomic has characteristics and behavior that are externally visible. Examples include a single building that is not related to other buildings, or a cable duct. It is abstract because it is intended to be subclassed.

[R22] This class represents stand-alone MCMPhysicalEntity objects (i.e., they **MUST NOT** contain another MCMPhysicalEntity object).

At this time, no attributes are defined for the MCMPhysicalEntityAtomic class.

At this time, no relationships are defined for the MCMPhysicalEntityAtomic class.

7.5.8 MCMPhysicalEntityComposite Class Definition

This is a concrete class, and specializes MCMPhysicalEntity. This class represents a set of related MCMPhysicalEntity objects that are organized into a tree structure. Its primary use is to collect other types of MCMPhysicalEntity objects (e.g., MCMPhysicalEntityAtomic and MCMPhysicalEntityComposite).

[O16] Each MCMPhysicalEntityComposite object **MAY** contain zero or more MCMPhysicalEntityAtomic and/or zero or more MCMPhysicalEntityAtomic objects.

For example, a Building may contain floors, wiring closets, and other physical entities that are of interest to the managed environment.

At this time, no attributes are defined for the MCMPhysicalEntityComposite class. Most attributes will likely be realized using relationships and/or operations. For example, a query to an instance of the MCMPhysicalEntityComposite class to provide its set of contained MCMPhysicalEntity objects (e.g., physical ports in one or more cards in one or more slots of a chassis) will be done by using class operations; the MCMPhysicalEntityComposite instance will query each of its contained MCMPhysicalEntity objects (which will in turn call their operations to acquire their MCMPhysicalEntity details), aggregate and organize the information, and provide that information in its operation response.

Table 17 defines the following operations for the MCMPhysicalEntityComposite class.

Operation Name	Description
getMCMPhysicalEntityChildList() : MCMPhysicalEntity[1..*]	This operation returns the set of all MCMPhysicalEntity objects that are contained in this specific MCMPhysicalEntityComposite object. There are no input parameters to this operation. This operation returns a list of zero or more MCMPhysicalEntity objects (i.e., the list is made up of MCMPhysicalEntityAtomic and/or MCMPhysicalEntityComposite objects). **[D53]** If this MCMPhysicalEntityComposite object has no children, then it **SHOULD** return a NULL MCMPhysicalEntity object.
setMCMPhysicalEntityChildList(childObjectList : MCMPhysicalEntity[1..*])	This operation defines a set of MCMPhysicalEntity objects that will be contained by this particular MCMPhysicalEntityComposite object. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMPhysicalEntity objects (i.e., one or more MCMPhysicalEntityAtomic and/or MCMPhysicalEntityComposite objects). This operation first creates an instance of the MCMHasPhysicalEntity aggregation between each MCMPhysicalEntity object in the childObjectList and this particular MCMPhysicalEntityComposite object. Note that this operation first deletes any existing contained MCMPhysicalEntity objects (and their aggregations and association classes), and then instantiates a new set of MCMPhysicalEntity objects; in doing so, each MCMPhysicalEntity object is contained within this particular MCMPhysicalEntityComposite object by first, creating an instance of the MCMHasPhysicalEntity aggregation, and second, realizing that aggregation instance as an association class. **[D54]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasPhysicalEntityDetail association class).
setMCMPhysicalEntityPartialChildList(childObjectList : MCMPhysicalEntity[1..*])	This operation defines a set of one or more MCMPhysicalEntity objects that should be contained within this particular MCMPhysicalEntityComposite object WITHOUT affecting any other existing contained MCMPhysicalEntity objects or the objects that are contained in them. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMPhysicalEntity objects. This operation creates a set of aggregations between this particular

	MCMPhysicalEntityComposite object and each of the MCMPhysicalEntity objects identified in the childObjectList. **[D55]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasPhysicalEntityDetail class).
delMCMPhysicalEntityChildList()	This operation deletes ALL contained MCMPhysicalEntity objects of this particular MCMPhysicalEntityComposite object. This has the effect of first, removing the association class, and second, removing the aggregation, between this MCMPhysicalEntityComposite object and each MCMPhysicalEntity object that is contained in this MCMPhysicalEntityComposite object. This operation has no input parameters.
delMCMPhysicalEntityPartialChildList (in childObjectList : MCMPhysicalEntity[1..*])	This operation deletes a set of MCMPhysicalEntity objects from this particular MCMPhysicalEntityComposite object. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMPhysicalEntity objects. This has the effect of first, removing the association class and second, removing the aggregation, between each MCMPhysicalEntity object specified in the input parameter and this MCMPhysicalEntityComposite object. Note that all other aggregations between this MCMPhysicalEntityComposite and other MCMPhysicalEntity objects that are not identified in the input parameter are NOT affected.

Table 17. Operations of the MCMPhysicalEntityComposite Class

The MCMPhysicalEntityComposite class defines a single aggregation, called MCMHasPhysicalEntity. This aggregation is used to define the set of MCMPhysicalEntity objects that are contained within this particular MCMPhysicalEntityComposite object. Its multiplicity is defined to be 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMPhysicalEntity objects can be aggregated by this particular MCMPhysicalEntityComposite object. Note that the cardinality on the part side (MCMPhysicalEntity) is 0..*; this enables an MCMPhysicalEntityComposite object to be defined without having to define an MCMPhysicalEntity object for it to aggregate.

The semantics of the MCMHasPhysicalEntity aggregation is realized using an association class, called MCMHasPhysicalEntityDetail. This enables the semantics of the MCMHasPhysicalEntity aggregation to be realized using the attributes, operations, and relationships of the MCMHasPhysicalEntityDetail association class. The Policy Pattern may be used to control which specific MCMPhysicalEntity objects are contained within a given MCMPhysicalEntityComposite object for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.6 MCMDomain Class Hierarchy

The MCMDomain class has a single subclass, as shown in Figure 10.

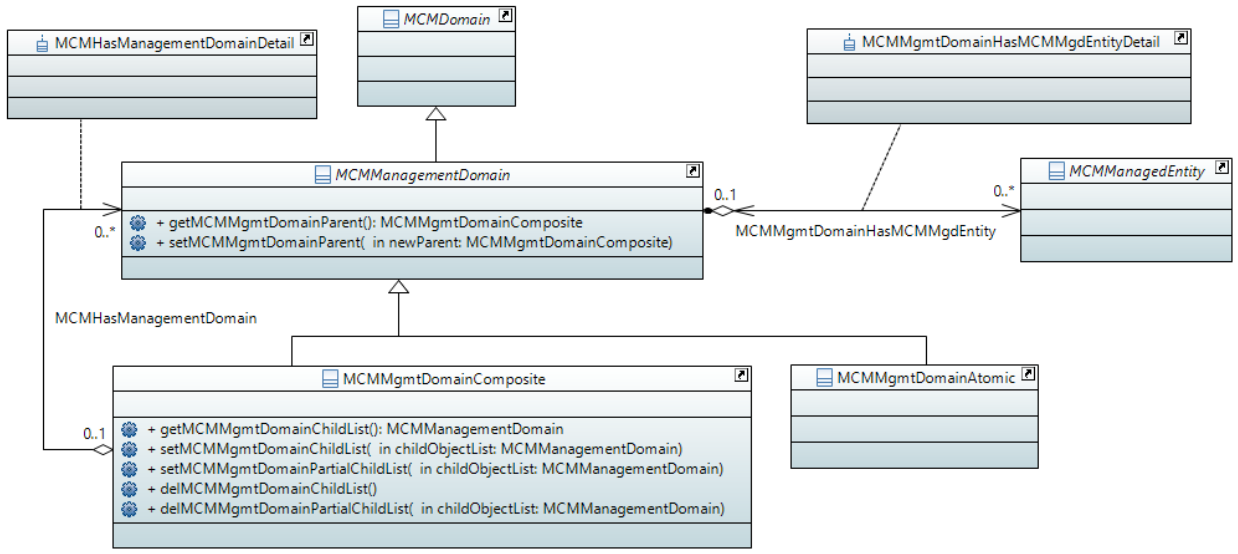

Figure 10. MCMDomain Subclasses

Table 18 defines the purpose of this hierarchy, and aligns them to MEF 55 [1]. The purpose of the MCMDomain hierarchy is to model the major different types of Entities that are inherently manageable using digital means, and which also are of interest to the managed environment. Examples include interfaces of a network device, protocols, policy rules, and behavior of an object.

Name of Class	Function	Relation to MEF 55
MCMDomain	Defines a collection of MCMEntities that share a common purpose. In addition, each constituent MCMEntity in an MCMDomain is both uniquely addressable and uniquely identifiable within that MCMDomain	Models the generic concept of an administrative domain.
MCMManagement-Domain	An MCMManagementDomain is used to contain MCMManagedEntities. It refines the notion of an MCMDomain by adding three important behavioral features: 1) it defines a set of administrators that govern the MCMManagedEntities that it contains; 2) it defines a set of applications that are responsible for different governance operations, such as monitoring, configuration, and so forth; 3) it defines a common set of management mechanisms, such as policy rules, that	This links the Policy Driven Orchestration work to MEF55.

	are used to govern the behavior of MCMManagedEntities contained in the MCMManagementDomain.	
MCMMgmtDomain Atomic	Represents MCMManagementDomains that are modeled as a single, stand-alone, manageable object.	The most common type of MCMManagementDomain.
MCMMgmt DomainComposite	Represents MCMManagementDomains that are modeled as a hierarchy of manageable objects. This produces three objects: the Composite MCMManagementDomains, the set of constituent component MCMManagementDomains, and the combination of these.	Accommodates nested Orders and Orders that form a tree-like hierarchy.

Table 18. Functions of the MCMDomain Class and its Subclasses

7.6.1 MCMDomain Class Definitiion

This is an abstract class, and specializes MCMEntity. An MCMDomain is a collection of MCMEntities that share a common purpose. In addition, each constituent MCMEntity in an MCMDomain is both uniquely addressable and uniquely identifiable within that MCMDomain.

At this time, no attributes are defined for the MCMDomain class.

At this time, no operations are defined for the MCMDomain class.

At this time, no relationships are defined for the MCMDomain class.

7.6.2 MCMManagementDomain Class Definition

This is a concrete class, and specializes MCMDomain. Unlike an MCMDomain, an MCMManagementDomain is used to contain MCMManagedEntities. Hence, it refines the notion of a Domain by adding several important behavioral features, as specified in the following requirements:

[R23] First, each MCMManagedEntity that is contained in an MCMManagementDomain **MUST** be uniquely identifiable for management purposes.

[D56] Second, an MCMManagementDomain **SHOULD** define a set of administrators that govern the ManagedEntities that it contains

[O17] Third, an administrator **MAY** be restricted to execute a subset of operations for a given MCMManagementDomain.

[D57] Fourth, an MCMManagementDomain **SHOULD** define a set of applications that are responsible for different governance operations, such as monitoring and configuration.

[O18] Fifth, different applications **MAY** be responsible for different governance operations (e.g., monitoring and configuration may be done by the same or different applications).

[D58] Sixth, an MCMManagementDomain **SHOULD** define a common set of management mechanisms, such as policy rules, that are used to govern the behavior of ManagedEntities contained in the ManagementDomain.

This set of features combine to enable an MCMManagementDomain to be administered as a single unit.

The above concepts are represented as follows:

- Unique identifiability is satisfied by the use of objectIDs (defined in MCMRootEntity, see section 7.2)
- Administrators are defined as a type of MCMPartyRole (see section 7.11.2.2); since an MCMPartyRole is a type of MetaData, it can be associated with an MCMManagementDomain through the use of the MCMEntityHasMCMMetaData aggregation (see section 7.4)
- Governance operations are a specific type of MCMInternalService (see section 7.8.5.6)
- Policies are defined in the MEF Policy Driven Orchestration project.

The constraint for having an MCMDomain contain MCMManagedEntities, and not simply MCMEntities, is realized using the MCMMgmtDomainHasMCMManagedEntity aggregation. This aggregation is realized using an association class (called MCMMgmtDomainHasMCMManagedEntityDetail), whose attributes are controlled by a set of policies.

[O19] This association **MAY** also be further refined using OCL.

Currently, no attributes are defined for this class.

Table 19 defines the operations for the MCMManagementDomain class.

Operation Name	Description
getMCMMgmtDomainParent() : MCMMgmtDomainCompositе[1..1]	This operation returns the parent of this MCMDomain object. **[D59]** If this MCMDomain object has no parent, then a NULL MCMDomainComposite object **SHOULD** be returned.
setMCMMgmtDomainParent(in newParent : MCMMgmtDomainCompositе[1..1])	This operation defines the parent of this MCMDomain object. **[D60]** If this MCMDomain object already has a parent, then an exception **SHOULD** be raised.

Table 19. Operations of the MCMManagementDomain Class

At this time, a single aggregation is defined for the MCMManagementDomain class. This aggregation is named MCMMgmtDomainHasMCMMgdEntity, and defines the set of MCMManagedEntities that are contained in this particular MCMManagementDomain. The multiplicity of this aggregation is 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMManagedEntity objects can be aggregated by this particular MCMManagementDomain object. Note that the cardinality on the part side (MCMManagedEntity) is 0..*; this enables an MCMManagementDomain object to be defined without having to define an associated MCMManagedEntity object for it to aggregate. Since there are different types of MCMManagementDomain objects as well as different types of MCMManagedEntity objects that can be contained within a given MCMManagementDomain object, the MCMMgmtDomainHasMCMManagedEntity aggregation is realized using an association class, called MCMMgmtDomainHasMCMMgdEntityDetail. This enables the semantics of the MCMMgmtDomainHasMCMMgdEntity aggregation to be realized using the attributes, operations, and relationships of the MCMMgmtDomainHasMCMMgdEntityDetail association class.

The Policy Pattern may be used to control which type of MCMManagedEntity objects are contained in a particular MCMManagementDomain object. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

This class also participates in a second aggregation, called MCMHasManagementDomain; this is defined in section 7.6.

7.6.3 MCMMgmtDomainAtomic Class Definition

This is a concrete class, and specializes MCMManagementDomain. Each MCMMgmtDomainAtomic has characteristics and behavior that is externally visible.

[R24] This class represents stand-alone MCMManagementDomain objects (i.e., they **MUST NOT** contain another MCMMgmtDomain object).

At this time, no attributes are defined for the MCMMgmtDomainAtomic class.

At this time, no operations are defined for the MCMMgmtDomainAtomic class.

At this time, no relationships are defined for the MCMMgmtDomainAtomic class.

7.6.4 MCMMgmtDomainComposite Class Definition

This is a concrete class, and specializes MCMManagementDomain. This class represents a set of related MCMManagementDomain objects that are organized into a tree structure.

[O20] Each MCMMgmtDomainComposite **MAY** contain zero or more MCMMgmtDomainAtomic and/or zero or more MCMMgmtDomainComposite objects.

At this time, no attributes are defined for the MCMManagementDomainComposite class.

Table 20 defines the operations for the MCMManagementDomainComposite class.

Operation Name	Description
getMCMMgmtDomainChild-List() : MCMManagement-Domain [1..*]	This operation returns the set of all MCMManagementDomain objects that are contained in this specific MCMManagement-DomainComposite object. There are no input parameters to this operation. This operation returns a list of zero or more MCMMan-agementDomain objects (i.e., the list is made up of MCMManage-mentDomainAtomic and/or MCMManagementDomainComposite objects). **[D61]** If this MCMManagementDomainComposite object has no child objects, then a NULL MCMManage-mentDomainComposite object **SHOULD** be re-turned.
setMCMMgmtDomainChild-List (childObjectList : MCMManagementDomain [1..*])	This operation defines a set of MCMManagementDomain objects that will be contained by this particular MCMMgmtDomainCom-posite object. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMManagementDomain objects (i.e., one or more MCMMgmtDomainAtomic and/or MCMMgmtDomainComposite objects). This has the effect of creating an instance of the MCMHasManagementDomain aggregation between each MCMManagementDomain object in the childObjectList and this particular MCMMgmtDomainComposite object. Note that this operation first deletes any existing contained MCMManagement-Domain objects (and their aggregations and association classes), and then instantiates a new set of MCMManagementDomain ob-jects; in doing so, each MCMManagementDomain object is con-tained within this particular MCMMgmtDomainComposite object by first, creating an instance of the MCMHasManagementDomain aggregation, and second, realizing that aggregation instance as an association class. **[D62]** Each created aggregation **SHOULD** have an associa-tion class (i.e., an instance of the MCMHasManage-mentDomainDetail association class).
setMCMMgmtDomainPar-tialChildList (childObjectList : MCMManagementDomain [1..*])	This operation defines a set of one or more MCMManagement-Domain objects that are contained within this particular MCMManagementDomainComposite object WITHOUT affecting any other existing contained MCMManagementDomain objects or the objects that are contained in them. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMManagementDomain objects. This has the ef-fect of creating a set of aggregations between this particular MCMManagementDomainComposite object and each of the

	MCMManagementDomain objects identified in the childObject-List. **[D63]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasManagementDomainDetail class).
delMCMMgmtDomainChild-List ()	This operation deletes ALL contained MCMManagementDomain objects of this particular MCMManagementDomainComposite object. This has the effect of first, removing the association class, and second, removing the aggregation, between this MCMManagementDomainComposite object and each MCMManagement-Domain object that is contained in this MCMManagement-DomainComposite object. This operation has no input parameters.
delMCMMgmtDomainPartialChildList (in childObjectList : MCMManagement-Domain[1..*])	This operation deletes a set of MCMManagementDomain objects from this particular MCMManagementDomainComposite object. This operation takes a single input parameter, called childObject-List, which is an array of one or more MCMManagementDomain objects. This has the effect of first, removing the association class and second, removing the aggregation, between each MCMManagementDomain object specified in the input parameter and this MCMManagementDomainComposite object. Note that all other aggregations between this MCMManagementDomainComposite and other MCMManagementDomain objects that are not identified in the input parameter are NOT affected.

Table 20. Operations of the MCMManagementDomainComposite Class

The MCMHasManagementDomainComposite class defines a single aggregation, called MCMHasManagementDomain. This aggregation is used to define the set of MCMManagement-Domains that are contained within this particular MCMMgmtDomainComposite. Its multiplicity is defined to be 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMManagementDomain objects can be aggregated by this particular MCMManagementDomainComposite object. Note that the cardinality on the part side (MCMManagement-Domain) is 0..*; this enables an MCMManagementDomainComposite object to be defined without having to define an associated MCMManagementDomain object for it to aggregate.

The semantics of the MCMHasManagementDomain aggregation is realized using an association class, called MCMHasManagementDomainDetail. This enables the semantics of the MCMHas-ManagementDomain aggregation to be realized using the attributes, operations, and relationships of the MCMHasManagementDomainDetail association class.

The Policy Pattern may be used to control which specific MCMManagementDomain objects are contained within a given MCMManagementDomainComposite object for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.7 MCMBusinessObject Class Hierarchy

The MCMBusinessObject class has two subclasses, as shown in Figure 11.

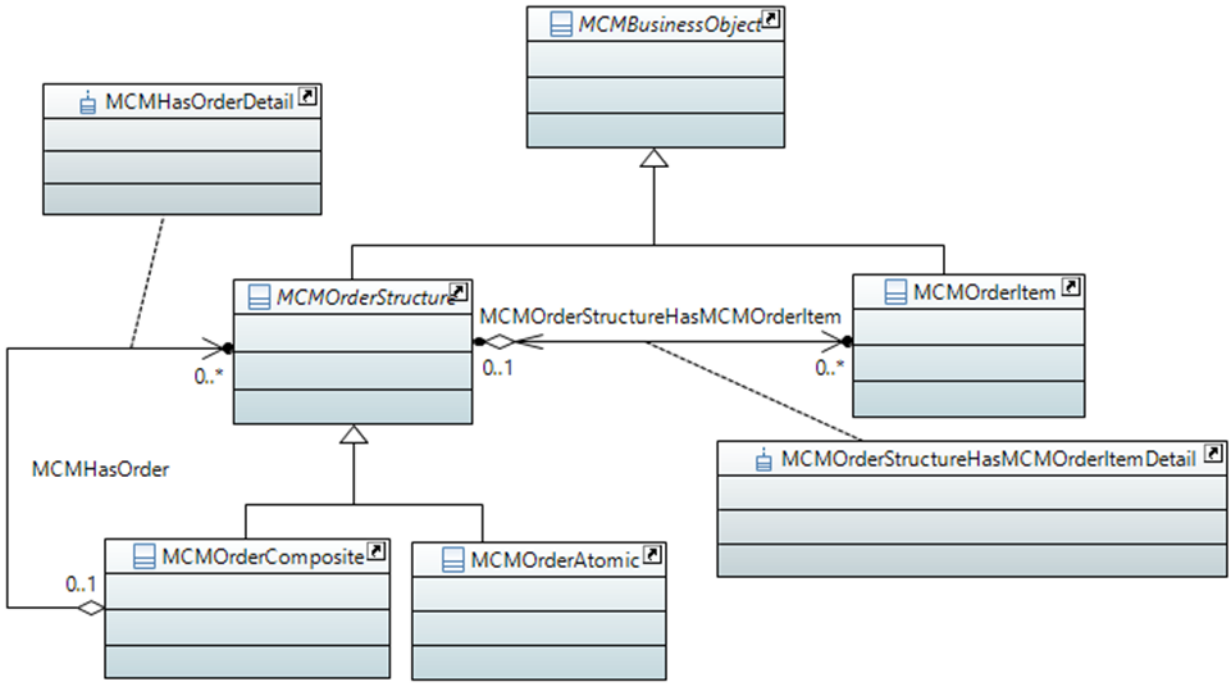

Figure 11. MCMBusinessObject Subclasses

MCMBusinessObject is a subclass of MCMEntity, and is a sibling of MCMManagedEntity. The MCM models business objects differently than other types of managed entities, because: (1) their lifecycle is different, and (2) their semantics are different. This class is the superclass of concepts such as Orders and TroubleTickets.

Table 21 defines the functions of the MCMBusinessObject class and its subclasses, and relates them to MEF55.

Name of Class	Function	Relation to MEF 55
MCMBusinessObject	Defines the abstract concept of business objects that are types of MCMEntities, but not types of MCMManagedEntities. Examples include Order, TroubleTicket, and Report.	Required by all MEF55 functional components that interact with Business Applications and/or Customers.
MCMOrderStructure	Defines the abstract concept of an Order, including common attributes, operations, and relationships.	Required by all MEF55 functional components that interact with Orders.
MCMOrderAtomic	Represents Orders that are modeled as a single, stand-alone, manageable object.	The most common type of Order.
MCMOrderComposite	Represents Orders that are modeled as a hierarchy of Orders. This produces three objects: the Composite Order, the set of constituent Atomic Orders, and the combination of these.	Accommodates nested Orders and Orders that form a tree-like hierarchy.
MCMOrderStructure HasMCMOrderItemDetail	An association, realized as an association class, that defines the semantics of an MCMOrderStructure having an MCMOrderItem.	Provides management control of attaching and removing parts of a Composite Order.
MCMOrderItem	Represents a set of MCMEntities that are contained in a particular MCMOrderAtomic or an MCMOrderComposite object.	Represents parts of an Order

Table 21. Functions of the MCMBusinessObject and its Subclasses

7.7.1 MCMBusinessObject Class Definition

This is an abstract class, and specializes MCMEntity. It represents business objects that are produced by the business but are not managed in the way that MCMManagedEntity objects are. Examples include Orders, TroubleTickets, and Reports.

Note that concepts like the set of MCMPartyRoles that interact with this MCMBusinessObject, and the time period in which this MCMBusinessObject is valid, are realized as relationships, not attributes. More specifically, the former is provided by MCMEntityHasMCMMetaData (see section 7.4), since MCMBusinessObject is a subclass of MCMEntity and therefore inherits this aggregation. The latter is already defined in MCMMetaData (see section 7.11).

Table 22 defines the attributes of the MCMBusinessObject class. Most attributes will likely be realized using relationships and/or operations. For example, concepts like the Buyer and Seller object identifiers, along with Buyer order, implementation, and technical contacts [15] will be defined using a combination of relationships and operations.

Attribute Name	Mandatory?	Description
mcmBusinessPurpose : String[0..1]	NO	This is a string attribute. It contains a textual description of the business purpose of this MCMBusinessObject. **[D64]** If an object does not have a value for the mcmCommonName attribute, then an empty string **SHOULD** be used.
mcmBizObjCreationDate : TimeAndDate[0..1]	NO	This is a TimeAndDate attribute, and contains the date and time of the manufacturing of this object. **[D65]** This attribute **SHOULD** have a complete and valid time and/or date. **[O21]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.

Table 22. Attributes of the MCMBusinessObject Class

Table 23 defines the operations of the MCMBusinessObject class.

Operation Name	Description
getMCMBusinessPurpose() : String[1..1]	This operation returns the mcmBusinessPurpose textual attribute for this particular MCMBusinessObject. There are no input parameters to this operation. **[D66]** If a business purpose is not defined, then an empty string **SHOULD** be returned.
setMCMBusinessPurpose(in bizPurpose: String[1..1])	This operation sets the mcmBusinessPurpose textual attribute for this particular MCMBusinessObject. There is a single input parameter, of type String, which contains the new text of the business purpose. **[D67]** If a business purpose is not defined, then an empty string **SHOULD** be used.
getMCMDateCreated() : TimeAndDate[1..1]	This is a TimeAndDate attribute, and contains the date and time that this object was created. **[D68]** This attribute **SHOULD** have a complete and valid time and/or date. **[O22]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.

setMCMDateCreated (in newCreationDate: Time-AndDate[1..1])	This is a TimeAndDate attribute, and contains the date and time that this object was created.

Table 23. Operations of the MCMBusinessObject Class

At this time, no relationships are defined for the MCMBusinessObject class.

7.7.2 MCMOrderStructure Class Definition

This is an abstract class, and specializes MCMBusinessObject. The purpose of this class is to enable the composite pattern to be used to define stand-alone and nested orders (represented by MCMOrderAtomic and MCMOrderComposite, respectively). It represents a written commitment to procure an MCMProduct from a Seller by a Buyer. The Seller and Buyer may be MCMParty or (preferably) MCMPartyRole objects. An MCMOrderStructure object (i.e., an MCMOrderAtomic or an MCMOrderComposite) may contain zero or more MCMOrderItem objects.

Table 24 defines the attributes of the MCMOrderStructure class. These are compliant with the use cases in [15].

Attribute Name	Mandatory?	Description
mcmIsPriorityExpedited : Boolean[1..1]	YES	This is a Boolean attribute that indicates if this Order is being expedited or not. The default value of this attribute is FALSE. **[R25]** IFF the value of this Boolean attribute is FALSE, then the implementation **MUST** ignore the value of the mcmOrderPriority class attribute. **[R26]** IFF the value of this Boolean attribute is TRUE, the the implementation **MUST** use the value of the mcmOrderPriority class attribute as the priority of this Order (if any).
mcmOrderCompleteActual : [0..1]	NO	This is a TimeAndDate attribute. It defines the date and time that this order was completed by the Seller (e.g., MCMServiceProvider or MCMPartner). It is optional to facilitate the case where an order has been issued but has not been complete. It is based on [15]. **[D69]** This attribute **SHOULD** have a complete and valid time and/or date.

		[O23] The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
mcmOrderComplete-Request : Time-AndDate[1..1]	YES	This is a TimeAndDate attribute. It defines the date and time by which the Buyer (e.g., MCMCustomer or MCMPartner) requested that this order be completed. It is based on [15]. **[D70]** This attribute **SHOULD** have a complete and valid time and/or date. **[O24]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
mcmOrderCreateDate : TimeAndDate[1..1]	YES	This is a TimeAndDate attribute. It defines the date and time that this order was created. It is based on [15]. **[D71]** This attribute **SHOULD** have a complete and valid time and/or date. **[O25]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
mcmOrderDesiredResponse : OrderDesiredResponse[1..1]	YES	This enumeration defines the desired response that the Buyer wishes to receive from the Seller. It is based on [15]. The values are defined in the OrderDesiredResponse enumeration, and include: ERROR INIT CONFIRMATION AND ENGINEERING DESIGN CONFIRMATION ONLY NONE
mcmOrderID : String[1..1]	YES	This is a string attribute. It contains a unique identifier for the order that is generated by the Seller when the order is initially accepted. **[R27]** This attribute **MUST NOT** be used as a naming attribute (i.e., to uniquely identify an instance of the object). **[R28]** The mcmOrderID **MUST** be provided by the Seller on all response messages.

		[R29] The mcmOrderID **MUST** remain the same for the life of the Order. [15]
mcmOrderPriority : Integer[0..1]	NO	This is a non-negative Integer attribute, which indicates the relative priority of this Order compared to other Orders between this particular Buyer and Seller. The meaning of this attribute is subject to agreement between the Buyer and the Seller. [15] **[R30]** IFF the value of the mcmIsPriorityExpedited class attribute is FALSE, then the implementation **MUST** ignore the value of this class attribute. **[R31]** IFF the value of the mcmIsPriorityExpedited class attribute is TRUE, then the implementation **MUST** assign the priority of this Order to the value of this class attribute. **[O26]** If two or more Orders have the same value of this attribute, then the implementation **MAY** process them in any order, as long as each is processed in appropriate numerical order of the mcmOrderPriority attribute value with respect to all other Orders. **[R32]** If an Order does not have a value for this attribute, then the implementation **MUST** process it (and any other similar Orders) after all Orders that do have a valid value for this attribute.
mcmOrderStartRequest : TimeAndDate[1..1]	YES	This is a TimeAndDate attribute. It defines the date and time that the Buyer (e.g., MCMCustomer or MCMPartner) would like work on this order to start. It is based on [15]. **[D72]** This attribute **SHOULD** have a complete and valid time and/or date. **[O27]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
mcmOrderType : OrderType[1..1]	YES	This is a mandatory enumeration, which defines the type of order that this instance is. It is based on the use cases in [15], where it is called ORDER

		ACTIVITY. The values are defined in the Order-Type enumeration, and include: ERROR INIT CREATE INSTALL CHANGE DISCONNECT QUERY AMEND CANCEL NOTIFY COMPLETE NO_CHANGE

Table 24. Attributes of the MCMOrderStructure Class

Table 25 defines the operations of the MCMOrderStructure class.

Operation Name	Description
getMCMIsPriorityExpedited() : Boolean[1..1]	This operation retrieves the value of the mcmIsPriorityExpedited class attribute. The default value of this attribute is FALSE. IFF the value of this Boolean attribute is FALSE, then the mcmOrderPriority class attribute is irrelevant. Otherwise, the priority of this Order (if any) is defined by the value of the mcmOrderPriority attribute. This operation takes no input parameters, and returns a Boolean. It is based on [15]. **[R33]** IFF the value of the mcmIsPriorityExpedited class attribute is FALSE, then the implementation **MUST** ignore the value of the mcmIsOrderPriority class attribute. **[R34]** IFF the value of the mcmIsPriorityExpedited class attribute is TRUE, then the implementation **MUST** assign the priority of this Order to the value of the mcmIsOrderPriority class attribute.
setMCMIsPriorityExpedited(in isExpedited : Boolean[1..1])	This operation defines the value of the mcmIsPriorityExpedited class attribute. The default value of this attribute is FALSE. IFF the value of this Boolean attribute is FALSE, then the mcmOrderPriority class attribute is irrelevant. Otherwise, the priority of this Order (if any) is defined by the value of the

	mcmOrderPriority attribute. This operation takes a single input parameter, called isExpedited, which is a Boolean. It is based on [15]. **[R35]** IFF the value of the mcmIsPriorityExpedited class attribute is FALSE, then the implementation **MUST** ignore the value of the mcmIsOrderPriority class attribute. **[R36]** IFF the value of the mcmIsPriorityExpedited class attribute is TRUE, then the implementation **MUST** assign the priority of this Order to the value of the mcmIsOrderPriority class attribute.
getMCMOrderCompleteActual() : TimeAndDate[1..1]	This operation returns the time and date (as a TimeAndDate data type) that this order was completed by the Seller (e.g., MCMServiceProvider or MCMPartner). There are no input parameters. It is based on [15]. **[D73]** This attribute **SHOULD** have a complete and valid time and/or date. **[O28]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
setMCMOrderCompleteActual(in dateCompleted : TimeAndDate[1..1])	This operation defines the time and date (as a TimeAndDate data type) that this order was completed by the Seller (e.g., MCMServiceProvider or MCMPartner). There is a single input parameter, called dateCompleted (of data type TimeAndDate) that contains the new time and date information. It is based on [15].
getMCMOrderCompleteRequest() : TimeAndDate[1..1]	This operation returns the time and date (as a TimeAndDate data type) that the Buyer (e.g., MCMCustomer or MCMPartner) requested that this order be completed. There are no input parameters. It is based on [14]. **[D74]** This attribute **SHOULD** have a complete and valid time and/or date. **[O29]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
setMCMOrderCompleteRequest(in dateRequested : TimeAndDate[1..1])	This operation defines the time and date (as a TimeAndDate data type) that this order was requested to be completed by the Buyer (e.g., MCMCustomer or MCMPartner). There is a single input parameter, called dateRequested (of data type TimeAndDate) that contains the new date and time information. It is based on [15].
getMCMOrderCreateDate() : TimeAndDate[1..1]	This operation returns the time and date (as a TimeAndDate data type) that this order was created by the Buyer. There are no input parameters. It is based on [15].

	[D75] This attribute **SHOULD** have a complete and valid time and/or date. **[O30]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
setMCMOrderCreateDate(in dateCreated : TimeAndDate[1..1])	This operation defines the time and date (as a TimeAndDate data type) that this order was requested to be completed by the Buyer (e.g., MCMCustomer or MCMPartner). There is a single input parameter, called dateRequested (of data type TimeAndDate) that contains the new time and date information. It is based on [15].
getMCMOrderDesiredResponse() : OrderDesiredResponse[1..1]	This operation retrieves the desired response that the Buyer wishes to receive from the Seller. This operation takes no input parameters, and returns a value from the OrderDesiredResponse enumeration. It is based on [15].
setMCMOrderDesiredResponse(in newResponse : OrderDesiredResponse[1..1])	This operation defines desired response that the Buyer wishes to receive from the Seller. This operation takes a single input parameter, the OrderDesiredResponse enumeration, which contains a value describing the desired response from the Seller to the Buyer. It is based on [15].
getMCMOrderID() : String[1..1]	This operation retrieves the order ID. It is based on [15]. **[R37]** This attribute **MUST NOT** be used as a naming attribute (i.e., to uniquely identify an instance of the object). **[R38]** This **MUST** be generated by the Seller when the order was initially accepted. **[R39]** This attribute **MUST NOT** change through the life of the Order.
setMCMOrderID(in newOrderID : String[1..1])	This operation defines a new order ID. This operation takes a single input parameter, called newOrderID, of data type string. It is based on [15]. **[R40]** This attribute **MUST NOT** be used as a naming attribute (i.e., to uniquely identify an instance of the object). **[R41]** This **MUST** be generated by the Seller when the order was initially accepted. **[R42]** This attribute **MUST NOT** change through the life of the Order.
getMCMOrderPriority() : Integer[1..1]	This operation retrieves the value of the mcmOrderPriority class attribute, which indicates the relative priority of this Order compared to other Orders between this particular Buyer and Seller. Orders with a higher priority execute before Orders

	with a lower priority; Orders of equal priority may execute in any order. Note that this value is only relevant if the value of the mcmIsPriorityExpedited class attribute is TRUE. This operation takes no input parameters, and returns an Integer. It is based on [15]. **[R43]** IFF the value of the mcmIsPriorityExpedited class attribute is FALSE, then the implementation **MUST** ignore the value of this class attribute. **[R44]** IFF the value of the mcmIsPriorityExpedited class attribute is TRUE, then the implementation **MUST** assign the priority of this Order to the value of this class attribute. **[O31]** If two or more Orders have the same value of this attribute, then the implementation **MAY** process them in any order, as long as each is processed in appropriate numerical order of the mcmOrderPriority attribute value with respect to all other Orders. **[R45]** If an Order does not have a value for this attribute, then the implementation **MUST** process it (and any other similar Orders) after all Orders that do have a valid value for this attribute.
setMCMOrderPriority(in newPriority : Integer[1..1])	This operation defines the value of the mcmOrderPriority class attribute. IFF the value of the mcmIsPriorityExpedited class attribute is FALSE, then the value of this class attribute is irrelevant. Otherwise, the priority of this Order (if any) is defined by the value of this attribute. This operation takes a single input parameter, called newPriority, which is an Integer. It is based on [15]. **[R46]** IFF the value of the mcmIsPriorityExpedited class attribute is FALSE, then the implementation **MUST** ignore the value of this class attribute. **[R47]** IFF the value of the mcmIsPriorityExpedited class attribute is TRUE, then the implementation **MUST** assign the priority of this Order to the value of this class attribute. **[O32]** If two or more Orders have the same value of this attribute, then the implementation **MAY** process them in any order, as long as each is processed in appropriate numerical order of the mcmOrderPriority attribute value with respect to all other Orders.

	[R48] If an Order does not have a value for this attribute, then the implementation **MUST** process it (and any other similar Orders) after all Orders that do have a valid value for this attribute.
getMCMOrderStartRequest() : TimeAndDate[1..1]	This operation returns the time and date (as a TimeAndDate data type) that the Buyer (e.g., MCMCustomer or MCMPartner) would like work on this order to start. It is based on [15]. **[D76]** This attribute **SHOULD** have a complete and valid time and/or date. **[O33]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
setMCMOrderStartRequest(in newStartTime : TimeAndDate[1..1])	This operation defines the time and date (as a TimeAndDate data type) that work on this order was requested to be started by the Buyer (e.g., MCMCustomer or MCMPartner). There is a single input parameter, called newStartTime (of data type TimeAndDate) that contains the new date and time information that work on this order should start. It is based on [15].
getMCMOrderType : OrderType[1..1]	This operation retrieves the value of the mcmOrderType class attribute, which indicates the type of Order that this instance is. The type of work to be done in this order is defined by an enumeration, called OrderType. This operation takes no input parameters, and returns a literal from the OrderType enumeration. It is based on [15].
setMCMOrderType(in newType : OrderType[1..1])	This operation defines the value of the mcmOrderType class attribute, which indicates the type of Order that this instance is. The type of work to be done in this order is defined by an enumeration, called OrderType. This operation takes a single input parameter, called newType, which is an OrderType enumeration that defines the different types of work that this Order can do. It is based on [15].
getMCMOrderStructureParent() : MCMOrderComposite[1..1]	This operation returns the parent of this MCMOrderStructure object. This operation takes no input parameters. **[D77]** If this MCMOrderStructure object has no parent, then a NULL MCMOrderComposite object **SHOULD** be returned.
setMCMOrderStructureParent(in newParent : MCMOrderComposite[1..1])	This operation defines the parent of this MCMOrderStructure object. The parent is defined in the input parameter, called newParent, and is of type MCMOrderComposite. **[D78]** If this MCMOrderStructure object already has a parent, then an exception **SHOULD** be raised.
getMCMOrderItemList() : MCMOrderItem[1..*]	This operation returns the set of MCMOrderItem objects that are currently contained by this MCMOrderStructure object.

	The return value is an array of one or more objects of type MCMOrderItem. This operation follows all instances of the MCMOrderStructureHasMCMOrderItem aggregation (i.e., from this MCMOrderStructure object to each MCMOrderItem object that it contains), and returns the associated MCMOrderItem objects as an array. **[D79]** If this object does not have any attached MCMOrderItem objects, then a NULL MCMOrderItem object **SHOULD** be returned.
setMCMOrderItemList (in newOrderItem: MCMOrderItem [1..*])	This operation defines the complete set of MCMOrderItem objects that will be contained by this MCMOrderStructure object. This operation takes a single input parameter, called newOrderItem, which is an array of one or more MCMOrderItem objects. This operation creates a set of aggregations between this particular MCMOrderStructure object and the set of MCMOrderItem objects identified in the input parameter. Note that this operation first deletes any existing attached MCMOrderItem objects (and their aggregations and association classes), and then instantiates a new set of MCMOrderItem objects; in doing so, each MCMOrderItem object is attached to this particular MCMOrderStructure object by first, creating an instance of the MCMOrderStructureHasMCMOrderItem aggregation, and second, realizing that aggregation instance as an association class. **[D80]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMOrderStructureHasMCMOrderItemDetail class).
setMCMOrderItemPartialList (in newOrderItemPartialList: MCMOrderItem [1..*])	This operation defines a set of one or more MCMOrderItem objects that should be attached to this particular MCMOrderStructure object WITHOUT affecting any other existing contained MCMOrderItem objects or the objects that are contained in them. This operation takes a single input parameter, called newOrderItemPartialList, which is an array of one or more MCMOrderItem objects. This operation creates a set of aggregations between this particular MCMOrderStructure object and the set of MCMOrderItem objects identified in the input parameter. **[D81]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMOrderStructureHasMCMOrderItemDetail class).
delMCMOrderItemList ()	This operation deletes ALL instances of attached MCMOrderItem objects for this particular MCMOrderStructure object. This operation first removes the association class, and second,

	removes the aggregation, between this MCMOrderStructure object and each MCMOrderItem object that is attached to this MCMOrderStructure object. This operation has no input parameters.
delMCMOrderItemPartialList (in orderItemPartialList : MCMOrderItem[1..*])	This operation deletes a set of MCMOrderItem objects from this particular MCMOrderStructure object. This operation takes a single input parameter, called orderItemPartialList, which is an array of one or more MCMOrderItem objects. This operation first, removes the association class and second, removes the aggregation, between each MCMOrderItem object specified in the input parameter and this MCMOrderStructure object. Note that all other aggregations between this MCMOrderStructure object and other MCMOrderItem objects that are not specified in the input parameter are NOT affected.

Table 25. Operations of the MCMOrderStructure Class

The MCMOrderStructureHasMCMOrderItem aggregation defines the set of MCMOrderItems that this MCMOrderStructure (i.e., an MCMOrderAtomic or an MCMOrderComposite) can contain. All subclasses of MCMOrderStructure inherit this relationship. The multiplicity of this aggregation is 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMOrderItem objects can be aggregated by this particular MCMOrderStructure object. Note that the cardinality on the part side (MCMOrderItem) is 0..*; this enables an MCMOrderStructure object to be defined without having to define an associated MCMOrderItem object for it to aggregate. The semantics of this aggregation are defined by the MCMOrderStructureHasMCMOrderItemDetail association class. This enables a particular set of MCMOrderItems to be attached to a given MCMOrderStructure. The Policy Pattern may be used to control which specific MCMOrderItem objects are attached to a given MCMOrderStructure object for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

This class participates in a second aggregation, called the MCMHasOrder aggregation; see section 7.7.4. The Policy Pattern may be used to control which specific MCMOrderStructure objects are contained within a given MCMOrderComposite for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. The Policy Pattern may be used to control which specific MCMOrderStructure objects are contained within a given MCMOrderComposite for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.7.3 MCMOrderAtomic Class Definition

This is a concrete class, and specializes MCMOrderStructure.

[R49] This class represents stand-alone MCMOrderStructure objects (i.e., they **MUST NOT** contain another MCMOrderStructure object).

[O34] Each MCMOrderAtomic **MAY** contain one or more MCMOrderItems.

At this time, no attributes are defined for the MCMOrderAtomic class.

At this time, no operations are defined for the MCMOrderAtomic class.

At this time, no relationships are defined for the MCMOrderAtomic class. Note that it inherits the MCMOrderStructureHasMCMOrderItem aggregation from MCMOrderStructure.

7.7.4 MCMOrderComposite Class Definition

This is a concrete class, and specializes MCMOrderStructure. This class represents a set of related MCMOrderStructure objects that are organized into a tree structure. Note that an MCMOrderComposite may also contain one or more MCMOrderItem objects.

[O35] Each MCMOrderComposite **MAY** contain zero or more MCMOrderAtomic and/or zero or more MCMOrderComposite objects.

[O36] Each MCMOrderComposite **MAY** contain one or more MCMOrderItems.

At this time, no attributes are defined for the MCMOrderComposite class. Most attributes will likely be realized using relationships and/or operations. For example, a query to an instance of the MCMOrderComposite class to provide its set of contained MCMOrderStructure objects will be done by using class operations; the MCMOrderComposite instance will query each of its contained MCMOrderStructure objects (which will in turn call their operations to acquire their details, including MCMOrderItems), aggregate and organize the information, and provide that information in its operation response.

Table 26 defines the following operations for this class:

Operation Name	Description
getMCMOrderStructureChildList() : MCMOrderStructure [1..*]	This operation returns the set of all MCMOrderStructure objects that are contained in this specific MCMOrderComposite object. There are no input parameters to this operation. This operation returns a list of one or more MCMOrderStructure objects (i.e., the list is made up of MCMOrderAtomic and/or MCMOrderComposite objects. **[D82]** If this object does not have any attached MCMOrderStructure objects, then a NULL MCMOrderStructure object **SHOULD** be returned.

setMCMOrderStructureChildList(in childObjectList : MCMOrderStructure [1..*])	This operation defines a set of MCMOrderStructure objects that will be contained by this particular MCMOrderComposite object. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMOrderStructure objects (i.e., one or more MCMOrderAtomic and/or MCMOrderComposite objects). This has the effect of creating an instance of the MCMHasOrder aggregation between each MCMOrderStructure object in the childObjectList and this particular MCMOrderComposite object. Note that this operation first deletes any existing contained MCMOrderStructure objects (and their aggregations and association classes), and then instantiates a new set of MCMOrderStructure objects; in doing so, each MCMOrderStructure object is contained within this particular MCMOrderComposite object by first, creating an instance of the MCMHasOrder aggregation, and second, realizing that aggregation instance as an association class. **[D83]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasOrderDetail association class).
setMCMOrderStructurePartialChildList(childObjectList : MCMOrderStructure [1..*])	This operation defines a set of one or more MCMOrderStructure objects that should be contained within this particular MCMOrderComposite object WITHOUT affecting any other existing contained MCMOrderStructure objects or the objects that are contained in them. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMOrderStructure objects. This has the effect of creating a set of aggregations between this particular MCMOrderComposite object and each of the MCMOrderStructure objects identified in the childObjectList. **[D84]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasOrderDetail class).
delMCMOrderCompositeChildren()	This operation deletes ALL contained MCMOrderStructure objects of this particular MCMOrderComposite object. This has the effect of first, removing the association class, and second, removing the aggregation, between this MCMOrderComposite object and each MCMOrderStructure object that is contained in this MCMOrderComposite object. This operation has no input parameters.
delMCMOrderCompositePartialChildList(in childObjectList : MCMLocation[1..*])	This operation deletes a set of MCMOrderStructure objects from this particular MCMOrderComposite object WITHOUT affecting any other existing contained MCMOrderStructure objects or the objects that are contained in them. This operation takes a single input parameter, called childObjectList, which is

	an array of one or more MCMOrderStructure objects. This has the effect of first, removing the association class and second, removing the aggregation, between each MCMOrderStructure object specified in the input parameter and this MCMOrder-Composite object. Note that all other aggregations between this MCMOrderComposite and other MCMOrderStructure objects that are not identified in the input parameter are NOT affected.

Table 26. Operations for the MCMOrderComposite Class

The MCMOrderComposite class defines a single aggregation, called MCMHasOrder. This aggregation is used to define the set of MCMOrderStructure objects that are contained within this particular MCMOrderComposite object. Its multiplicity is defined to be 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMOrderStructure objects can be aggregated by this particular MCMOrderComposite object. Note that the cardinality on the part side (MCMOrderStructure) is 0..*; this enables an MCMOrderComposite object to be defined without having to define an associated MCMOrderStructure object for it to aggregate.

The semantics of the MCMHasOrder aggregation is realized using an association class, called MCMHasOrderDetail. This enables the semantics of the MCMHasOrder aggregation to be realized using the attributes, operations, and relationships of the MCMHasOrderDetail association class.

The Policy Pattern may be used to control which specific MCMOrderStructure objects are contained within a given MCMOrderComposite object for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.7.5 MCMOrderItem Class Definition

This is a concrete class, and specializes MCMBusinessObject. It represents a set of MCMEntities tha can be ordered by a Buyer from a Seller, and are contained in a particular MCMOrderStructure object.

Table 27 defines the attributes of the MCMOrderItem class.

Attribute Name	Mandatory?	Description
mcmOrderItemCompleteActual : TimeAndDate[0..1]	NO	This is a TimeAndDate attribute. It defines the date and time that this MCMOrderItem was completed by the Seller (e.g., MCMServiceProvider or MCMPartner). This enables individual MCMOrderItems to be

		tracked. It is optional to facilitate the case where an order has been issued but has not been complete. It is based on [15]. **[D85]** This attribute **SHOULD** have a complete and valid time and/or date. **[O37]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
mcmOrderItemCompleteRequest : TimeAndDate[1..1]	YES	This is a TimeAndDate attribute. It defines the date and time by which the Buyer (e.g., MCMCustomer or MCMPartner) requested that this MCMOrderItem be completed. It is based on [15]. **[D86]** This attribute **SHOULD** have a complete and valid time and/or date. **[O38]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
mcmOrderItemDesiredResponse : OrderDesiredResponse[1..1]	YES	This is a mandatory enumeration, which defines the desired response that the Buyer wishes to receive from the Seller concerning this particular MCMOrderItem. It is based on [15]. The values are defined in the OrderDesiredResponse enumeration, and includes: ERROR INIT CONFIRMATION AND ENGINEERING DESIGN CONFIRMATION ONLY NONE
mcmIsOrderItemPriorityExpedited : Boolean[1..1]	YES	This is a Boolean attribute that indicates if this OrderItem is being expedited or not. The default value of this attribute is FALSE. IFF the value of this Boolean is FALSE, then the mcmOrderItemPriority class attribute is irrelevant. Otherwise, the priority of this OrderItem (if any) is defined by the value of the mcmOrderItemPriority attribute. It is based on [15]. **[R50]** IFF the value of the mcmIsPriorityExpedited class attribute is FALSE, then the implementation **MUST** ignore the value of this class attribute.

		[R51] IFF the value of the mcmIsPriorityExpedited class attribute is TRUE, then the implementation **MUST** assign the priority of this Order to the value of this class attribute. **[O39]** If two or more Orders have the same value of this attribute, then the implementation **MAY** process them in any order, as long as each is processed in appropriate numerical order of the mcmOrderPriority attribute value with respect to all other Orders. **[R52]** If an Order does not have a value for this attribute, then the implementation **MUST** process it (and any other similar Orders) after all Orders that do have a valid value for this attribute.
mcmOrderItemPriority : Integer[1..1]	NO	This is a non-negative Integer attribute, which indicates the relative priority of this Order compared to other Orders between this particular Buyer and Seller. The meaning of this attribute is subject to agreement between the Buyer and the Seller. [15] **[R53]** IFF the value of the mcmIsPriorityExpedited class attribute is FALSE, then the implementation **MUST** ignore the value of this class attribute. **[R54]** IFF the value of the mcmIsPriorityExpedited class attribute is TRUE, then the implementation **MUST** assign the priority of this Order to the value of this class attribute. **[O40]** If two or more Orders have the same value of this attribute, then the implementation **MAY** process them in any order, as long as each is processed in appropriate numerical order of the mcmOrderPriority attribute value with respect to all other Orders. **[R55]** If an Order does not have a value for this attribute, then the implementation

		MUST process it (and any other similar Orders) after all Orders that do have a valid value for this attribute.
mcmOrderItemType : OrderType[1..1]	YES	This is a mandatory enumeration, which defines the type of MCMOrderItem that this instance is. It is based on the use cases in [15]. The values are defined in the OrderType enumeration, and include: ERROR INIT CREATE INSTALL CHANGE DISCONNECT QUERY AMEND CANCEL NOTIFY COMPLETE NO_CHANGE

Table 27. Attributes of the MCMOrderItem Class

Table 28 defines the operations for this class:

Operation Name	Description
getMCMOrderItemCompleteActual() : TimeAndDate[1..1]	This operation returns the date and time that this MCMOrderItem was completed by the Seller (e.g., MCMServiceProvider or MCMPartner). There are no input parameters to this operation. **[D87]** This attribute **SHOULD** have a complete and valid time and/or date. **[O41]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
setMCMOrderItemCompleteActual(in newCompletionDate : TimeAndDate[1..1])	This operation defines the time and date (as a TimeAndDate data type) that the Buyer (e.g., MCMCustomer or MCMPartner) requested that this MCMOrderItem be completed. There is a single input parameter, called newCompletionDate (of data type TimeAndDate) that contains the new date and time information that this order should be finished. It is based on [15].

getMCMOrderItem-CompleteRequest() : TimeAndDate[1..1]	This operation returns the date and time that the Buyer (e.g., MCMCustomer or MCMPartner) requested that this MCMOrder-Item be completed. There are no input parameters to this operation. **[D88]** This attribute **SHOULD** have a complete and valid time and/or date. **[O42]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
setMCMOrderItem-CompleteRequest(in newCompletionDate : TimeAndDate[1..1])	This operation defines the time and date (as a TimeAndDate data type) that the Buyer (e.g., MCMCustomer or MCMPartner) requested that this MCMOrderItem be completed. There is a single input parameter, called newRequestDate (of data type Time-AndDate) that contains the new date and time information that this order is requested to be finished. It is based on [15].
getMCMOrder-ItemDesiredResponse() : OrderDesiredRe-sponse[1..1]	This operation retrieves the value of the mcmOrderItemDesire-dRe-sponse class attribute, which defines the desired response that the Buyer wishes to receive from the Seller concerning this particular MCMOrderItem. This operation takes no input parameters, and returns a value from the OrderDesiredResponse enumeration. It is based on [15].
setMCMOrder-ItemDesiredResponse(in newResponse : Order-DesiredResponse[1..1])	This operation defines the value of the mcmOrderItemDesiredResponse class attribute, which indicates the desired response that the Buyer wishes to receive from the Seller concerning this particular MCMOrderItem. This operation takes a single input parameter, called newResponse, which is an OrderDesiredResponse enumeration. It is based on [15].
getMCMIsOrderItem-PriorityExpedited() : Boolean[1..1]	This operation retrieves the value of the mcmOrderItemIsPriority-Expedited class attribute. IFF the value of this Boolean is FALSE, then the mcmOrderItemPriority class attribute is irrelevant. Otherwise, the priority of this OrderItem (if any) is defined by the value of the mcmOrderItemPriority attribute. This operation takes no input parameters, and returns a Boolean indicating true (if the MCMOrderItem should be expedited) or false. It is based on [15]. **[R56]** IFF the value of this class attribute is FALSE, then the implementation **MUST** ignore the value of the mcmIsOrder-ItemPriority class attribute. **[R57]** IFF the value of this class attribute is TRUE, then the implementation **MUST** assign the priority of this OrderItem to the value of the mcmIsOrderItemPriority class attribute.
setMCMIsOrderItem-PriorityExpedited(in	This operation defines the value of the mcmOrderItemIsPriorityEx-pedited class attribute. IFF the value of this Boolean is FALSE, then

isExpedited : Boolean[1..1])	the mcmOrderItemPriority class attribute is irrelevant. Otherwise, the priority of this OrderItem (if any) is defined by the value of the mcmOrderItemPriority attribute. This operation takes a single input parameter, called isExpedited, which is true if the MCMOrderItem should be expedited, and false otherwise. It is based on [15]. **[R58]** IFF the value of this class attribute is FALSE, then the implementation **MUST** ignore the value of the mcmIsOrderItemPriority class attribute. **[R59]** IFF the value of this class attribute is TRUE, then the implementation **MUST** assign the priority of this OrderItem to the value of the mcmIsOrderItemPriority class attribute.
getMCMOrderItemPriority() : Integer[1..1]	This operation retrieves the value of the mcmOrderItemPriority class attribute. This defines the relative priority of this OrderItem compared to other OrderItems for this Order between this particular Buyer and Seller. This operation takes no input parameters, and returns an Integer indicating the relative priority of this OrderItem. It is based on [15]. **[R60]** IFF the value of the mcmIsOrderItemPriorityExpedited class attribute is FALSE, then the implementation **MUST** ignore the value of this class attribute. **[R61]** IFF the value of the mcmIsOrderItemPriorityExpedited class attribute is TRUE, then the implementation **MUST** assign the priority of this OrderItem to the value of this class attribute. . **[O43]** If two or more OrderItems have the same value of this attribute, then the implementation **MAY** process them in any order, as long as each is processed in appropriate numerical order of the mcmOrderPriority attribute value with respect to all other Orders. **[R62]** If an OrderItem does not have a value for this attribute, then the implementation **MUST** process it (and any other similar Orders) after all Orders that do have a valid value for this attribute.
setMCMOrderItemPriority (in newPriority : Integer[1..1])	This operation defines the value of the mcmOrderItemPriority class attribute. This defines the relative priority of this OrderItem compared to other OrderItems for this Order between this particular Buyer and Seller. This operation takes a single input parameter, called newPriority, which defines the relative priority of this OrderItem compared to other OrderItems in the same Order. It is based on [15].

	[R63] IFF the value of the mcmIsOrderItemPriorityExpedited class attribute is FALSE, then the implementation **MUST** ignore the value of this class attribute. **[R64]** IFF the value of the mcmIsOrderItemPriorityExpedited class attribute is TRUE, then the implementation **MUST** assign the priority of this OrderItem to the value of this class attribute. **[O44]** If two or more OrderItems have the same value of this attribute, then the implementation **MAY** process them in any order, as long as each is processed in appropriate numerical order of the mcmOrderPriority attribute value with respect to all other Orders. **[R65]** If an OrderItem does not have a value for this attribute, then the implementation **MUST** process it (and any other similar Orders) after all Orders that do have a valid value for this attribute.
getMCMOrder-ItemType() : Order-Type[1..1]	This operation retrieves the value of the mcmOrderItemType class attribute. This defines the type of MCMOrderItem that this instance is. This operation takes no input parameters, and returns a value from the OrderType enumeration. It is based on [15].
setMCMOrderItemType (in newType : Order-Type[1..1])	This operation defines the value of the mcmOrderItemType class attribute. This defines the type of MCMOrderItem that this instance is. This operation takes a single input parameter, called newType, which defines the type of OrderItem based on the values of the OrderType enumeration. It is based on [15].
getMCMOrderOfOrder-Item() : MCMOrder-Structure [1..1]	This operation retrieves the MCMOrderStructure that contains this MCMOrderItem. This enables MCMOrderItems to query the data of the MCMOrderStructure that contains them. This operation takes no input parameters. **[D89]** If this MCMOrderItem has no containing MCMOrderStructure, then it **SHOULD** return a NULL MCMOrderStructure object.
setMCMOrderOfOrder-Item(in newContain-ingOrder : MCMOrder-Structure [1..1], in new-Position : Integer[1..1])	This operation defines a new MCMOrder to contain this particular MCMOrderItem. This operation takes two input parameters, called newContainingOrder and newPosition. The first is an MCMOrder object, and the second is the position that this MCMOrderItem should occupy, if any (0 is a don't care, while positive integers correspond to an increasing sequence of MCMOrder-Items, starting with 1).
delMCMOrderItemOf-Order()	This operation removes the aggregation, and its association class, that enables this MCMOrderStructure to contain this MCMOrder-

	Item. This is done by first, removing the association class, and second, removing the aggregation, between this MCMOrderItem object and this MCMOrderStructure object. This operation does NOT affect the MCMOrderStructure object; it just deletes the aggregation between this MCMOrderStructure and this MCMOrderItem. This operation has no input parameters.

Table 28. Operations of the MCMOrderItem Class

At this time, no relationships are defined for the MCMOrderItem class; it participates in the MCMOrderStructureHasMCMOrderItem aggregation (see section 7.7.2).

7.8 MCMManagedEntity Class Hierarchy

The MCMManagedEntity class has five abstract subclasses, as shown in Figure 12.

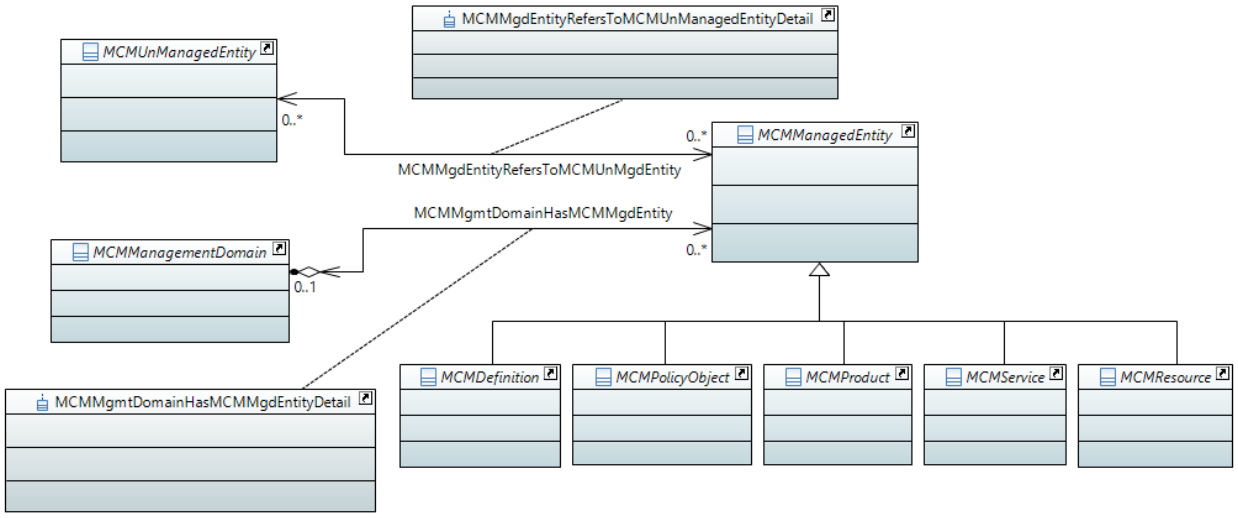

Figure 12. ManagedEntity Subclasses

Table 29 defines the purpose of this hierarchy, and aligns them to MEF 55.1. The purpose of the MCMManagedEntity hierarchy is to model the major different types of manageable entities that are of interest to the managed environment. This hierarchy is based around the need to represent and manage Products, Services, and Resources. As such, the MCMDefinition hierarchy is used to specify common characteristics and behavior of these three concepts, and the MCMPolicyObject hierarchy is used to manage these three concepts.

Name of Class	Function	Relation to MEF 55
MCMManagedEntity	Represents objects that have the following common semantics: (1) each has the potential to be managed; (2) each can be associated with at least one MCMManagementDomain; (3) each is related to Products, Resources, and/or Services of the system being managed.	The base class for defining Products, Services, and Resources that are defined and used in MEF55.
MCMDefinition	The MCM equivalent of the ONF and TMF "specification" classes. It defines the salient characteristics, capabilities, and constraints of concrete subclasses of an MCMManagedEntity. When concrete subclasses of MCMDefinition are instantiated, these characteristics, capa-	Critical to enabling scalable and consistent creation of Product, Service, and Resource hierarchies that share common properties and behavior.

	bilities, and constraints will be invariant over all instances of each concrete subclass of MCMDefinition.	
MCMPolicyObject	The root of the Policy Model. This provides a set of abstractions for viewing any type of Policy, regardless of its programming paradigm (e.g., imperative, declarative, intent), as a set of statements.	Realizes the Policy Driven Orchestration information model. Enables imperative, declarative, and intent policies to be used in an MEF LSO environment.
MCMProduct	Defines the set of goods and services, offered to a market by an MCMParty that is playing an appropriate MCMPartyRole. MCMProducts are purchased by an MCMCustomer, which is a type of MCMPartyRole. Each such purchased Product is based on an MCMProductOffer, even if it uses shared Resources and/or Services, and results in a separate instance of the MCMProduct class.	Models Products in an extensible way.
MCMResource	Defines a set of capabilities that may be consumed by other Resources and/or Services. Resources are typically limited in quantity and/or availability. Resources may be logical or virtual in nature. Note that physical resources are NOT defined as a subclass of Resource because a physical entity is not inherently manageable. Rather, physical resources are defined by the PhysicalElement class, which is a subclass of UnManagedEntity.	Models Resources in an extensible way. This includes legacy as well as NFV, SDN, and other types of Resources.
MCMService	Represents functionality that can be used by different internal and external users (e.g., a management system and a Customer, respectively) for different purposes. Services may be used by other Services, but not by Resources.	Models Services in an extensible way.

Table 29. Functions of the MCMManagedEntity Class and its Subclasses

7.8.1 MCMManagedEntity Class Definition

This is an abstract class, and specializes MCMEntity. It represents objects that have the following common semantics: (1) each has the potential to be managed; (2) each can be associated with at least one ManagementDomain; (3) each can be related to Products, Resources, and/or Services of the system being managed.

A common need of many operational and business support systems is to define an objectID that meets their business needs. For example, a purchase order ID might be expected to have a particular structure. The MCM has therefore defined an attribute, called mcmExternalIDAttrName, to provide this flexibility.

[R66] The mcmExternalIDAttrName attribute **MUST** be defined as a string, in order to simplify the design and improve interoperability.

This enables operational and business support systems to name an attribute that can be used for all MCMManagedEntity classes. This attribute is defined as a string, to enable different applications to use this objectID in an interoperable manner.

[O45] MCMMetaData **MAY** be used to augment the meaning of these attributes by attaching a set of MCMMetaData objects to an instance of the MCMManagedEntity class (or any of its subclasses).

Table 30 defines the attributes of the MCMManagedEntity class.

Attribute Name	Mandatory?	Description
mcmAdminState : MCMAdminState[1..1]	YES	This is a mandatory enumeration that defines the set of states for what the IETF and ITU-T call "AdminStatus". Note that the MCM extends both of these concepts. This attribute defines the current ability of this MCMManagedEntity to communicate with and respond to service requests from other MCMManagedEntity objects. The values that this attribute can have are defined by the MCMAdminState enumeration, and include: ERROR INIT ENABLED_FOR_USE LOCKED IN_TEST UNKNOWN

mcmOperState : MCMOperState[1..1]	YES	This is a mandatory enumeration that defines the set of states for what the IETF and ITU-T call "OperStatus". Note that the MCM version extends both of these concepts. This attribute defines the current operational state of this MCMManagedEntity. The values that this attribute can have are defined by the MCMOperState enumeration, and include: ERROR INIT ENABLED_FOR_USE INSTALLED_AND_OPERATING_CORRECTLY INSTALLED_AND_NOT_OPERATING_CORRECTLY INSTALLED_BUT_NOT_OPERATING NOT_INSTALLED IN_TEST LOCKED UNKNOWN INSTALLED_AND_NOT_OPERATING_CORRECTLY means that the object installed but has one or more pending alarms that have not been cleared. INSTALLED_BUT_NOT_OPERATING means that the object is in a shutdown, powered-off, or similar state. IN_TEST means that the object can only respond to testing commands and communications LOCKED means that the object is prohibited from being used UNKNOWN means that this object was unable to report its status when communication was last attempted
mcmMgdEntityCreationDate : TimeAndDate[1..1]	YES	This is a TimeAndDate attribute. It defines the date and time that this MCMManagedEntity object instance was created. **[D90]** This attribute **SHOULD** have a complete and valid time and/or date. **[O46]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
mcmExternalIDAttrName : String[0..1]	NO	The mcmExternalIDAppName attribute is a string, and defines the name of an objectID that an external Application is using. **[R67]** This attribute **MUST NOT** be used as a naming attribute (i.e., to uniquely identify an instance of the object). **[D91]** If an object does not have a value for this class attribute, then an empty string **SHOULD** be used.

Table 30. Attributes of the MCMManagedEntity Class

Table 31 defines the operations for this class:

Operation Name	Description
getMCMAdminState() : MCMAdminState[1..1]	This operation returns the value of the mcmAdminState attribute. There are no input parameters to this operation. The value returned is one of the values defined in the MCMAdminState enumeration.
setMCMAdminState (in newAdminState : MCMAdminState[1..1])	This operation defines the new value for the mcmAdminState attribute. There is a single input parameter, called newAdminState (of data type MCMAdminState) that contains a set of valid values to be used.
getMCMOperState() : MCMOperState[1..1]	This operation returns the value of the mcmOperState attribute. There are no input parameters to this operation. The value returned is one of the values defined in the MCMOperState enumeration.
setMCMOperState(in newOperState : MCMOperState[1..1])	This operation defines the new value for the mcmOperState attribute. There is a single input parameter, called newOperState (of data type MCMOperState) that contains a set of valid values to be used.
getMCMMgdEntityCreationDate() : TimeAndDate[1..1]	This operation returns the value of the mcmMgdEntityCreationDate attribute. There are no input parameters to this operation. The value returned is a TimeAndDate attribute. **[D92]** This attribute **SHOULD** have a complete and valid time and/or date. **[O47]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
setMCMMgdEntityCreationDate(in newTimeAndDate : TimeAndDate [1..1])	This operation defines a new value for the mcmMgdEntityCreationDate attribute. There is a single input parameter, called newTimeAndDate (of data type TimeAndDate) that contains a set of valid values to be used.
getMCMExternalIDAttrName () : String[1..1]	This operation retrieves the value of the mcmExternalIDAttrName attribute, which is a string that contains the name of the ExternalID attribute that is being defined for use in the MCM. **[R68]** This class attribute **MUST NOT** be used as a naming attribute (i.e., to uniquely identify an instance of the object). **[D93]** If an object does not have a value for this class attribute, then an empty string **SHOULD** be used.
setMCMExternalIDAttrName (in newAttrName : String[1..1])	This operation defines a new value for the ExternalIDAttrName attribute. There is a single input parameter, called newAttrName (of

	data type String) that defines the new name of the mcmExternalIDAttrName attribute. **[R69]** This class attribute **MUST NOT** be used as a naming attribute (i.e., to uniquely identify an instance of the object). **[O48]** If an object does not have a value for this class attribute, then an empty string **SHOULD** be used.
getMCMParentDomain() : MCMManagementDomain[1..1])	This operation retrieves the MCMManagementDomain that contains this MCMManagedEntity. This operation takes no input parameters. **[D94]** If this MCMManagedEntity has no containing MCMManagementDomain, then it **SHOULD** return a NULL MCMManagementDomain object.
setMCMParentDomain (in newMgmtDomain : MCMManagementDomain[1..1])	This operation defines a new MCMManagementDomain to contain this particular MCMManagedEntity. This operation takes a single input parameter, called newMgmtDomain, which is an MCMManagementDomain object. If this MCMManagedEntity object already has a parent MCMManagementDomain, then this MCMManagementDomain will be deleted by first, deleting the accompanying association class, and second, deleting the corresponding aggregation. Then, a new aggregation (an instance of MCMMgmtDomainHasMCMMgdEntity) is created; following that, a new association class is then created to realize the semantics of the aggregation.
delMCMParentDomain()	This operation removes the aggregation, and its association class, that enables this MCMManagedEntity to be contained by this MCMManagementDomain. This operation does NOT affect either the MCMManagementDomain object or the MCMManagedEntity object; it just deletes the aggregation between this MCMManagementDomain object and this MCMManagedEntity. This operation has no input parameters.
getReferredMCMUnManagedEntityList() : MCMUnManagedEntity[1..*]	This operation retrieves the set of MCMUnManagedEntity objects that refer to this MCMManagedEntity object. This operation takes no input parameters. **[D95]** If this MCMManagedEntity object has no MCMUnManagedEntity object that it refers to, then it **SHOULD** return a NULL MCMUnManagedEntity object.
setReferredMCMUnManagedEntityList(in newUnMgdEntityList : MCMUnManagedEntity[1..*])	This operation defines a new set of MCMUnManagedEntity objects that refer to this particular MCMManagedEntity object. This operation takes a single input parameter, called newUnMgdEntityList, which defines a set of one or more MCMUnManagedEntity objects. If this MCMManagedEntity object already has a set of one or more MCMUnManagedEntity objects that it refers

	to, then those MCMUnManagedEntity objects will be deleted by first, deleting the accompanying association class, and second, deleting the corresponding association. Then, a new association (an instance of MCMMgdEntityRefersToMCMUnManagedEntity) is created for each UnManagedEntity object in the newUnMgdEntityList. **[D96]** Every association created **SHOULD** have a new association class created to realize the semantics of that association.
setReferredMCMUnManagedEntityPartialList(in newUnMgdEntityList : MCMUnManagedEntity[1..*])	This operation defines a new set of MCMUnManagedEntity objects that refer to this particular MCMManagedEntity object. This operation takes a single input parameter, called newUnMgdEntityList, which defines a set of one or more MCMUnManagedEntity objects. If this MCMManagedEntity object already has a set of one or more MCMUnManagedEntity objects that it refers to, then those MCMUnManagedEntity objects are ignored. Then, a new association (an instance of MCMMgdEntityRefersToMCMUnManagedEntity) is created for each UnManagedEntity object in the newUnMgdEntityList. **[D97]** Every association created **SHOULD** have a new association class created to realize the semantics of that association.
delReferredMCMUnManagedEntity()	This operation removes the association, and its association class, that enables this MCMManagedEntity object to refer to any MCMUnManagedEntity objects. This operation does NOT affect either the MCMUnManagedEntity object or the MCMManagedEntity object; it just deletes the association between this MCMManagedEntity object and this MCMUnManagedEntity object. This operation has no input parameters.
delReferredMCMUnManagedEntityPartial(in unMgdEntityList : MCMUnManagedEntity[1..1])	This operation removes the association, and its association class, for each MCMUnManagedEntity object in the unMgdEntityList that is associated with this particular MCMManagedEntity. Any association between this MCMManagedEntity object and other MCMUnManagedEntity objects that are not specified in the unMgdEntityList are NOT affected. This operation does NOT affect either the MCMUnManagedEntity object or the MCMManagedEntity object; it just deletes the association between this MCMManagedEntity object and this MCMUnManagedEntity object. This operation has no input parameters.

Table 31. Operations of the MCMManagedEntity Class

At this time, the MCMManagedEntity class defines a single association, called MCMMgdEntityRefersToMCMUnMgdEntity. This association enables an MCMManagedEntity to refer to a set of MCMUnManagedEntities, and vice versa. The multiplicity of this relationship is 0..1 – 0..*. This means that this association is optional (i.e., the "0" part of the 0..1 cardinality). If this association is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMUnManagedEntity objects can be associated with this particular MCMManagedEntity object. Note that the cardinality on the part side (MCMUnManagedEntity) is 0..*; this enables an MCMManagedEntity object to be defined without having to define an associated MCMUnManagedEntity object for it. For example, an MCMService could be associated with the location of an MCMPhysicalEntity at a particular MCMLocation.

The semantics of this association are defined by the MCMMgdEntityRefersToMCMUnMgdEntityDetail association class. This enables the semantics of the association to be defined using the attributes and behavior of this association class. For example, it can be used to define which MCMUnManagedEntity objects are allowed to be associated with which MCMManagedEntity objects (or vice-versa).

The Policy Pattern (see Figure 3) may be used to define policy rules that constrain which objects of one type are related to which objects of the other type (e.g., which MCMUnManagedEntity objects are related to which MCMManagedEntity objects). Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

The MCMManagedEntity class also participates in two aggregations, called MCMMgmtDomainHasMCMMgdEntity and MCMCatalogItemContainsMCMManagedEntity. These two aggregations are defined in sections 7.6 and 7.8.6.9, respectively.

7.8.2 MCMDefinition Class Hierarchy

The MCMDefinition class hierarchy is shown in Figure 13.

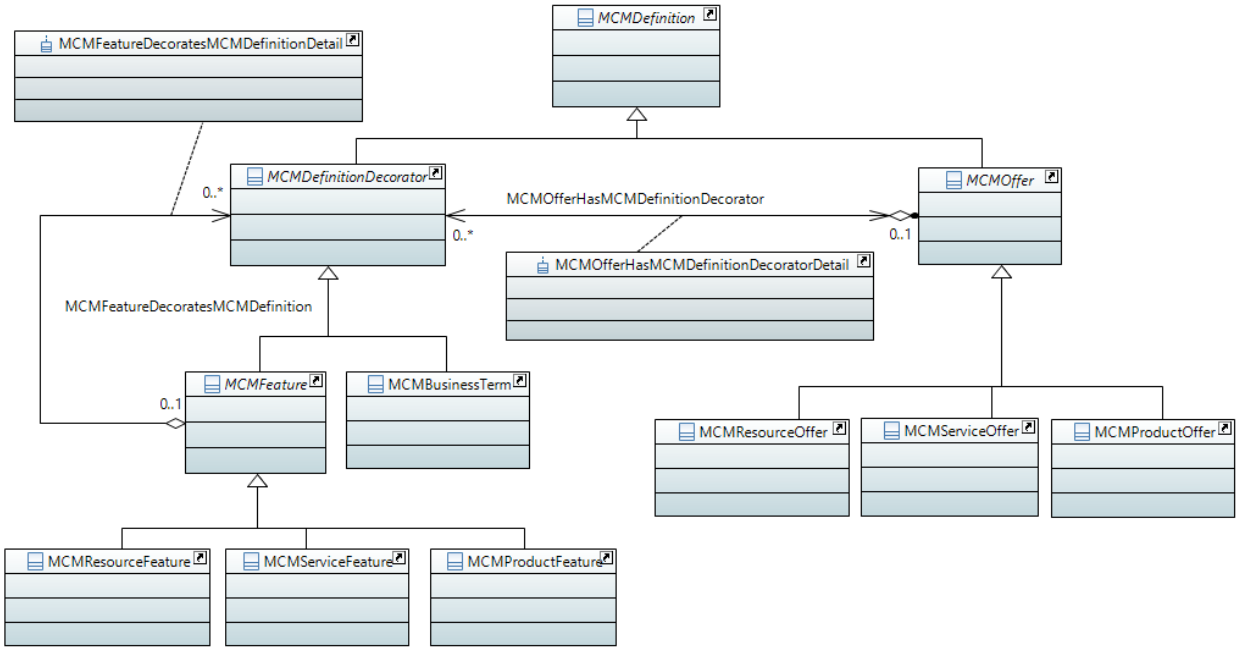

Figure 13. MCMDefinition Class Hierarchy

7.8.2.1 MCMDefinition Class Definition

This is an abstract class, and specializes MCMManagedEntity. It provides the salient characteristics, capabilities, and constraints of concrete subclasses of an MCMManagedEntity. Hence, it can be thought of as a template that define common characteristics and behavior of instantiated objects of this class. When concrete subclasses of MCMDefinition are instantiated, these characteristics, capabilities, and constraints will be invariant over all instances of each concrete subclass of MCMDefinition.

At this time, no attributes are defined for the MCMDefinition class.

At this time, no operations are defined for the MCMDefinition class.

At this time, no relationships are defined for the MCMDefinition class.

7.8.2.2 MCMDefinitionDecorator Class Definition

This is an abstract class, and specializes MCMDefinition. It defines the decorator pattern applied to an MCMDefinitionDecorator, which enables all or part (e.g., a subset of the attributes of a class) of one or more concrete subclasses of MCMFeature to "wrap" another concrete subclass of MCMDefinitionDecorator (e.g., a subclass of MCMFeature or MCMBusinessTerm).

At this time, no attributes are defined for the MCMDefinitionDecorator class.

At this time, no relationships are defined for the MCMDefinitionDecorator class. It participates in two aggregations, called MCMFeatureDecoratesMCMDefinition (see section 7.8.2.4) and MCMOfferHasMCMDefinitionDecorator (see section 7.8.2.8).

7.8.2.3 MCMBusinessTerm Class Definition

This is a concrete class, and specializes MCMDefinitionDecorator. It defines the set of business terms that dictate how a particular type of MCMOffer (i.e., a business offering, typically based on demographics,) is sold to Customers. An MCMOffer aggregates one or more MCMFeatures, MCMBusinessTerms, and other business logic; please see section 7.8.2.8 for the definition of an MCMOffer.

Table 32 defines the attributes of the MCMBusinessTerm class.

Attribute Name	Mandatory?	Description
mcmBusTermRMM : String[0..1]	NO	This is a string attribute. It consists of free-form text that describes the remote monitoring and management (RMM) capabilities included in this MCMOffer. RMM solutions enable many mundane, time-consuming activities to be scripted and delivered on a scheduled basis without human intervention (e.g., operating system and software application patch management, antivirus and antispam updates, disk optimization and backup).
mcmBusTermServiceDesk : String[0..1]	NO	This is a string attribute. It defines the type of problem management and remediation services that are available to MCMCustomers that purchase this MCMOffer. The service desk functions as the single point of contact for all end-user issues.
mcmBusTermVendorMgmt : String[0..1]	NO	This is a string attribute. It defines the type of vendor management that is included for Buyers that purchase an MCMOffer that has this MCMBusinessTerm. Vendor management offloads all interactions with the vendors from the customer. This service adds tremendous value to the relationship between the MCMCustomer and the MCMServiceProvider, as the MCMCustomer need only open a service request for any issue affecting their MCMProduct purchase.

Table 32. Attributes of the MCMBusinessTerm Class

Table 33 defines the operations for this class:

Operation Name	Description
getMCMBusTermRMM() : MCMString[1..1]	This operation returns the value of the mcmBusTermRMM attribute. There are no input parameters to this operation. The value returned is a string attribute that describes the remote monitoring and management capabilities of this MCMBusinessTerm. **[D98]** If the mcmBusTermRMM attribute is empty, then an empty string **SHOULD** be returned.
setMCMBusTermRMM (in newString : String[1..1])	This operation defines a new value for the mcmBusTermRMM attribute. There is a single input parameter, called newString (of data type String) that contains the text that describes the remote monitoring and management capabilities of this MCMBusinessTerm.

	[O49] The newString attribute **MAY** contain an empty string (e.g., for clearing this field).
getMCMBusTerm-ServiceDesk() : String[1..1]	This operation returns the value of the mcmBusTermServiceDesk attribute. There are no input parameters to this operation. The value returned is a string attribute that describes the problem management and remediation services of this MCMBusi-nessTerm. **[D99]** If the mcmBusTermRMM attribute is empty, then an empty string **SHOULD** be returned.
setMCMBusTerm-ServiceDesk (in new-String : String[1..1])	This operation defines the new value for the mcmBusTerm-ServiceDesk attribute. There is a single input parameter, called newString (of data type String) that contains a description of the problem management and remediation services of this MCMBusi-nessTerm. **[O50]** The newString attribute **MAY** contain an empty string (e.g., for clearing this field).
getMCMBusTermVen-dorMgmt() : String[1..1]	This operation returns the value of the mcmBusTermVen-dorMgmt attribute. There are no input parameters to this opera-tion. The value returned is a String that describes the type of ven-dor management that is included for Buyers that purchase an MCMOffer that has this MCMBusinessTerm. **[D100]** If the mcmBusTermVendorMgmt attribute is empty, then an empty string **SHOULD** be returned.
setMCMBusTermVen-dorMgmt (in newString : String[1..1])	This operation defines a new value for the mcmBusTermVen-dorMgmt attribute. There is a single input parameter, called new-String (of data type String) that contains a description of the type of vendor management that is included for Buyers that purchase an MCMOffer that has this MCMBusinessTerm. **[O51]** The newString attribute **MAY** contain an empty string (e.g., for clearing this field).
getMCMFeatureList() : MCMFeature[1..*]	This operation returns the set of MCMFeature objects that cur-rently decorate this MCMBusinessTerm object. The return value is an array of one or more objects of type MCMFeature. **[D101]** If this MCMBusinessTerm object is not decorated by any MCMFeature objects, then a NULL MCMFeature object **SHOULD** be returned.
setMCMFeatureList(in newFeatureList : MCMFeature[1..*])	This operation defines the set of MCMFeatures that will decorate this MCMBusinessTerm object. This method takes a single input parameter, called newFeatureList, which is an array of MCMFea-ture objects. This operation decorates this particular MCMBusi-nessTerm object with the set of MCMFeature objects identified in

	the input parameter. Note that this operation first deletes any existing MCMFeature objects that decorate the MCMBusinessTerm object, and then instantiates a new set of MCMFeature objects to decorate this particular MCMBusinessTerm object. **[O52]** Implementations **MAY** realize the decorator pattern in any way they wish, so long as the Decorator forwards requests to the object that it is wrapping. **[O53]** A decorator object **MAY** perform additional actions before and/or after forwarding requests to the object that it is wrapping.
setMCMFeaturePartialList(in newFeaturePartialList : MCMFeature[1..*])	This operation defines the set of MCMFeatures that will decorate this MCMBusinessTerm object WITHOUT affecting any other decorated objects on this MCMBusinessTerm object. This method takes a single input parameter, called newFeaturePartialList, which is an array of MCMFeature objects. This operation decorates this particular MCMBusinessTerm object with the set of MCMFeature objects identified in the input parameter. No other model elements of this MCMBusinessTerm object are affected. **[O54]** Implementations **MAY** realize the decorator pattern in any way they wish, so long as the Decorator forwards requests to the object that it is wrapping. **[O55]** A decorator object **MAY** perform additional actions before and/or after forwarding requests to the object that it is wrapping.
delMCMFeatureList()	This operation removes ALL instances of MCMFeature objects that were decorating this particular MCMBusinessTerm object. **[O56]** Implementations **MAY** remove the decorating object any way they wish, including deleting the object.
delMCMFeaturePartialList(in newFeaturePartialList : MCMFeature[1..*])	This operation removes the set of MCMFeature objects identified in the input parameter that were decorating this MCMBusinessTerm object WITHOUT affecting any other decorated objects on this MCMBusinessTerm object. This operation takes a single input parameter, called newFeaturePartialList, which is an array of one or more MCMFeature objects. **[O57]** Implementations **MAY** remove the decorating object any way they wish, including deleting the object.

Table 33. Operations of the MCMBusinessTerm Class

At this time, no relationships are defined for the MCMBusinessTerm class.

7.8.2.4 MCMFeature Class Description

This is an abstract class, and specializes MCMDefinitionDecorator. It defines the characteristics or behavior of a set of functions that are contained in an MCMOffer.

Conceptually, an MCMFeature is a salient type of characteristic or behavior of an object that it describes. An MCMFeature may be related to one or more MCMCapability objects (see section 7.11.6.1) via the MCMEntityHasMCMMetaData aggregation (see section 7.4). This enables a list of used and unused capabilities to augment the definition of each MCMFeature object.

MCMFeature is the superclass for three subclasses – MCMProductFeature, MCMServiceFeature, and MCMResourceFeature. This enables features that are part of the templates that define MCMProduct, MCMService, and MCMResource, respectively, to be used to construct a business offering (a subclass of MCMOffer). MCMFeatures play an important role in constructing MCMOffers; please see section 7.8.2.4.

At this time, no attributes are defined for the MCMFeature class.

At this time, no operations are defined for the MCMFeature class.

At this time, a single aggregation is defined for the MCMFeature class. This aggregation is named MCMFeatureDecoratesMCMDefinition, and defines the set of MCMFeatures that wrap (or decorate) this particular MCMDefinition object. The multiplicity of this aggregation is 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMFeature objects can wrap this particular MCMDefinitionDecorator object. The 0..* cardinality enables an MCMFeature object to be defined without having to define an associated MCMDefinitionDecorator object for it to aggregate. The semantics of this aggregation are defined by the MCMFeatureDecoratesMCMDefinitionDetail association class. This enables the management system to control which set of concrete subclasses of MCMFeature (e.g., a subclass of MCMFeature) are used to wrap a concrete subclass of MCMDefinitionDecorator (e.g., an MCMBusinessTerm).

The Policy Pattern may be used to control which specific MCMFeature objects are used to wrap a given MCMDefinition object for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.8.2.5 MCMProductFeature Class Definition

This is a concrete class, and specializes MCMFeature. It defines a set of the salient characteristics and behavior used to construct an MCMProductOffer for sale to a market. The characteristics and behavior of this class are application-specific, so in this definition of the MCM, the purpose of this class is solely to define the concept for different applications using the MCM to be able to create a common subclass for interoperability.

At this time, no attributes are defined for the MCMProductFeature class.

At this time, no operations are defined for the MCMProductFeature class.

At this time, no relationships are defined for the MCMProductFeature class.

7.8.2.6 MCMService Feature Class Definition

This is a concrete class, and specializes MCMFeature. It defines a set of the salient characteristics and behavior used to construct an MCMProductOffer or an MCMServiceOffer for use by an internal or external consumer. The characteristics and behavior of this class are application-specific, so in this definition of the MCM, the purpose of this class is solely to define the concept for different applications using the MCM to be able to create a common subclass for interoperability.

At this time, no attributes are defined for the MCMServiceFeature class.

At this time, no operations are defined for the MCMServiceFeature class.

At this time, no relationships are defined for the MCMServiceFeature class.

7.8.2.7 MCMResourceFeature Class Definition

This is a concrete class, and specializes MCMFeature. It defines a set of the salient characteristics and behavior used to construct an MCMProductOffer, MCMServiceOffer, or MCMResourceOffer for use by an internal or external consumer. The characteristics and behavior of this class are application-specific, so in this definition of the MCM, the purpose of this class is solely to define the concept for different applications using the MCM to be able to create a common subclass for interoperability.

At this time, no attributes are defined for the MCMResourceFeature class.

At this time, no operations are defined for the MCMResourceFeature class.

At this time, no relationships are defined for the MCMResourceFeature class.

7.8.2.8 MCMOffer Class Definition

This is an abstract class, and specializes MCMDefinition. It defines a business offering, typically based on demographics, to interact with internal or external Customers. An Offer aggregates one or more MCMFeatures, MCMBusinessTerms, and other business logic.

It is the superclass for three subclasses – MCMProductOffer, MCMServiceOffer, and MCMResourceOffer. This enables features from MCMProduct, MCMService, and MCMResource, respectively, to be used to construct a business offering (a subclass of MCMOffer).

The structure of MCMOffer parallels that of MCMFeature; this markedly simplifies usability of both. Note that the MCMOfferHasMCMDefinitionDecorator aggregation is part of a pattern that enables MCMOffers to be made up of a combination of different MCMFeatures and MCMBusinessTerms. Since both MCMFeature and MCMBusinessTerm are subclasses of MCMDefinitionDecorator, both can be added dynamically at runtime to an MCMOffer. This addresses the use case of changing an order in flight without having to recompile and redeploy.

At this time, no attributes are defined for the MCMOffer class. Note that concepts such as a time period that defines the starting and ending time that this MCMOffer is valid for are realized as associated MCMMetadata objects.

Table 34 defines the operations for this class:

Operation Name	Description
getMCMBusinessTermList() : MCMBusinessTerm[1..*]	This operation returns the set of MCMBusinessTerm objects that are currently contained by this MCMOffer object. The return value is an array of one or more objects of type MCMBusinessTerm. This operation follows all instances of the MCMOfferHasMCMDefinitionDecorator aggregation (i.e., from this MCMOffer object to each MCMBusinessTerm object that it contains), and returns the aggregated MCMBusinessTerm objects as an array. This operation does not return any MCMFeature objects that are decorating the set of MCMBusinessTerm objects; if that is desired, use the getMCMFeature operation for each MCMBusinessTerm object that is returned. **[D102]** If this object does not contain any MCMBusinessTerm objects, then a NULL MCMBusinessTerm object **SHOULD** be returned.
setMCMBusinessTermList(in newBusinessTermList : MCMBusinessTerm [1..*])	This operation defines the complete set of MCMBusinessTerm objects that will be aggregated by this MCMOffer object. This operation takes a single input parameter, called newBusinessTermList, which is an array of one or more MCMBusinessTerm objects; this represents the new MCMBusinessTerm objects that will be aggregated by this MCMOffer object. Any existing MCMBusinessTerm objects that are aggregated by this MCMOffer object are first deleted. This is done by deleting each instance of the MCMOfferHasMCMDefinitionDecorator aggregation (and its association class), which disconnects the MCMBusinessTerm object from this MCMOffer object. Note that the MCMBusinessTerm object, and any decorating MCMFeature objects, are NOT deleted. This operation then creates a set of aggregations (i.e., an instance of MCMOfferHasMCMDefinitionDecorator) between this particular MCMOffer object and the set of MCMBusinessTerm objects identified in the input parameter. However, this operation does not create any decorating MCMFeature objects for a given MCMBusinessTerm object.

	[D103] Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMOfferHasMCMDefinitionDecoratorDetail class).
setMCMBusinessTermPartialList (in newBusinessTermPartialList: MCMBusinessTerm [1..*])	This operation defines a set of one or more MCMBusinessTerm objects that will be aggregated by this particular MCMOffer object WITHOUT affecting any other existing MCMBusinessTerm objects or the objects that are decorating them. This operation takes a single input parameter, called newBusinessTermItemPartialList, which is an array of one or more MCMBusinessTerm objects. This operation creates a set of aggregations (i.e., an instance of MCMOfferHasMCMDefinitionDecorator) between this particular MCMOffer object and the set of MCMBusinessTerm objects identified in the input parameter. This operation does not create any decorating MCMFeature objects for a given MCMBusinessTerm object. **[D104]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMOfferHasMCMDefinitionDecoratorDetail class).
delMCMBusinessTermList()	This operation disconnects ALL instances of contained MCMBusinessTerm objects for this particular MCMOffer object. This operation first removes the association class, and second, removes the aggregation, between this MCMOffer object and each MCMBusinessTerm object that is attached to this MCMOffer object. This operation does not affect either the MCMBusinessTerm object, or any MCMFeature objects that are decorating each MCMBusinessTerm object.
delMCMBusinessTermPartialList (in newBusinessTermPartialList: MCMBusinessTerm[1..*])	This operation disconnects a set of MCMBusinessTerm objects from being contained by this particular MCMOffer object. This operation takes a single input parameter, called newBusinessTermPartialList, which is an array of one or more MCMBusinessTerm objects. This operation first, removes the association class and second, removes the aggregation, between each MCMBusinessTerm object specified in the input parameter and this MCMOffer object. This operation does not affect either the MCMBusinessTerm object, or any MCMFeature objects that are decorating each MCMBusinessTerm object, specified in the input parameter. In other words, this operation disconnects each MCMBusinessTerm (and any MCMFeature objects that are decorating it) that is specified in the input parameter from this MCMOffer object.

Table 34. Operations of the MCMOffer Class

At this time, a single aggregation is defined for the MCMOffer class. This aggregation is named MCMOfferHasMCMDefinitionDecorator, and defines the set of MCMDefinitionDecorators that are contained by this particular MCMOffer object. The multiplicity of this aggregation is 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMDefinitionDecorator objects can be aggregated by this particular MCMOffer object. Note that the cardinality on the part side (MCMDefinitionDecorator) is 0..*; this enables an MCMOffer object to be defined without having to define an associated MCMDefinitionDecorator object for it to aggregate. The semantics of this aggregation are defined by the MCMOfferHasMCMDefinitionDecoratorDetail association class. This enables the management system to control which set of concrete subclasses of MCMDefinitionDecorator (e.g., a concrete subclass of MCMFeature) are contained by this particular (concrete subclass of) MCMOffer.

The Policy Pattern may be used to control which specific concrete subclasses of MCMDefinitionDecorator are used to wrap a given concrete subclass of MCMOffer for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.8.2.9 MCMProductOffer Class Definition

This is a concrete class, and specializes MCMOffer. It defines a business offering, typically based on demographics, to sell MCMProducts to MCMCustomers. It consists of a set of features (defined by one or more MCMProductFeatures, MCMServiceFeatures, and MCMResourceFeatures), MCMBusinessTerms, and other functionality that make up an MCMProduct.

Table 35 defines the attributes for the MCMProductOffer class.

Attribute Name	Mandatory?	Description
mcmProductOfferType : MCMProductOrderType[1..1]	YES	This is a mandatory enumeration that defines the type of MCMProduct that this instance is. Valid values are defined by the MCMProductOrderType enumeration. Note that only *one* MCMProduct can be ordered in a single order request. Values include: 0: ERROR 1: INIT 2: UniProduct 3: AccessELineProduct

Table 35. Attributes of the MCMProductOffer Class

Table 36 defines the operations for this class:

Operation Name	Description
getMCMProductOffer-Type() : MCMProductOr-derType[1..1]	This operation returns the type of MCMProduct that this instance is. There are no input parameters to this operation. Valid values are defined by the MCMProductOrderType enumeration.
setMCMProductOrder-Type(in newMCMProduct: MCMProductOrder-Type[1..1])	This operation defines the type of MCMProduct that this instance is. There is a single input parameter, called new-MCMProduct, which is of type MCMProductOrderType. Valid values are defined by the MCMProductOrderType enumeration.

Table 36. Operations of the MCMProductOffer Class

At this time, a single aggregation is defined for MCMProductOffer. This is shown in Figure 14.

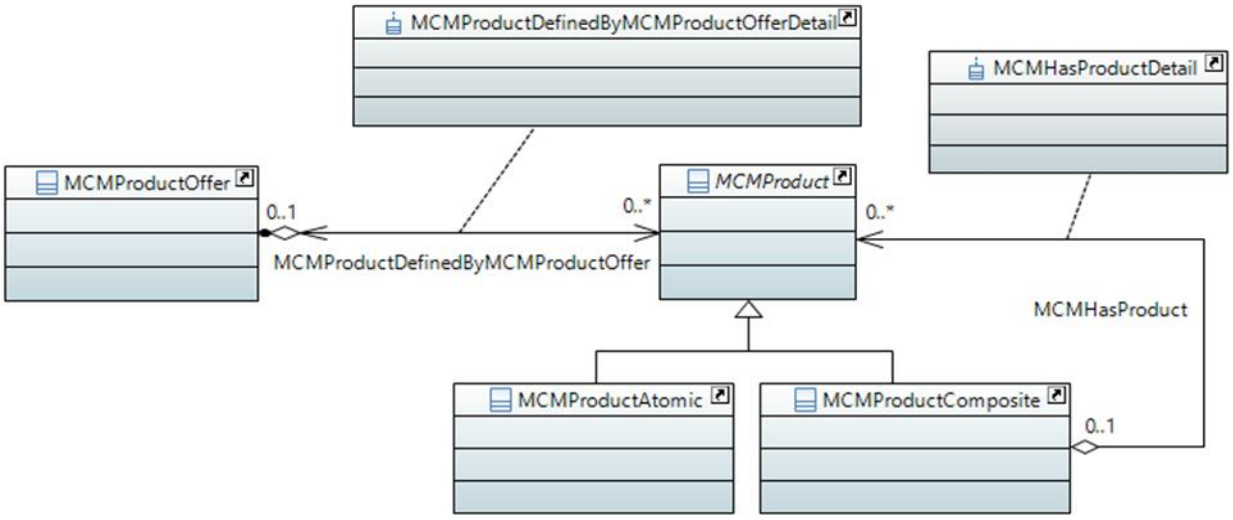

Figure 14. The MCMProductDefinedByMCMProductOffer Aggregation

The MCMProductDefinedByMCMProductOffer aggregation specifies the set of MCMProducts whose characteristics and behavior are defined by this set of MCMProductOffers. The multiplicity of this aggregation is 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMProduct objects can be aggregated by this particular MCMProductOffer object. Note that the cardinality on the part side (MCMProduct) is 0..*; this enables an MCMProductOffer object to be defined without having to define an associated MCMProduct object for it to aggregate. For example, different MCMProductOffers could be used to specify the

customer premise equipment, connectivity services, and application guarantees of a bundled MCMProduct offering.

The semantics of this aggregation are defined by the MCMProductDefinedByMCMProductOfferDetail association class. This enables the management system to control which set of concrete subclasses of MCMProduct are defined by this particular MCMProductOffer class. The Policy Pattern may be used to control which specific MCMProduct objects are affected by which MCMProductOffer objects for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.8.2.10 MCMServiceOffer Class Definition

This is a concrete class, and specializes MCMOffer. It defines a business offering, typically based on demographics, to provide Services to Consumers. It defines the characteristics and behavior of Services that are invariant across all MCMOrderedService and MCMInternalService instances. Users of these Services can be internal or external Applications, Services, other Resources, PartyRoles, and other appropriate Entities. The characteristics and behavior of this class are application-specific, so in this definition of the MCM, the purpose of this class is solely to define the concept for different applications using the MCM to be able to create a common subclass for interoperability.

At this time, no attributes are defined for the MCMServiceOffer class.

At this time, no operations are defined for the MCMServiceOffer class.

At this time, a single aggregation is defined for MCMServiceOffer. This is shown in Figure 15.

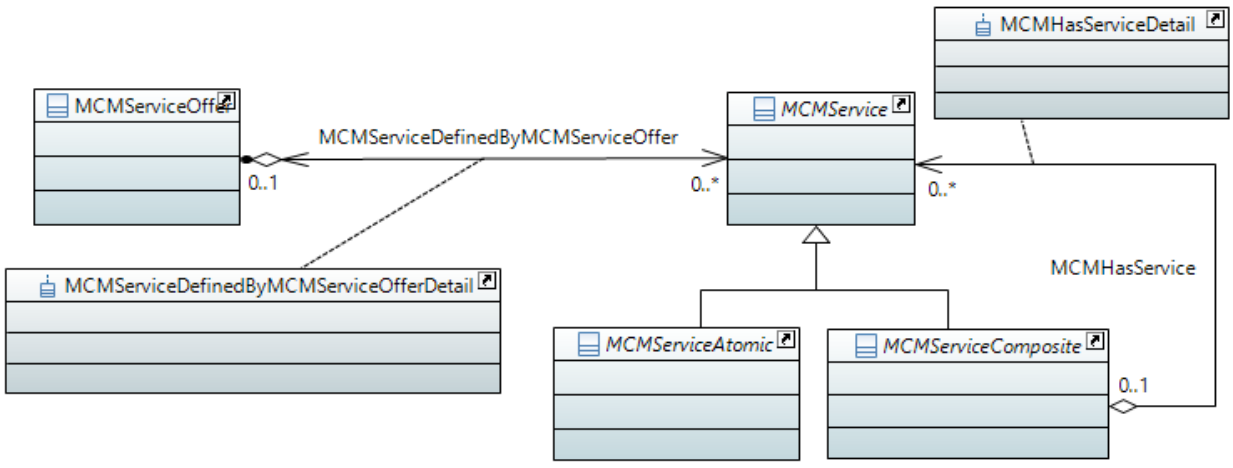

Figure 15. The MCMServiceDefinedByMCMService Offer Aggregation

The MCMServiceDefinedByMCMServiceOffer aggregation specifies the set of MCMServices whose characteristics and behavior are defined by this set of MCMServiceOffers. The multiplicity of this aggregation is 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMService objects can be aggregated by this particular MCMServiceOffer object. Note that the cardinality on the part side (MCMService) is 0..*; this enables an MCMServiceOffer object to be defined without having to define an associated MCMService object for it to aggregate. For example, different MCMServiceOffers could be used to specify different application performance, response, and other behavior of an MCMService.

The semantics of this aggregation are defined by the MCMServiceDefinedByMCMServiceOfferDetail association class. This enables the management system to control which set of concrete subclasses of MCMService are defined by this particular MCMServiceOffer class. The Policy Pattern may be used to control which specific MCMService objects are affected by which MCMServiceOffer objects for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.8.2.11 MCMResourceOffer Class Definition

This is a concrete class, and specializes MCMOffer. It defines a business offering, typically based on demographics, to provide Resources to internal or external Applications, Services, other Resources, PartyRoles, and other appropriate Entities. It defines the characteristics and behavior of Resources that are invariant across all concrete subclasses of Resource. The characteristics and behavior of this class are application-specific, so in this definition of the MCM, the purpose of this class is solely to define the concept for different applications using the MCM to be able to create a common subclass for interoperability.

At this time, no attributes are defined for the MCMResourceOffer class.

At this time, no operations are defined for the MCMResourceOffer class.

At this time, a single aggregation is defined for MCMResourceOffer. This is shown in Figure 16.

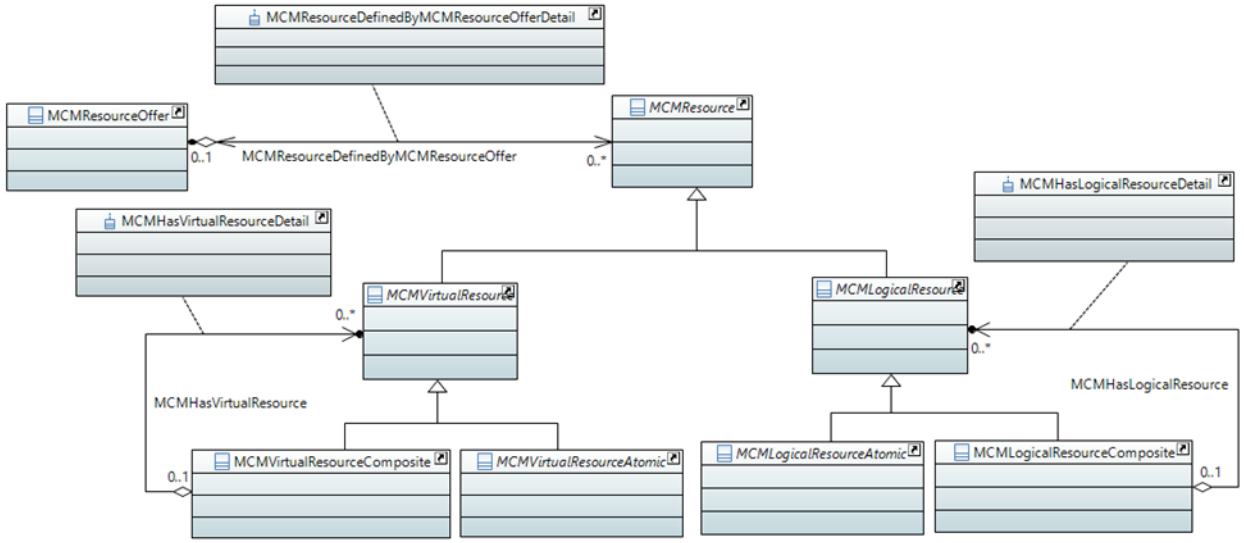

Figure 16. The MCMResourceDefinedByMCMResourceOffer Aggregation

The MCMResouceDefinedByMCMResourceOffer aggregation specifies the set of MCMResources whose characteristics and behavior are defined by this set of MCMResourceOffers. The multiplicity of this aggregation is 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMResource objects can be aggregated by this particular MCMResourceOffer object. Note that the cardinality on the part side (MCMResource) is 0..*; this enables an MCMResourceOffer object to be defined without having to define an associated MCMResource object for it to aggregateFor example, different MCMResourceOffers could be used to define the storage, computing power, and connectivity required by a given MCMService.

The semantics of this aggregation are defined by the MCMResourceDefinedByMCMResourceOfferDetail association class. This enables the management system to control which set of concrete subclasses of MCMResource objects are defined by this particular MCMResourceOffer object. The Policy Pattern may be used to control which specific MCMResource objects are affected by which MCMResourceOffer objects for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.8.3 MCMPolicyObject Class Definition

This is an abstract class, and specializes MCMManagedEntity. It is the root of the MCM Policy system (i.e., all other classes in the model are subclasses of this class) except for any Metadata objects associated with a Policy object (these are defined in the MCMMetaData class hierarchy). This simplifies code generation and reusability. It also enables different types of MCMMetadata objects to be attached to any appropriate subclass of MCMPolicyObject.

An MCMPolicyObject may be qualified by a set of zero or more MCMMetadata objects. Two different aggregations are defined for this purpose. The MCMPolicyObjectHasMCMMetaData aggregation is used to relate generic metadata (e.g., version and best current practice information) to an MCMPolicyObject. In contrast, the MCMPolicyObjectHasMCMPolicyMetaData aggregation enables different types of policy-specific metadata to be associated with an MCMPolicyObject.

7.8.4 MCMProduct Class Hierarchy

The MCMProduct class hierarchy is shown in Figure 17.

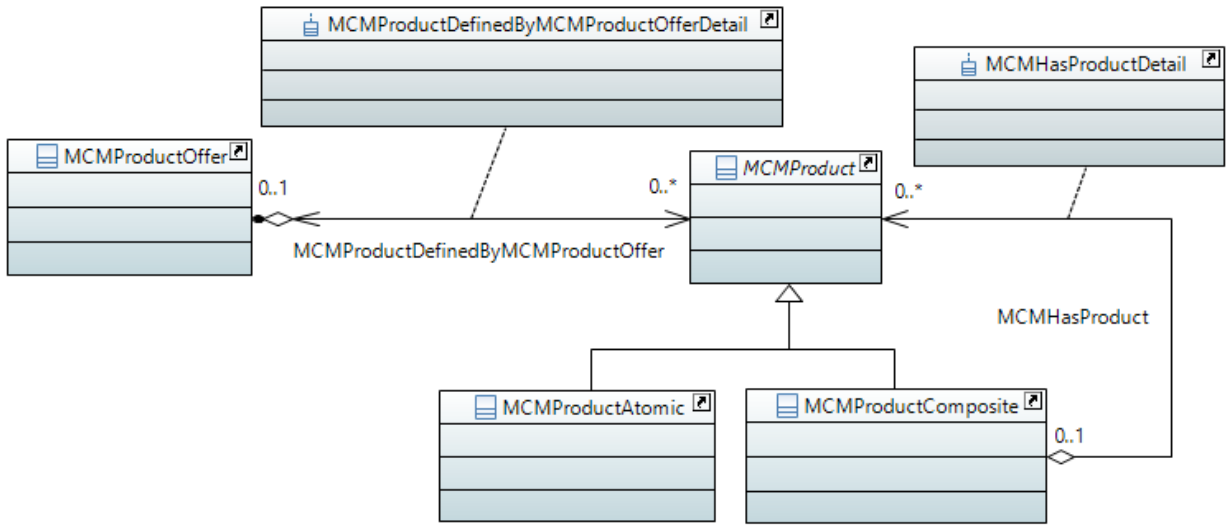

Figure 17. The MCMProduct Class Hierarchy

7.8.4.1 MCMProduct Class Definition

This is an abstract class, and specializes MCMManagedEntity. This defines the set of goods and services, offered to a market, by a set of MCMParties, which is playing a set of appropriate MCMPartyRoles. MCMProducts are purchased by an MCMCustomer, which is a specific type of MCMPartyRole.

Each such Product is based on an MCMProductOffer, even if it uses shared Resources and/or Services, and results in a separate instance of the MCMProduct class.

Note that an MCMProduct may exist in a purchased or unpurchased state. For example, it may be exposed to an MCMCustomer using an MCMCatalog.

At this time, no attributes are defined for the MCMProduct class. Most attributes will likely be realized using relationships and/or operations. For example, the *usage* of an MCMProduct can be considered from two viewpoints: (1) how much *content* is left (e.g., a subscription limits down-loads to 1Gb/months, and the current usage is 750Mb), and (2) how much *time* is left (e.g., the MCMProduct is being used on a time-limited subscription). In either of these cases, an attribute is inappropriate, since one or more computations and information from one or more relationships are required to provide a value. In addition, the MCMProduct itself doesn't "know" how much usage is incurred, but can find out (e.g., by using an operation).

Table 37 defines the operations for this class:

Operation Name	Description
getMCMProductParent() : MCMProductComposite[1..1]	This operation returns the parent of this MCMProduct object. The parent is be of type MCMProductComposite. This operation takes no input parameters. **[D105]** If this MCMProduct object has no parent, then a NULL MCMProduct object **SHOULD** be returned.
setMCMProductParent(in newParent : MCMProductComposite[1..1])	This operation defines the parent of this MCMProduct object. The parent is defined in the input parameter, called newParent, and is be of type MCMProductComposite. **[D106]** If this MCMProduct object already has a parent, then an exception **SHOULD** be raised.

Table 37. Operations of the MCMProduct Class

The MCMProduct class participates in two aggregations, which are shown in Figure 17. MCMHasProduct is defined in section 7.8.4.3, and MCMProductDefinedByMCMProductOffer is defined in section 7.8.2.9.

7.8.4.2 MCMProductAtomic Class Definition

This is a concrete class, and specializes MCMProduct. In addition, each MCMProductAtomic has characteristics and behavior that are externally visible.

[R70] This class **MUST NOT** contain another MCMProduct object.

At this time, no attributes are defined for the MCMProductAtomic class.

At this time, no operations are defined for the MCMProductAtomic class.

At this time, no relationships are defined for the MCMProductAtomic class.

7.8.4.3 MCMProductComposite Class Definition

This is a concrete class, and specializes MCMProduct. This class represents a set of related MCMProduct objects that are organized into a tree structure.

[O58] Each MCMProduct **MAY** contain zero or more MCMProductAtomic and/or zero or more MCMProductComposite objects.

At this time, no attributes are defined for the MCMProductComposite class. Most attributes will likely be realized using relationships and/or operations. For example, a query to an instance of the MCMProductComposite class to provide its set of contained MCMProducts (e.g., the individual MCMProducts that represent a triple-play or quad-play Product) will be done by using class operations; the MCMProductComposite instance will query each of its contained MCMProducts

(which will in turn call their operations to acquire their MCMProducts), aggregate and organize the information, and provide that information in its operation response.

Table 38 defines the operations for this class.

Operation Name	Description
getMCMProductChildList() : MCMProduct[1..*]	This operation returns the set of all MCMProduct objects that are contained in this specific MCMProductComposite object. There are no input parameters to this operation. This operation returns a list of zero or more MCMProduct objects (i.e., the list is made up of MCMProductAtomic and/or MCMProductComposite objects). **[D107]** If this MCMProductComposite object has no children, then it **SHOULD** return a NULL MCMProductComposite object.
setMCMProductChildList (in childObjectList : MCMProduct[1..*])	This operation defines a set of MCMProduct objects that will be contained by this particular MCMProductComposite object. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMProduct objects (i.e., one or more MCMProductAtomic and/or MCMProductComposite objects). This operation first deletes any existing contained MCMProduct objects (and their aggregations and association classes), and then instantiates a new set of MCMProduct objects; in doing so, each MCMProduct object is contained within this particular MCMProductComposite object by creating an instance of the MCMHasProduct aggregation. **[D108]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasProductDetail association class).
setMCMProductChildPartialList (in childObjectList : MCMProduct[1..*])	This operation defines a set of one or more MCMProduct objects that should be contained within this particular MCMProductComposite object WITHOUT affecting any other existing contained MCMProduct objects or the objects that are contained in them. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMProduct objects. This operation creates a set of aggregations between this particular MCMProductComposite object and each of the MCMProduct objects identified in the childObjectList. **[D109]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasProductDetail association class).

delMCMProductChildList()	This operation deletes ALL contained MCMProduct objects of this particular MCMProductComposite object. This has the effect of first, removing the association class, and second, removing the aggregation, between this MCMProductComposite object and each MCMProduct object that is contained in this MCMProductComposite object. This operation has no input parameters.
delMCMProductPartialChildList (in childObjectList : MCMProduct[1..*])	This operation deletes a set of MCMProduct objects from this particular MCMProductComposite object. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMProduct objects. This has the effect of first, removing the association class and second, removing the aggregation, between each MCMProduct object specified in the input parameter and this MCMProductComposite object. Note that all other aggregations between this MCMProductComposite and other MCMProduct objects that are not identified in the input parameter are NOT affected.

Table 38. Operations of the MCMProductComposite Class

The MCMProductComposite class defines a single aggregation, called MCMHasProduct. This aggregation is used to define the set of MCMProducts that are contained within this particular MCMProductComposite. Its multiplicity is defined to be 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMProduct objects can be aggregated by this particular MCMProductComposite object. Note that the cardinality on the part side (MCMProduct) is 0..*; this enables an MCMProductComposite object to be defined without having to define an associated MCMProduct object for it to aggregate.

The semantics of the MCMHasProduct aggregation is realized using an association class, called MCMHasProductDetail. This enables the semantics of the MCMHasProduct aggregation to be realized using the attributes, operations, and relationships of the MCMHasProductDetail association class.

The Policy Pattern may be used to control which specific MCMProduct objects are contained within a given MCMProductComposite object for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.8.5 MCMService Class Hierarchy

The MCMService class hierarchy is shown in Figure 18.

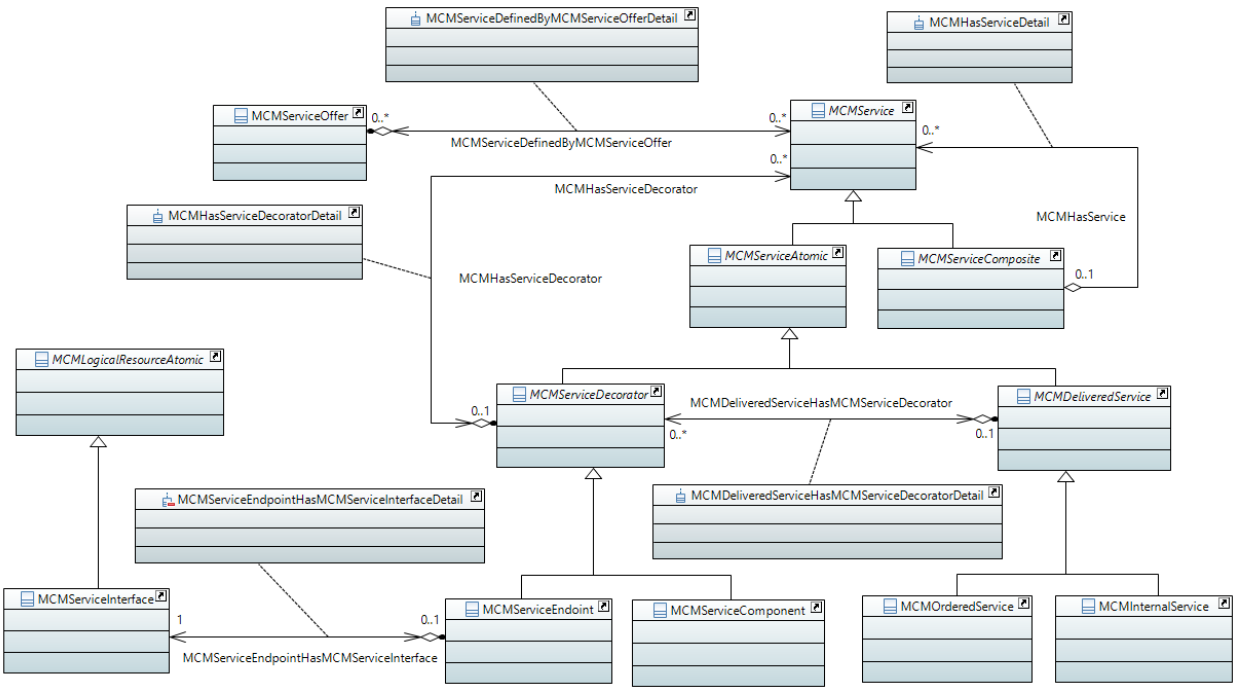

Figure 18. The MCMService Class Hierarchy

7.8.5.1 MCMService Class Definition

This is an abstract class, and specializes MCMManagedEntity. It represents functionality that can be used by different internal and external users (e.g., a management system and a Customer, respectively) for different purposes. Services may be consumed by other Services, but not by Resources. A Service has a distinct state.

At this time, no attributes are defined for the MCMService class. Most attributes will likely be realized using relationships and/or operations. For example, the *usage* of an MCMService can be considered from two viewpoints: (1) how much *content* is left (e.g., a subscription limits downloads to 1Gb/months, and the current usage is 750Mb), and (2) how much *time* is left (e.g., the MCMService is being used on a time-limited subscription). In this example, the MCMService itself doesn't "know" how much usage is incurred, but can find out (e.g., by using a operation). As another example, an MCMManagedEntity may need to know the status of all of the MCMServiceEndpoints (see section 7.8.5.9) and MCMServiceComponents (see section 7.8.5.8) that are associated with a particular MCMService. In either of these cases, an attribute is inappropriate, since one or more computations and information from one or more relationships are required to provide a value. This is exacerbated in the latter case, since MCMServiceComponents

and MCMServiceEndpoints are both objects that decorate an MCMDeliveredService (see section 7.8.5.4).

Table 39 defines the operations for this class:

Operation Name	Description
getMCMServiceParent() : MCMServiceComposite[1..1]	This operation returns the parent of this MCMService object. The parent MUST be of type MCMServiceComposite. This operation takes no input parameters. **[D110]** If this MCMService object has no parent, then a NULL MCMService object **SHOULD** be returned.
setMCMServiceParent(in newParent : MCMServiceComposite [1..1])	This operation defines the parent of this MCMService object. The parent is defined in the input parameter, called newParent, and MUST be of type MCMServiceComposite. **[D111]** If this MCMService object already has a parent, then an exception **SHOULD** be raised.

Table 39. Operations of the MCMService Class

The MCMService class participates in three aggregations, as shown in Figure 18. The MCMServiceDefinedByServiceOffer aggregation is defined in section 7.8.2.10, the MCMHasServiceDecorator is defined in section 7.8.5.7, and the MCMHasService aggregation is defined in section 7.8.5.3.

7.8.5.2 MCMServiceAtomic Class Definition

This is an abstract class, and specializes MCMService. This class represents stand-alone MCMService objects.

[R71] This object **MUST NOT** contain another MCMService object.

At this time, no attributes are defined for the MCMServiceAtomic class.

At this time, no operations are defined for the MCMServiceAtomic class.

At this time, no relationships are defined for the MCMServiceAtomic class.

7.8.5.3 MCMServiceComposite Class Definition

This is an abstract class, and specializes MCMService. This class represents a set of related MCMServiceComposite objects that are organized into a tree structure.

[O59] Each MCMServiceComposite **MAY** contain zero or more MCMServiceAtomic and/or zero or more MCMServiceComposite objects.

At this time, no attributes are defined for the MCMServiceComposite class. Most attributes will likely be realized using relationships and/or operations. For example, a query to an instance of the MCMServiceComposite class to provide its set of contained MCMServices will be done by using class operations; the MCMServiceComposite instance will query each of its contained MCMServiceAtomic and MCMServiceComposite objects (which will in turn call their operations to acquire their MCMServices), aggregate and organize the information, and provide that information in its operation response. In more detail, the MCMServiceComposite could ask for the set of MCMInternalServices (see section 7.8.5.6) that are used to support an MCMDeliveredService; the set of MCMInternalServices in this example could include Analytics, Traffic Engineering, and other MCMInternalServices that are not visible to the MCMCustomer.

Table 40 defines following operations for this class:

Operation Name	Description
getMCMServiceList() : MCMService[1..*]	This operation returns the set of all MCMService objects that are contained in this specific MCMServiceComposite object. There are no input parameters to this operation. This operation returns a list of zero or more MCMService objects (i.e., the list is made up of MCMServiceAtomic and/or MCMServiceComposite objects). **[O60]** If this object does not contain any MCMService objects, then a NULL MCMService object **SHOULD** be returned.
setMCMServiceList (in childObjectList : MCMService [1..*])	This operation defines a set of MCMService objects that will be contained by this particular MCMServiceComposite object. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMService objects (i.e., one or more MCMServiceAtomic and/or MCMServiceComposite objects). This operation first deletes any existing contained MCMService objects (and their aggregations and association classes), and then instantiates a new set of MCMService objects; in doing so, each MCMService object is contained within this particular MCMServiceComposite object by creating an instance of the MCMHasService aggregation. **[D112]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasServiceDetail association class).
setMCMServicePartialList (in childObjectList : MCMService[1..*])	This operation defines a set of one or more MCMService objects that should be contained within this particular MCMServiceComposite object WITHOUT affecting any other

	existing contained MCMService objects or the objects that are contained in them. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMService objects. This operation creates a set of aggregations between this particular MCMServiceComposite object and each of the MCMService objects identified in the childObjectList. **[D113]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasServiceDetail association class).
delMCMServiceList()	This operation deletes ALL contained MCMService objects of this particular MCMServiceComposite object. This has the effect of first, removing the association class, and second, removing the aggregation, between this MCMServiceComposite object and each MCMService object that is contained in this MCMServiceComposite object. This operation has no input parameters.
delMCMServicePartialList (in childObjectList : MCMService[1..*])	This operation deletes a set of MCMService objects from this particular MCMServiceComposite object. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMService objects. This has the effect of first, removing the association class and second, removing the aggregation, between each MCMService object specified in the input parameter and this MCMServiceComposite object. Note that all other aggregations between this MCMServiceComposite and other MCMService objects that are not identified in the input parameter are NOT affected.

Table 40. Operations for the MCMServiceComposite Class

The MCMServiceComposite class defines a single aggregation, called MCMHasService. This aggregation is used to define the set of MCMServices that are contained within this particular MCMServiceComposite. Its multiplicity is defined to be 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMService objects can be aggregated by this particular MCMServiceComposite object. Note that the cardinality on the part side (MCMService) is 0..*; this enables an MCMServiceComposite object to be defined without having to define an associated MCMService object for it to aggregate.

The semantics of the MCMHasService aggregation is realized using an association class, called MCMHasServiceDetail. This enables the semantics of the MCMHasService aggregation to be realized using the attributes, operations, and relationships of the MCMHasServiceDetail association class.

The Policy Pattern may be used to control which specific MCMService objects are contained within a given MCMServiceComposite object for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.8.5.4 MCMDeliveredService Class Definition

This is an abstract class, and specializes MCMServiceAtomic. It represents MCMServices that are used by consumers. Its functionality is defined by a set of one or more MCMServiceDecorators.

[R72] An operational MCMDeliveredService **MUST** have a set of MCMServiceDecorators.

At this time, no attributes are defined for the MCMDeliveredService class.

Table 41 defines following operations for this class:

Operation Name	Description
getMCMServiceComponentList() : MCMServiceComponent[1..*]	This operation returns the set of MCMServiceComponent objects for this MCMDeliveredService object. This operation takes no input parameters. First, this operation determines if there are any instances of the MCMDeliveredServiceHasMCMServiceDecorator aggregation for this particular MCMDeliveredService object. For each instance of the MCMDeliveredServiceHasMCMServiceDecorator aggregation, the instance is inspected to see if the decorating object is of type MCMServiceComponent. All MCMServiceComponent objects found (for all aggregation instances) are returned as an array. **[D114]** If there are no instances of the MCMDeliveredServiceHasMCMServiceDecorator aggregation, then a NULL MCMServiceComponent object **SHOULD** be returned. **[D115]** If an instance of the MCMDeliveredServiceHasMCMServiceDecorator aggregation exists, but none of the decorating objects are of type MCMServiceComponent, then a NULL MCMServiceComponent object **SHOULD** be returned.
setMCMServiceComponentList (in newServiceComponentList : MCMServiceComponent[1..*])	This operation defines a set of MCMServiceComponent objects that will decorate this MCMDeliveredService object. Note that this operation will first disconnect all existing MCMServiceComponent objects that are aggregated by this MCMDeliveredService object. This is done by first, removing

	the MCMDeliveredSeviceHasMCMServiceDecoratorDetail association class, and second, deleting the MCMDeliveredServiceHasMCMServiceDecorator aggregation, for every existing MCMComponent that is currently aggregated by this MCMDeliveredService. Note that this operation does not delete the MCMServiceComponent, it simply deletes the aggregation (and association class). Once this is done, for each MCMServiceComponent in the input parameter, an instance of the MCMDeliveredServiceHasMCMServiceDecorator aggregation is created. **[D116]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMDeliveredServiceHasMCMServiceDecoratorDetail association class).
setMCMServiceComponentPartialList (in newServiceComponentList : MCMServiceComponent[1..*])	This operation will add a set of MCMServiceComponent objects that will decorate this MCMDeliveredService object WITHOUT affecting any existing MCMServiceComponent objects. This operation takes a single input parameter, called newServiceComponentList, which is an array of MCMServiceComponent objects. For each MCMServiceComponent in the input parameter, an instance of the MCMDeliveredServiceHasMCMServiceDecorator aggregation is created. **[D117]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMDeliveredServiceHasMCMServiceDecoratorDetail association class).
delMCMServiceComponentList()	This operation disconnects all MCMServiceComponent object instances from this MCMDeliveredService object. This operation takes no input parameters. First, this operation determines if there are any instances of the MCMDeliveredServiceHasMCMServiceDecorator aggregation for this particular MCMDeliveredService object. For each instance of the MCMDeliveredServiceHasMCMServiceDecorator aggregation, the instance is inspected to see if the decorating object is of type MCMServiceComponent. If so, then the aggregation and its association class are both deleted.
delMCMServiceComponentPartialList (in newServiceComponentList : MCMServiceComponent[1..*])	This operation deletes a set of MCMServiceComponent objects from this MCMDeliveredService object WITHOUT affecting any other existing contained MCMServiceComponent objects or the objects that are contained in them. This operation

	takes a single input parameter, called newServiceComponentList, which is an array of one or more MCMServiceComponent objects. This has the effect of first, removing the association class and second, removing the aggregation, between each MCMServiceComponent object specified in the input parameter and this MCMDeliveredService object. Note that all other aggregations between this MCMDeliveredService and other MCMServiceComponent objects that are not identified in the input parameter are NOT affected.
getMCMServiceEndpointList() : MCMServiceEndpoint[1..*]	This operation returns the set of MCMServiceEndpoint objects for this MCMDeliveredService object. This operation takes no input parameters. First, this operation determines if there are any instances of the MCMDeliveredServiceHasMCMServiceDecorator aggregation for this particular MCMDeliveredService object. Then, for each instance of the MCMDeliveredServiceHasMCMServiceDecorator aggregation, the instance is inspected to see if the decorating object is of type MCMServiceEndpoint . All MCMServiceEndpoint objects found (for all aggregation instances) are returned as an array. **[D118]** If there are no instances of the MCMDeliveredServiceHasMCMServiceDecorator aggregation, then a NULL MCMServiceComponent object **SHOULD** be returned. **[D119]** If an instance of the MCMDeliveredServiceHasMCMServiceDecorator aggregation exists, but none of the decorating objects are of type MCMServiceEndpoint, then a NULL MCMServiceComponent object **SHOULD** be returned.
setMCMServiceComponentList (in newServiceComponentList : MCMServiceEndpoint[1..*])	This operation defines a set of MCMServiceEndpoint objects that will decorate this MCMDeliveredService object. Note that this operation will first disconnect all existing MCMServiceEndpoint objects that are aggregated by this MCMDeliveredService object. This is done by first, removing the MCMDeliveredServiceHasMCMServiceDecoratorDetail association class, and second, deleting the MCMDeliveredServiceHasMCMServiceDecorator aggregation, for every existing MCMComponent that is currently aggregated by this MCMDeliveredService. Note that this operation does not delete the MCMServiceEndpoint object, it simply deletes the aggregation (and association class).

	Once this is done, for each MCMServiceComponent in the input parameter, an instance of the MCMDeliveredServiceHasMCMServiceDecorator aggregation is created. **[D120]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMDeliveredServiceHasMCMServiceDecoratorDetail association class).
setMCMServiceComponentPartialList (in newServiceComponentList : MCMServiceEndpoint[1..*])	This operation will add a set of MCMServiceEndpoint objects that will decorate this MCMDeliveredService object WITHOUT affecting any existing MCMServiceEndpoint objects. This operation takes a single input parameter, called newServiceComponentList, which is an array of MCMServiceComponent objects. For each MCMServiceComponent in the input parameter, an instance of the MCMDeliveredServiceHasMCMServiceDecorator aggregation is created. **[D121]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMDeliveredServiceHasMCMServiceDecoratorDetail association class).

Table 41. Operations for the MCMDeliveredService Class

At this time, a single aggregation is defined for the MCMDeliveredService class. This aggregation is named MCMDeliveredServiceHasMCMServiceDecorator, and defines the set of MCMServiceDecorators that are contained by this particular MCMDeliveredService object. The multiplicity of this aggregation is 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMServiceDecorator objects can be aggregated by this particular MCMDeliveredService object. Note that the cardinality on the part side (MCMServiceDecorator) is 0..*; this enables an MCMDeliveredService object to be defined without having to define an associated MCMServiceDecorator object for it to aggregate.

The semantics of this aggregation are defined by the MCMDeliveredServiceHasMCMServiceDecoratorDetail association class. This enables the management system to control which set of concrete subclasses of MCMServiceDecorators are contained by this particular MCMDeliveredService class. The Policy Pattern may be used to control which specific MCMServiceDecorator objects are contained within a given MCMDeliveredService object for a particular context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.8.5.5 MCMOrderedService Class Definition

This is a concrete class, and specializes MCMDeliveredService. It represents an MCMService that is used by an MCMProduct. This MCMService is realized within the Service Provider's and/or Partners' infrastructure, but is delivered to an external entity (e.g., a Customer).

Note the difference between MCMOrderedService and MCMInternalService. The former is an MCMService delivered to a Customer, while the latter is an MCMService that is necessary for the proper operation of the infrastructure of the Service Provider or Partner.

At this time, no attributes are defined for the MCMOrderedService class.

At this time, no operations are defined for the MCMOrderedService class.

At this time, no relationships are defined for the MCMOrderedService class.

7.8.5.6 MCMInternalService Class Definition

This is a concrete class, and specializes MCMDeliveredService. It represents an MCMService that is necessary for the proper operation of the Service Provider's infrastructure. For example, it could represent an internal telemetry collecting service, or an internal analytics service, or an internal service to configure an object; in all of these examples, "internal" means that the service is not visible to external entities outside of the MCMDomain see section 7.6) that it exists in.

At this time, no attributes are defined for the MCMInternalService class.

At this time, no operations are defined for the MCMInternalService class.

At this time, no relationships are defined for the MCMInternalService class.

7.8.5.7 MCMServiceDecorator Class Definition

This is an abstract class, and specializes MCMServiceAtomic. It applies the decorator pattern to MCMServiceAtomic objects. It enables all or part of one or more concrete subclasses of MCMServiceDecorator to "wrap" another concrete subclass of MCMServiceAtomic. For example. any concrete subclass of MCMDeliveredService may be wrapped by any concrete subclass of MCMServiceDecorator.

At this time, no attributes are defined for the MCMServiceDecorator class.

Table 42 defines following operations for this class:

Operation Name	Description
getMCMServiceComponentList() : MCMServiceComponent[1..*]	This operation returns the set of MCMServiceComponent objects that are decorating this MCMDeliveredService object. There are no input parameters. **[D122]** If this MCMDeliveredService object is not decorated by any MCMServiceComponent objects, then a NULL MCMServiceComponent object **SHOULD** be returned.
setMCMServiceComponentList (in newDecoratorList : MCMServiceComponent[1..*])	This operation defines the set of MCMServiceComponent objects that will decorate this MCMDeliveredService object. This operation takes a single input parameter, called newDecoratorList, which is of type MCMServiceComponent. This operation creates a set of aggregations between this particular MCMDeliveredService object and the set of MCMServiceComponent objects identified in the input parameter. Note that this operation first deletes any existing MCMServiceComponent objects (and their aggregations and association classes) that decorate this MCMDeliveredService object, and then instantiates a new set of MCMServiceComponent objects; in doing so, each MCMServiceComponent object is attached to this particular MCMDeliveredService object by first, creating an instance of the MCMDeliveredServiceHasMCMServiceDecorator aggregation, and second, realizing that aggregation instance as an association class. **[D123]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMDeliveredServiceHasMCMServiceDecorator association class).
setMCMServiceComponentPartialList (in newDecoratorList : MCMServiceComponent[1..*])	This operation defines a set of one or more MCMServiceComponent objects that will decorate this MCMDeliveredService object WITHOUT affecting any other existing MCMServiceComponent objects that are decorating this MCMDeliveredService object. This operation takes a single input parameter, called newDecoratorList, which is an array of one or more MCMServiceComponent objects. This operation creates a set of aggregations between this particular MCMDeliveredService object and the set of MCMServiceComponent objects identified in the input parameter. **[D124]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMDeliveredServiceHasMCMServiceDecorator association class).

delMCMServiceComponentList()	This operation deletes ALL MCMServiceComponent object instances that are decorating this MCMDeliveredService object. This operation first removes the association class, and second, removes the aggregation, between this MCMDeliveredService object and each MCMServiceComponent object that is decorating this MCMDeliveredService object. This operation has no input parameters. This operation does not delete any of the MCMServiceComponent objects; it simply disconnects them from the MCMDeliveredService that they were decorating.
delMCMServiceComponentPartialList (in newDecoratorList : MCMServiceComponent[1..*])	This operation deletes a set of MCMServiceComponent objects that are decorating this particular MCMDeliveredService object. This operation takes a single input parameter, called newDecoratorList, which is an array of one or more MCMServiceComponent objects. This operation first removes the association class and second, removes the aggregation, between each MCMServiceComponent object specified in the input parameter and this MCMDeliveredService object. Note that all other aggregations between this MCMDeliveredService object and other MCMServiceComponent objects that are not specified in the input parameter are NOT affected.
getMCMServiceEndpointList() : MCMServiceEndpoint [1..*]	This operation returns the set of MCMServiceEndpoint objects that are decorating this MCMDeliveredService object. There are no input parameters. **[D125]** If this MCMDeliveredService object is not decorated by any MCMServiceEndpoint objects, then a NULL MCMServiceEndpoint object **SHOULD** be returned.
setMCMServiceEndpointList (in newDecoratorList : MCMServiceEndpoint[1..*])	This operation defines the set of MCMServiceEndpoint objects that will decorate this MCMDeliveredService object. This operation takes a single input parameter, called newDecoratorList, which is of type MCMServiceEndpoint. This operation creates a set of aggregations between this particular MCMDeliveredService object and the set of MCMServiceEndpoint objects identified in the input parameter. Note that this operation first deletes any existing MCMServiceEndpoint objects (and their aggregations and association classes) that decorate this MCMDeliveredService object, and then instantiates a new set of MCMServiceEndpoint objects; in doing so, each MCMServiceEndpoint object is attached to this particular MCMDeliveredService object by first, creating an instance of

	the MCMDeliveredServiceHasMCMServiceDecorator aggregation, and second, realizing that aggregation instance as an association class. **[D126]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMDeliveredServiceHasMCMServiceDecorator association class).
setMCMServiceEndpoint-PartialList (in newDecoratorList : MCMServiceEndpoint[1..*])	This operation defines a set of one or more MCMServiceEndpoint objects that will decorate this MCMDeliveredService object WITHOUT affecting any other existing MCMServiceEndpoint objects that are decorating this MCMDeliveredService object. This operation takes a single input parameter, called newDecoratorList, which is an array of one or more MCMServiceEndpoint objects. This operation creates a set of aggregations between this particular MCMDeliveredService object and the set of MCMServiceEndpoint objects identified in the input parameter. **[D127]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMDeliveredServiceHasMCMServiceDecorator association class).
delMCMServiceEndpoint-List()	This operation deletes ALL MCMServiceEndpoint object instances that are decorating this MCMDeliveredService object. This operation first removes the association class, and second, removes the aggregation, between this MCMDeliveredService object and each MCMServiceEndpoint object that is decorating this MCMDeliveredService object. This operation has no input parameters. This operation does not delete any of the MCMServiceEndpoint objects; it simply disconnects them from the MCMDeliveredService that they were decorating.
delMCMServiceEndpoint-PartialList (in newDecoratorList : MCMServiceEndpoint[1..*])	This operation deletes a set of MCMServiceEndpoint objects that are decorating this particular MCMDeliveredService object. This operation takes a single input parameter, called newDecoratorList, which is an array of one or more MCMServiceEndpoint objects. This operation first removes the association class and second, removes the aggregation, between each MCMServiceEndpoint object specified in the input parameter and this MCMDeliveredService object. Note that all other aggregations between this MCMDeliveredService object and other MCMServiceEndpoint objects that are not specified in the input parameter are NOT affected.

Table 42. Operations of the MCMServiceDecorator Class

At this time, a single aggregation is defined for MCMServiceDecorator. This aggregation is named MCMHasServiceDecorator, and defines the set of MCMServiceDecorator objects that wrap (or decorate) a concrete subclass of MCMService. This enables both MCMServiceAtomic as well as MCMServiceComposite objects to be decorated. The multiplicity of this aggregation is 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMService objects can be decorated (i.e., "wrapped") by this particular MCMServiceDecorator object. Note that the cardinality on the part side (MCMService) is 0..*; this enables an MCMServiceDecorator object to be defined without having to define an associated MCMService object for it to aggregate.

The semantics of this aggregation are defined by the MCMHasServiceDecoratorDetail association class. This enables the management system to control which set of concrete subclasses of MCMServiceDecorator wrap this particular concrete subclass of MCMService. The Policy Pattern may be used to control which specific MCMServiceDecorator objects are allowed to wrap a given MCMService object for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

The MCMServiceDecorator class also participates in another aggregation, called MCMDelivered-ServiceHasMCMServiceDecorator; see section 7.8.5.4.

7.8.5.8 MCMServiceComponent Class Definition

This is a concrete class, and specializes MCMServiceDecorator. It makes available a set of MCMServiceEndpoints, including the behavior of the MCMService between those MCMServiceEndpoints (e.g., its connectivity). An MCMServiceComponent is contained in a single MCMManagementDomain, which is managed independently by the Service Provider.

At this time, no attributes are currently defined for this class.

At this time, no operations are currently defined for this class.

At this time, no relationships are defined for this class.

7.8.5.9 MCMServiceEndpoint Class Definition

This is a concrete class, and specializes MCMServiceDecorator. It represents a (logical) point of delivery of the Service to a consumer, as viewed by the Service. An MCMServiceEndpoint that is in use MUST be associated with a single MCMServiceInterface. An MCMService may exist without an MCMServiceInterface; in such a case, the MCMService is in a planned or some other type of conceptual state, but it is not yet instantiated.

At this time, no attributes are currently defined for this class.

At this time, no operations are currently defined for this class.

At this time, a single aggregation is defined for the MCMServiceEndpoint class. This aggregation is named MCMServiceEndpointHasMCMServiceInterface, and defines the set of

MCMServiceInterfaces that are associated with this particular MCMServiceEndpoint object. The multiplicity of this aggregation is 0..1 - 1. This means that this aggregation is optional (i.e., the “0” part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the “1” part of the 0..1 cardinality), then only one MCMServiceInterface object can be aggregated by this particular MCMServiceEndoint object..

The semantics of this aggregation are defined by the MCMServiceEndpointHasMCMServiceInterfaceDetail association class. This enables the management system to control which MCMServiceInterface is used with a given MCMServiceEndpoint. The Policy Pattern may be used to control which specific MCMServiceInterface object is used with a given MCMServiceEndpoint for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.8.6 MCMResource Class Hierarchy

The MCMResource class hierarchy is shown in Figure 19, Figure 20, and Figure 21.

Figure 19. The MCMResource Class Hierarchy, Part 1

7.8.6.1 MCMResource Class Definition

This is an abstract class, and specializes MCMManagedEntity. It provides capabilities that may be consumed by other MCMResources and/or MCMServices. In addition, an MCMResource may consume other MCMResources. An MCMResource has a distinct state. MCMResources are typically limited in quantity and/or availability. MCMResources may be logical or virtual in nature. Note that physical entities are *not* defined as a subclass of MCMResource, because a physical entity is not inherently manageable. Rather, physical entities are defined by the MCMPhysicalEntity class, which is a subclass of MCMUnManagedEntity (see section 7.5).

At this time, no attributes are currently defined for this class. A future version of this specification will add attributes to this class hierarchy after discussions about backwards compatibility with other models (e.g., SNMP, YANG, etc.) are completed.

At this time, no operations are currently defined for this class.

The MCMResource class participates in a single aggregation, called MCMResourceDefinedByMCMResourceOffer, as shown in 7.8.2.11.

7.8.6.2 MCMVirtualResource Class Hierarchy

This is an abstract class, and specializes MCMResource. It represents a set of objects that are configured by software to produce a new set of objects that behave like the resource(s) being virtualized. However, the behavior of the newly created set of MCMVirtualResources are *not* directly associated with the underlying physical hardware.

At this time, no attributes are currently defined for this class.

Table 43 defines following operations for this class:

Operation Name	Description
getMCMVirtualResourceParent() : MCMVirtualResource[1..1]	This operation returns the parent of this MCMVirtualResource object. This operation takes no input parameters. **[D128]** If this MCMVirtualResource object has no parent, then a NULL MCMVirtualResource object **SHOULD** be returned.
setMCMVirtualResourceParent(in newParent : MCMVirtualResourceComposite[1..1])	This operation defines the parent of this MCMVirtualResource object. The parent is defined in the input parameter, called newParent, and is of type MCMVirtualResourceComposite. **[D129]** If this MCMVirtualResource object already has a parent, then an exception **SHOULD** be raised.

Table 43. Operations of the MCMVirtualResource Class

The MCMVirtualResource class participates in one aggregation, called MCMHasVirtualResource; see section 7.8.6.4.

7.8.6.3 MCMVirtualResourceAtomic Class Definition

This is an abstract class, and specializes MCMVirtualResource.

It represents an MCMResource that is modeled as a single, stand-alone, manageable that is virtual, and not directly associated with the underlying physical hardware.

[R73] This object **MUST NOT** contain another MCMVirtualResource object.

At this time, no attributes are currently defined for this class.

At this time, no operations are currently defined for this class.

At this time, no relationships are defined for this class.

7.8.6.4 MCMVirtualResourceComposite Class Definition

This is an abstract class, and specializes MCMVirtualResource. It represents an MCMResource that is composite in nature (e.g., made up of multiple distinct MCMResource objects, at least one of which can be separately managed). An MCMVirtualResourceComposite represents a whole-part relationship; this produces a tree-structured class hierarchy. Note that a composite object defines three types of objects: the whole, the part, and the assembly of the whole with its parts.

[R74] An MCMVirtualResourceComposite object **MAY** contain zero or more MCMVirtualResourceAtomic and/or zero or more MCMVirtualResourceComposite objects.

[O61] Each MCMOrderComposite **MAY** contain one or more MCMOrderItems.

At this time, no attributes are defined for the MCMVirtualResourceComposite class. Most attributes will likely be realized using relationships and/or methods. For example, a query to an instance of the MCMVirtualResourceComposite class to provide its set of contained MCMVirtualResources (e.g., a set of virtual Ethernet ports associated with a virtual NIC) will be done by using class methods; the MCMVirtualResourceComposite instance will query each of its contained MCMVirtualResources (which will in turn call their methods to acquire their MCMVirtualResources), aggregate and organize the information, and provide that information in its method response.

Table 44 defines following operations for this class:

Operation Name	Description
getMCMVirtualResourceList() : MCMVirtualResource[1..*]	This operation returns the set of all MCMVirtualResource objects that are contained in this specific MCMVirtualResourceComposite object. There are no input parameters to this operation. This operation returns a list of zero or more MCMVirtualResource objects (i.e., the list is made up of MCMVirtualResourceAtomic and/or MCMVirtualResourceComposite objects). **[D130]** If this MCMVirtualResourceComposite object has no children, then it **SHOULD** return a NULL MCMVirtualResource object.
setMCMVirtualResourceList (in childObjectList : MCMVirtualResource [1..*])	This operation defines a set of MCMVirtualResource objects that will be contained by this particular MCMVirtualResourceComposite object. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMVirtualResource objects (i.e., one or more MCMVirtualResourceAtomic and/or MCMVirtualResourceComposite objects). This operation first deletes any existing

	contained MCMVirtualResource objects (and their aggregations and association classes), and then instantiates a new set of MCMVirtualResource objects; in doing so, each MCMVirtualResource object is contained within this particular MCMVirtualResourceComposite object by creating an instance of the MCMHasVirtualResource aggregation. **[D131]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasVirtualResourceDetail association class).
setMCMVirtualResourcePartialList (in childObjectList : MCMVirtualResource[1..*])	This operation defines a set of one or more MCMVirtualResource objects that should be contained within this particular MCMVirtualResourceComposite object WITHOUT affecting any other existing contained MCMVirtualResource objects or the objects that are contained in them. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMVirtualResource objects. This operation creates a set of aggregations between this particular MCMVirtualResourceComposite object and each of the MCMVirtualResource objects identified in the childObjectList. **[D132]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasVirtualResourceDetail association class).
delMCMVirtualResourceList()	This operation deletes ALL contained MCMVirtualResource objects of this particular MCMVirtualResourceComposite object. This has the effect of first, removing the association class, and second, removing the aggregation, between this MCMVirtualResource Composite object and each MCMVirtualResource object that is contained in this MCMVirtualResourceComposite object. This operation has no input parameters.
delMCMVirtualResourcePartialList (in childObjectList : MCMVirtualResource [1..*])	This operation deletes a set of MCMVirtualResource objects from this particular MCMVirtualResourceComposite object. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMVirtualResource objects. This has the effect of first, removing the association class and second, removing the aggregation, between each MCMVirtualResource object specified in the input parameter and this MCMVirtualResourceComposite object. Note that all other aggregations between this MCMVirtualResourceComposite and other MCMVirtualResource objects that are not identified in the input parameter are NOT affected.

Table 44. Operations of the MCMVirtualResourceComposite Class

The MCMVirtualResourceComposite class defines a single aggregation, called MCMHasVirtualResource. The multiplicity of this aggregation is 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMVirtualResource objects can be aggregated by this particular MCMVirtualResourceComposite object. Note that the cardinality on the part side (MCMVirtualResource) is 0..*; this enables an MCMVirtualResourceComposite object to be defined without having to define an associated MCMVirtualResource object for it to aggregate.

The semantics of this aggregation are defined by the MCMHasVirtualResourceDetail association class. This enables the management system to control which set of concrete subclasses of MCMVirtualResource are aggregated by this particular MCMVirtualResourceComposite object. The Policy Pattern may be used to control which specific MCMVirtualResource objects can be aggregated by which MCMVirtualResourceComposite objects for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.8.6.5 MCMLogicalResource Class Definition

The top of the MCMLogicalResource class hierarchy is shown in Figure 20.

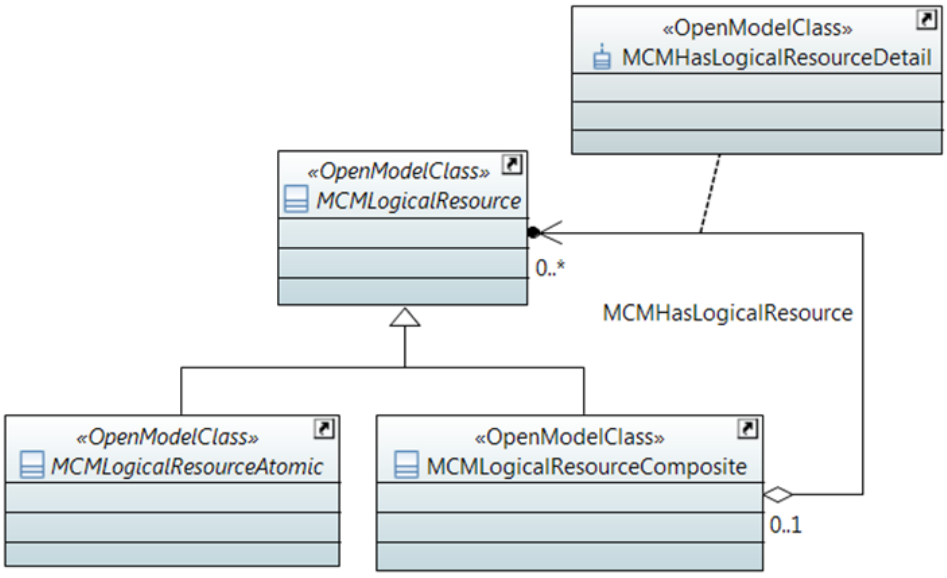

Figure 20. MCMResource Class Hierarchy, Part 2

This is an abstract class, and specializes MCMResource. It represents MCMResources that are neither physical nor virtual in nature, and which have inherent digital communication and management capabilities. Examples include operating systems, application and management software, protocols, and the logic required to perform forwarding, routing, and other functions.

At this time, no attributes are currently defined for this class.

Table 45 defines following operations for this class:

Operation Name	Description
getMCMLogicalResourceParent() : MCMLogicalResource[1..1]	This operation returns the parent of this MCMLogicalResource object. This operation takes no input parameters. **[D133]** If this MCMLogicalResource object has no parent, then a NULL MCMLogicalResource object **SHOULD** be returned.
getMCMLogicalResourceParent (in newParent) : MCMLogicalResource[1..1]	This operation defines the parent of this MCMLogicalResource object. The parent is defined in the input parameter, called newParent, and is of type MCMLogicalResourceComposite. **[D134]** If this MCMLogicalResource object already has a parent, then an exception **SHOULD** be raised.

Table 45. Operations of the MCMLogicalResource Class

The MCMLogicalResource class participates in one aggregation, called MCMHasLogicalResource; see section 7.8.6.7.

7.8.6.6 MCMLogicalResourceAtomic Class Definition

This is an abstract class, and specializes MCMLogicalResource. It represents an MCMResource that is modeled as a single, stand-alone, manageable object.

[R75] This object **MUST NOT** contain another MCMLogicalResource object.

At this time, no attributes are currently defined for this class.

At this time, no operations are currently defined for this class.

At this time, no relationships are defined for this class.

7.8.6.7 MCMLogicalResourceComposite Class Definition

This is an abstract class, and specializes MCMLogicalResource. It represents MCMResources that are composite in nature (e.g., made up of multiple distinct MCMResource objects, at least one of which can be separately managed). An MCMLogicalResourceComposite represents a whole-part relationship; this produces a tree-structured class hierarchy. Note that a composite object defines three types of objects: the whole, the part, and the assembly of the whole with its parts.

[O62] An MCMLogicalResourceComposite object **MAY** contain zero or more MCMLogicalResourceAtomic and/or zero or more MCMLogicalResourceComposite objects.

At this time, no attributes are defined for the MCMLogicalResourceComposite class. Most attributes will likely be realized using relationships and/or methods. For example, the *usage* of an

MCMLogicalResourceComposite can be considered from two viewpoints: (1) how much *content* is left (e.g., a subscription limits downloads to 1Gb/months, and the current usage is 750Mb), and (2) how much *time* is left (e.g., it is being used on a time-limited subscription). In either of these cases, an attribute is inappropriate, since one or more computations and information from one or more relationships are required to provide a value. In addition, the MCMLogicalResourceComposite itself doesn't "know" how much usage is incurred, but can find out (e.g., by using a method). Hence, class methods will likely be added to provide more detailed information for instances of this class in the next CfC.

Table 46 defines following operations for this class:

Operation Name	Description
getMCMLogicalResourceList() : MCMLogicalResource[1..*]	This operation returns the set of all MCMLogicalResource objects that are contained in this specific MCMLogicalResourceComposite object. There are no input parameters to this operation. This operation returns a list of zero or more MCMLogicalResource objects (i.e., the list is made up of MCMLogicalResourceAtomic and/or MCMLogicalResourceComposite objects). **[D135]** If this MCMLogicalResourceComposite object has no children, then it **SHOULD** return a NULL MCMLogicalResource object.
setMCMLogicalResourceList (in childObjectList : MCMLogicalResource [1..*])	This operation defines a set of MCMLogicalResource objects that will be contained by this particular MCMLogicalResourceComposite object. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMLogicalResource objects (i.e., one or more MCMLogicalResourceAtomic and/or MCMLogicalResourceComposite objects). This operation first deletes any existing contained MCMLogicalResource objects (and their aggregations and association classes), and then instantiates a new set of MCMLogicalResource objects; in doing so, each MCMLogicalResource object is contained within this particular MCMLogicalResourceComposite object by creating an instance of the MCMHasLogicalResource aggregation. **[D136]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasLogicalResourceDetail association class).
setMCMLogicalResourcePartialList (in childObjectList : MCMLogicalResource[1..*])	This operation defines a set of one or more MCMLogicalResource objects that should be contained within this particular MCMLogicalResourceComposite object WITHOUT affecting any other existing contained MCMLogicalResource ob-

	jects or the objects that are contained in them. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMLogicalResource objects. This operation creates a set of aggregations between this particular MCMLogicalResourceComposite object and each of the MCMLogicalResource objects identified in the childObjectList. **[D137]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasLogicalResourceDetail association class).
delMCMLogicalResourceList()	This operation deletes ALL contained MCMLogicalResource objects of this particular MCMLogicalResourceComposite object. This has the effect of first, removing the association class, and second, removing the aggregation, between this MCMLogicalResourceComposite object and each MCMLogicalResource object that is contained in this MCMLogicalResourceComposite object. This operation has no input parameters.
delMCMLogicalResourcePartialList (in childObjectList : MCMLogicalResource [1..*])	This operation deletes a set of MCMLogicalResource objects from this particular MCMLogicalResourceComposite object. This operation takes a single input parameter, called childObjectList, which is an array of one or more MCMLogicalResource objects. This has the effect of first, removing the associationnc class and second, removing the aggregation, between each MCMLogicalResource object specified in the input parameter and this MCMLogicalResourceComposite object. Note that all other aggregations between this MCMLogicalResourceComposite and other MCMLogicalResource objects that are not identified in the input parameter are NOT affected.

Table 46. Operations of the MCMLogicalResource Class

At this time, a single aggregation is defined for the MCMLogicalResourceComposite class. This aggregation is named MCMHasLogicalResource, and defines the set of MCMLogicalResource objects that are contained in this particular MCMLogicalResourceComposite object. The multiplicity of this aggregation is 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMLogicalResource objects can be aggregated by this particular MCMLogicalResourceComposite object. Note that the cardinality on the part side (MCMLogicalResource) is 0..*; this enables an MCMLogicalResourceComposite object to be defined without having to define an associated MCMLogicalResource object for it to aggregate.

The semantics of this aggregation are defined by the MCMHasLogicalResourceDetail association class. This enables the management system to control which set of concrete subclasses of MCMLogicalResource are contained by this particular MCMLogicalResourceComposite. This enables a particular set of MCMLogicalResource (i.e., zero or more MCMLogicalResourceAtomic and/or zero or more MCMLogicalResourceComposite) objects to be contained within a particular MCMLogicalResourceComposite object. The Policy Pattern may be used to control which specific MCMLogicalResource objects are contained within a given MCMLogicalResourceComposite for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.8.6.8 MCMCatalog Class Definition

Figure 21 shows the remaining subclasses of MCMLogicalResourceAtomic and MCMLogicalResourceComposite.

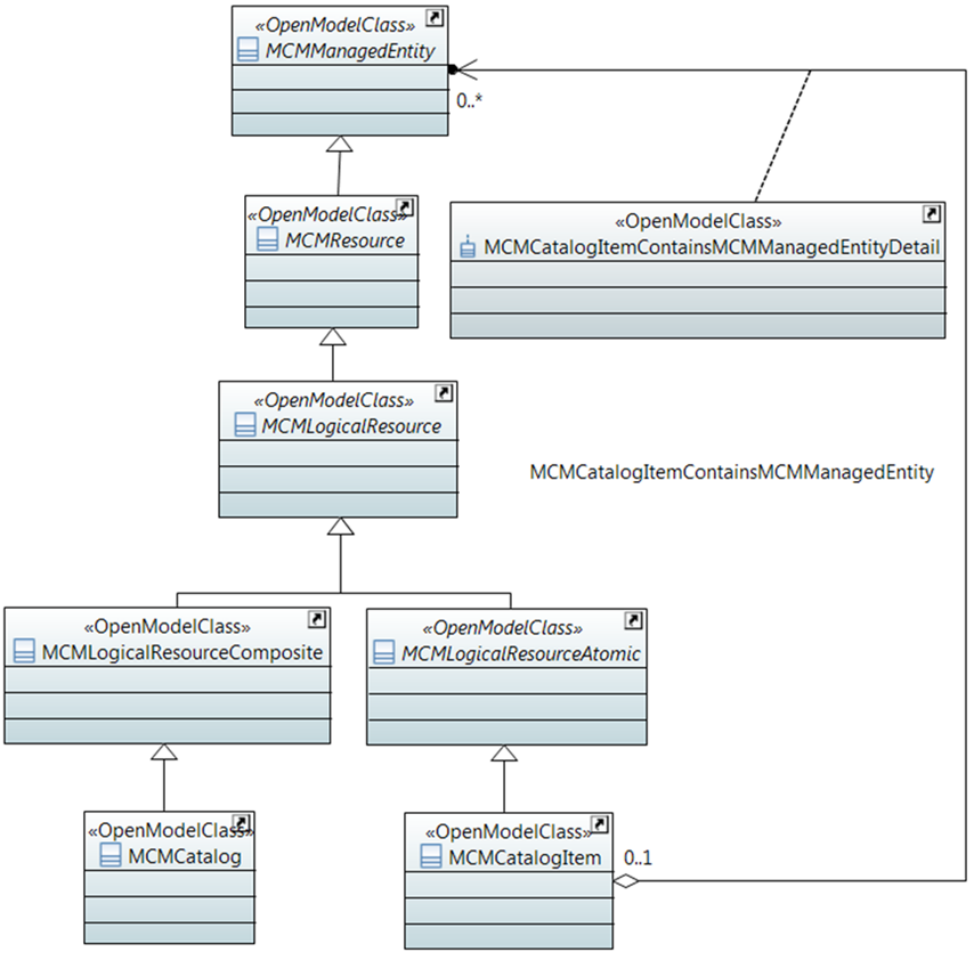

Figure 21. MCMResource Class Hierarchy, Part 3

MCMCatalog is a concrete class, and specializes MCMLogicalResourceComposite. It defines a container that aggregates a set of MCMCatalogItem objects. Each MCMCatalogItem object can either represent an item of interest to the MCMCatalog directly, or can represent a set of MCMManagedEntities (using the MCMCatalogItemContainsMCMManagedEntity aggregation). The semantics of this aggregation enable a set of MCMRoles or other MCMManagedEntities to control which MCMCatalogItems are viewable in a given MCMCatalog. The set of MCMCatalogItems are organized according to one or more identifying objectives (e.g., subject attributes added in a subclass of MCMCatalog, or metadata attached to the MCMCatalog).

At this time, no attributes are defined for the MCMCatalog class. Most attributes will likely be realized using relationships and/or methods. For example, a query to an instance of the MCMCatalog class to provide its set of contained MCMCatalog and MCMCatalogItem objects will be done by using class methods. The MCMCatalog instance will query each of its contained MCMCatalog objects, as well as any MCMCatalogItem objects that it contains, aggregate and organize the information, and provide that information in its method response.

Table 47 defines following operations for this class:

Operation Name	Description
getMCMCatalogItemList() : MCMCatalogItem[1..*]	This operation returns the set of all MCMCatalogItem objects that are contained in this specific MCMCatalog object, and not for any MCMCatalogs that are contained within this MCMCatalog object. There are no input parameters to this operation. This operation returns a list of zero or more MCMCatalogItem objects. **[D138]** If this object does not contain any MCMCatalogItem objects, then a NULL MCMCatalogItem object **SHOULD** be returned.
setMCMCatalogItemList(in newCatalogItemList : MCMCatalogItem[1..*])	This operation defines a set of MCMCatalogItem objects that will be contained by this particular MCMCatalog object, and not for any MCMCatalogs that are contained within this MCMCatalog object. This operation takes a single input parameter, called newCatalogItemList, which is an array of one or more MCMCatalogItem objects. This operation first deletes any existing contained MCMCatalogItem objects (and their aggregations and association classes), and then instantiates a new set of MCMCatalogItem objects; in doing so, each MCMCatalogItem object is contained within this particular MCMCatalog object by creating an instance of the MCMHasLogicalResource aggregation. **[D139]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasLogicalResourceDetail association class).

setMCMServicePartialList (in newCatalogItemList: MCMCatalogItem[1..*])	This operation defines a set of one or more MCMCatalogItem objects that should be contained within this particular MCMCatalog object WITHOUT affecting any other existing contained MCMCatalogItem or MCMCatalog objects. This operation takes a single input parameter, called newCatalogItemList, which is an array of one or more MCMCatalogItem objects. This operation creates a set of aggregations between this particular MCMCatalog object and each of the MCMCatalogItem objects identified in the newCatalogItem. **[D140]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasLogicalResourceDetail association class).
delMCMCatalogItemList()	This operation deletes ALL contained MCMCatalogItem objects contained within this particular MCMCatalog object. This has the effect of first, removing the association class, and second, removing the aggregation, between this MCMCatalog object and each MCMCatalogItem object that is contained in this MCMCatalog object. This operation has no input parameters.
delMCMCatalogItemPartialList (in newCatalogItemList : MCMCatalogItem[1..*])	This operation deletes a set of MCMCatalogItem objects from this particular MCMCatalog object. This operation takes a single input parameter, called newCatalogItemList, which is an array of one or more MCMCatalogItem objects. This has the effect of first, removing the association class and second, removing the aggregation, between each MCMCatalogItem object specified in the input parameter and this MCMCatalog object. Note that all other aggregations between this MCMCatalog and other MCMCatalogItem objects that are not identified in the input parameter are NOT affected.

Table 47. Operations of the Catalog Class

At this time, no relationships are defined for this class. Note that MCMCatalog objects may contain any number of MCMCatalogItems, because an MCMCatalog inherits the ability to aggregate MCMLogicalResourceAtomic and/or MCMLogicalResourceComposite objects from MCMLogicalResourceComposite. This also means that an MCMCatalog may itself contain other MCMCatalogs (e.g., to define a tree structure of catalogs).

7.8.6.9 MCMCatalogItem Class Definition

This is an abstract class, and specializes MCMLogicalResourceAtomic. It represents a set of MCMManagedEntities that are contained in a particular MCMCatalog and organized by a particular identifying objective. The MCMManagedEntities to be contained in an MCMCatalog can either be defined indirectly using the MCMCatalogItemContainsMCMManagedEntity aggregation, or using another means (e.g., creating a subclass with dedicated attributes and operations that describe the MCMManagedEntity directly).

No attributes are currently defined for this class.

No operations are currently defined for this class.

At this time, a single relationship, called MCMCatalogItemContainsMCMManagedEntity, is defined for the MCMCatalogItem class. Its multiplicity is defined as 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMManagedEntity objects can be aggregated by this particular MCMCatalogItem object. Note that the cardinality on the part side (MCMManagedEntity) is 0..*; this enables an MCMCatalogItem object to be defined without having to define an associated MCMManagedEntity object for it to aggregate. Significantly, this means that an MCMCatalogItem may be any type of MCMManagedEntity; this addresses the use case of managed objects (e.g., a VNF) not being able to be categorized into a single subclass (i.e., is it a product, resource, or service).

The semantics of this aggregation are defined by the MCMCatalogItemContainsMCMManagedEntityDetail association class. This enables the semantics of the aggregation to be defined using the attributes and behavior of this association class. For example, it can be used to define which MCMManagedEntity objects are allowed to be associated with which MCMCatalogItem objects.

Both of the above association classes can be further enhanced by using the Policy Pattern (see Figure 3) to define policy rules that constrain which MCMManagedEntity objects are attached to which MCMCatalogItem object. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.8.6.10 MCMServiceInterface Class Definition

This is a concrete class, and specializes MCMLogicalResourceAtomic. It represents a logical point in a topology where other MCMResources may be used to enable an MCMServiceEndpoint to function (i.e., be instantiated). An MCMServiceInterface may support multiple MCMServiceEndpoints.

No attributes are currently defined for this class.

No operations are currently defined for this class.

At this time, a single aggregation is defined for the MCMServiceInterface class. This aggregation is named MCMServiceEndpointHasMCMServiceInterface, and is defined in section 7.8.6.10.

7.9 MCMParty Class Hierarchy

The MCMParty class has two subclasses, as shown in Figure 22.

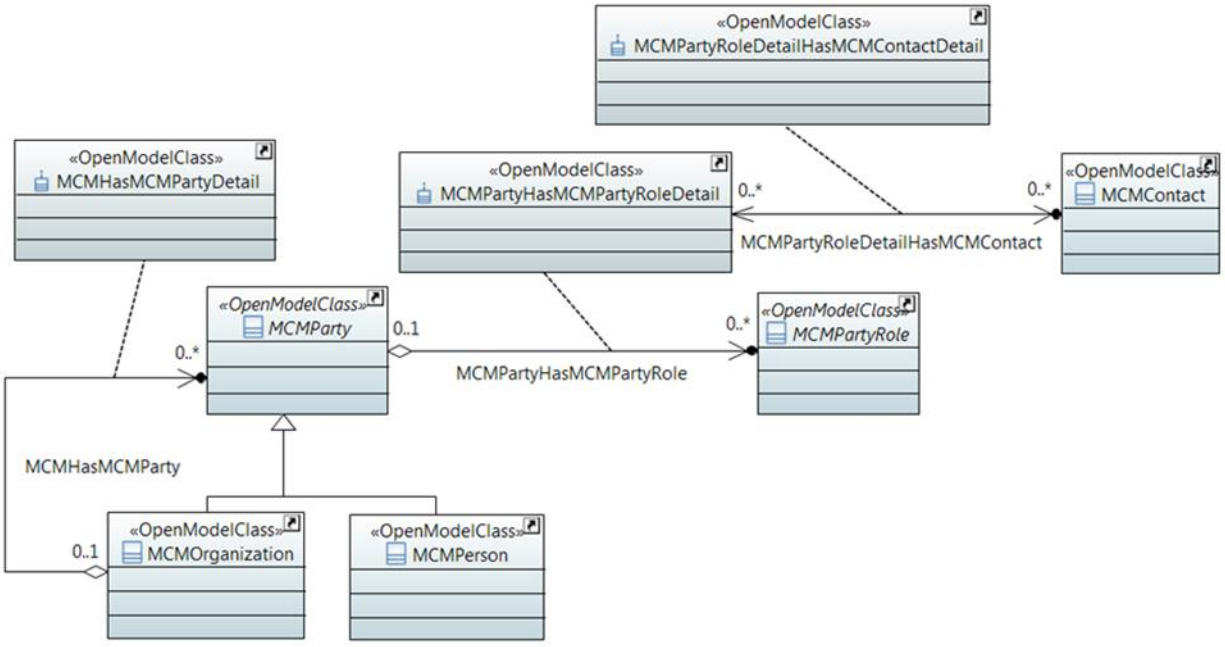

Figure 22. MCMParty Class Hierarchy

Table 48 defines the purpose of this hierarchy, and aligns them to MEF 55 [1]. The purpose of the MCMParty class hierarchy is to represent different individuals, groups of people, and organizations that perform business functions in the managed environment. Such people could be internal or external to the organization. Note that MCMParty aggregates one or more MCMPartyRole objects (see section 7.11.2.2), which both provide a context for the business function as well as define a set of responsibilities that a particular MCMParty has.

Name of Class	Function	Relation to MEF 55
MCMParty	Represents either an individual person or a group of people. An MCMParty may take on zero or more MCMPartyRoles.	Represents human actors in the MEF LSO RA.
MCMPerson	Represents an individual Person	Represents individual human actors in the MEF LSO RA.
MCMOrganization	Represents a group of People and/or Organizations	Represents human actors in a group in the MEF LSO RA.

Table 48. Functions of the MCMParty Class and its Subclasses

The following subsections describe these classes in more detail.

7.9.1 MCMParty Class Definition

This is an abstract class, and specializes MCMEntity. It represents either an individual person or a group of people functioning as an organization. A group of people can also be structured as an organization made up of organizational units. An MCMParty may take on zero or more MCMPartyRoles; this acts as a filter. For example, an MCMParty that takes on the role HelpDesk can be used to represent any group or individual that performs a HelpDesk function. Behavior and characteristics that are common to both organization and person objects are modeled in this class.

At this time, no attributes are defined for this class.

Class operations and relationships are used to provide flexibility and power in using this class (and its subclasses). For example, an MCMManagedEntity may need to know which set of MCMPartyRoles are currently associated with this particular MCMParty. Since MCMPartyRoles can change dynamically at runtime, an attribute cannot accurately reflect this. In contrast, a method can simply look for instantiated aggregations of type MCMPartyHasMCMPartyRole (see next paragraph); it can even look at the MCMPartyHasMCMPartyRoleDetail association class, and/or associated MCMMetaData objects, if it needs further detail.

Table 49 defines following operations for this class:

Operation Name	Description
getMCMPartyParent() : MCMOrganization[1..1]	This operation returns the parent of this MCMParty object. This operation takes no input parameters. **[D141]** If this MCMParty object has no parent, then a NULL MCMParty object **SHOULD** be returned.
setMCMPartyParent(in newParent : MCMOrganization [1..1])	This operation defines the parent of this MCMParty object. The parent is defined in the input parameter, called newParent, and is of type MCMOrganization. **[D142]** If this MCMService object already has a parent, then an exception **SHOULD** be raised.
getMCMPartyRoleList() : MCMPartyRole[1..*]	This operation returns the set of MCMPartyRole objects that are decorating this MCMParty object. There are no input parameters. This operation identifies any instances of the MCMPartyHasMCMPartyRole aggregation. For each instance of this aggregation, this operation then adds each MCMPartyRole defined in this aggregation, and adds each MCMPartyRole to an array that is returned by this operation.

	[D143] If this MCMParty object does not aggregate any MCMPartyRole objects, then a NULL MCMPartyRole object **SHOULD** be returned.
setMCMPartyRoleList (in newPartyRoleList : MCMPartyRole[1..*])	This operation defines the set of MCMPartyRole objects that will be aggregated by this MCMParty object. This operation takes a single input parameter, called newPartyRoleList, which is of type MCMPartyRole. This operation creates a set of aggregations between this particular MCMParty object and the set of MCMPartyRole objects identified in the input parameter. Note that this operation first deletes any existing MCMPartyRole objects (and their aggregations and association classes) that were aggregated by this MCMParty object, and then instantiates a new set of MCMPartyRole objects; in doing so, each MCMPartyRole object is attached to this particular MCMParty object by first, creating an instance of the MCMPartyHasMCMPartyRole aggregation, and second, realizing that aggregation instance as an association class. **[D144]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMPartyHasMCMPartyRoleDetail association class).
setMCMPartyRolePartial-List (in newPartyRoleList: MCMPartyRole[1..*])	This operation defines a set of one or more MCMPartyRole objects that will decorate this MCMParty object WITHOUT affecting any other existing MCMPartyRole objects that are decorating this MCMParty object. This operation takes a single input parameter, called newPartyRoleList, which is an array of one or more MCMPartyRole objects. This operation creates a set of aggregations between this particular MCMParty object and the set of MCMPartyRole objects identified in the input parameter. **[D145]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMPartyHasMCMPartyRoleDetail association class).
delMCMPartyRoleList()	This operation deletes ALL MCMPartyRole object instances that are decorating this MCMParty object. This operation first removes the association class, and second, removes the aggregation, between this MCMParty object and each MCMPartyRole object that is aggregated by this MCMParty object. This operation has no input parameters. This operation does not delete any of the MCMPartyRole objects; it simply disconnects them from the MCMParty that they were aggregating.

delMCMPartyRolePartial-List (in newPartyRoleList : MCMPartyRole[1..*])	This operation deletes a set of MCMPartyRole objects that are aggregated by this particular MCMParty object. This operation takes a single input parameter, called newPartyRoleList, which is an array of one or more MCMPartyRole objects. This operation first removes the association class and second, removes the aggregation, between each MCMPartyRole object specified in the input parameter and this MCMParty object. Note that all other aggregations between this MCMParty object and other MCMPartyRole objects that are not specified in the input parameter are NOT affected.

Table 49. Operations of the MCMParty Class

At this time, a single aggregation is defined for the MCMParty class. This aggregation is named MCMPartyHasMCMPartyRole, and defines the set of MCMPartyRoles that this particular MCMParty can take on. The multiplicity of this aggregation is 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMPartyRole objects can be aggregated by this particular MCMParty object. Note that the cardinality on the part side (MCMPartyRole) is 0..*; this enables an MCMParty object to be defined without having to define an associated MCMPartyRole object for it to aggregate.

The semantics of this aggregation are defined by the MCMPartyHasMCMPartyRoleDetail association class. This enables the management system to control which set of concrete subclasses of MCMPartyHasMCMPartyRole are taken on by this particular MCMParty. The Policy Pattern may be used to control which specific responsibilities, which are defined by a set of MCMPartyHasMCMPartyRole objects, are taken on by a given MCMParty for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

The MCMParty class also participates in a single aggregation, called MCMHasMCMParty; please see section 7.9.

One additional important aggregation is MCMPartyRoleDetailHasMCMContact; please see section 7.9.1.

7.9.2 MCMOrganization Class Definition

This is a concrete class, and specializes MCMParty. An MCMOrganization is defined as a group of people (e.g., defined as instances of either MCMPerson, MCMOrganization, or an appropriate subclass of each) identified by shared interests or purpose. This includes attributes such as the legal name of the organization; attributes such as the head of the organization, or which types of employees belong to which organization, are instead modeled as subclasses of MCMMetaData and associated with that MCMOrganization using the Role-Object pattern, since (1) they are not necessary to define the concept of an MCMOrganization, and (2) they can change dynamically.

An MCMOrganization object can interact with other MCMOrganization and MCMPerson objects directly or through its MCMPartyRole(s). Behavior and characteristics that are specific to an MCMOrganization are modeled using a combination of classes for specific concepts augmented by the Role-Object pattern for each; this ensures that (1) the model is not dependent on one particular person, group, or organization, and (2) it separates the characteristics and behavior of the Entity being modeled from its responsibilities and functions. This provides a more accurate and extensible model.

Table 50 defines following attributes for this class:

Attribute Name	Mandatory?	Description
mcmIsLegalEntity : Boolean[0..1]	NO	This is a Boolean attribute. If its value is TRUE, then this organization is a legal entity.
mcmIsTempOrg : Boolean[0..1]	NO	This is a Boolean attribute. If its value is TRUE, then this organization represents a temporary organization that has a defined lifetime (defined in associated MCMMetaData). Its budget, space, resources, and other factors are allocated only for a defined period.
mcmIsVirtualOrg : Boolean[0..1]	NO	This is a Boolean attribute. If its value is TRUE, then this organization represents a virtual organization that convenes using an electronic mechanism (e.g, via phone or Internet).

Table 50. Attributes of the MCMOrganization Class

Table 51 defines following operations for this class:

Operation Name	Description
getMCMIsLegalEntity() : Boolean[1..1]	This operation returns the value of the mcmIsLegalEntity attribute. This operation takes no input parameters.
setMCMIsLegalEntity(in isLegal : Boolean[1..1])	This operation defines the value of the mcmIsLegalEntity attribute. It contains a single input parameter, of type Boolean. If the value of this attribute is TRUE, then this MCMOrganization is a legal entity.
getMCMIsTempOrg(): Boolean[1..1]	This operation returns the value of the mcmIsTempOrg attribute. This operation takes no input parameters.
setMCMIsTempOrg (in isTemp : Boolean[1..1])	This operation defines the value of the mcmIsTempOrg attribute. It contains a single input parameter, of type Boolean. If the value of this attribute is TRUE, then this MCMOrganization is a temporary organization.
getMCMIsVirtualOrg(): Boolean[1..1]	This operation returns the value of the mcmIsLegalEntity attribute. This operation takes no input parameters.

setMCMIsVirtualOrg(in isVirtualOrg : Boolean[1..1])	This operation defines the value of the mcmIsLegalEntity attribute. It contains a single input parameter, of type Boolean. If the value of this attribute is TRUE, then this MCMOrganization is a virtual organization.

Table 51. Operations of the MCMOrganization Class

At this time, a single aggregation is defined for the MCMOrganization class. It is named MCMHasMCMParty. The multiplicity of this aggregation is 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMParty objects can be aggregated by this particular MCMOganization object. Note that the cardinality on the part side (MCMParty) is 0..*; this enables an MCMOganization object to be defined without having to define an associated MCMParty object for it to aggregate.

The semantics of this aggregation are defined by the MCMHasMCMPartyDetail association class. This enables the management system to control which set of concrete subclasses of MCMParty objects are aggregated by this particular MCMOrganization.

The Policy Pattern may be used to control which specific part objects (i.e., MCMParty) are associated with which specific aggregate (i.e., MCMOrganization) object, respectively, for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.9.3 MCMPerson Class Definition

This is a concrete class, and specializes MCMParty. An MCMPerson defines the concept of an individual that may have a set of MCMPartyRoles that formalize the responsibilities of that individual. Attributes such as username, password, phone number(s), the format of the name of the MCMPerson, and skills that the MCMPerson has are modeled as subclasses of MetaData and associated with that Person using the Role-Object pattern, since (1) they are not necessary to define the concept of an MCM attributes such as gender and birthDate Person, and (2) they can change dynamically.

An MCMPerson can interact with an MCMOrganization directly or through his or her MCMPartyRole(s). Behavior and characteristics that are specific to an individual are modeled using a combination of classes for specific concepts augmented by the Role-Object pattern for each; this ensures that (1) the model is not dependent on one particular person, group, or organization, and (2) it separates the characteristics and behavior of the individual and his or her responsibilities being modeled from its responsibilities and functions. This provides a more accurate and extensible model.

No attributes are currently defined for this class.

No operations are currently defined for this class.

No relationships are currently defined for this class.

7.10 The InformationResource Class Hierarchy

Figure 23 shows the MCMInformationResource class hierarchy on the left, and some important aggregations that the MCMInformationResource class participates in on the right. The following subsections describe the classes in the MCMInformationResource class hierarchy in more detail.

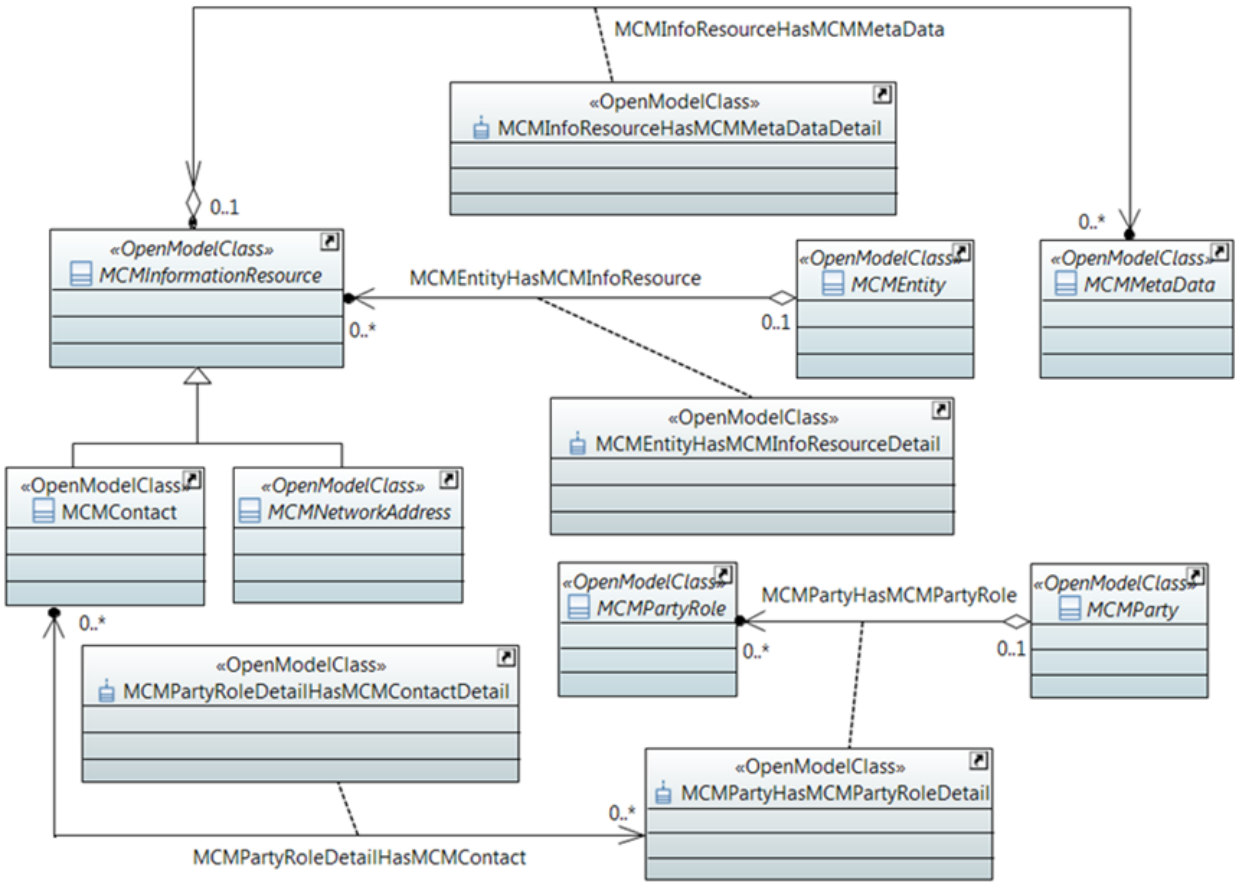

Figure 23. The MCMInformationResource Class Hierarchy

7.10.1 MCMInformationResource Class Definition

This is an abstract class, and specializes MCMRootEntity. It defines information that is needed by a management system to describe other information. However, that information is not an inherent part of an MCMEntity; rather, that information is managed and controlled using another MCMManagedEntity. For example, an IPAddress is an important concept in networking. However, an IPAddress is not directly managed; rather, an MCMManagedEntity that is responsible for the lifecycle of the IPAddress (e.g., a DHCPServer) is responsible for its management. Therefore, the concept of an IPAddress is represented as a type of MCMInformationResource, and is associated to an MCMManagedEntity that performs its management.

No attributes are currently defined for this class.

Table 52 defines following operations for this class:

Operation Name	Description
getMCMNetworkAddress-FreeList() : MCMNetwork-Address[1..1]	This operation returns the set of all MCMNetworkAddress objects that are free-standing (i.e., they are not aggregated by any subclass of an MCMEntity class). The getMCMInfoResourceList operation is used to retrieve the set of MCMNetworkAddress objects that are aggregated by a given MCMEntity object. **[D146]** If no MCMNetworkAddress objects are found, then a NULL MCMNetworkAddress object **SHOULD** be returned.
setMCMNetworkAddress-FreeList(in newNetAddrFreeList : MCMNetwork-Address[1..1])	This operation defines a new set of MCMNetworkAddresses to be created that are free-standing (i.e., they are not aggregated by any subclass of MCMEntity). A single input parameter, of type MCMNetworkAddress, defines an array of one or more MCMNetworkAddress objects. The operation only defines the MCMNetworkAddress objects; it does not associate them with an MCMEntity. The setMCMInfoResourceList and setMCMInfoResourcePartialList operations are used to associate an MCMNetworkAddress to a particular MCMEntity object.
setMCMNetworkAddress-FreePartialList(in newNetAddrFreeList : MCMNetworkAddress[1..*])	This operation defines a new set of one or more free-standing MCMNetworkAddress objects (i.e., they are not aggregated by any subclass of MCMEntity) WITHOUT affecting any other existing MCMNetworkAddress objects that are associated with this MCMEntity object. This operation takes a single input parameter, called newNetAddrFreeList, which is an array of one or more MCMNetworkAddress objects. This operation only defines the MCMNetworkAddress objects; it does not associate them with an MCMEntity. The setMCMInfoResourceList and setMCMInfoResourcePartialList operations are used to associate an MCMNetworkAddress to a particular MCMEntity object.
delMCMNetworkAddress-FreeList()	This operation is used to delete all free-standing MCMNetworkAddress objects; use the delMCMInfoResourceList or delMCMInfoResourcePartialList operations to delete the set of MCMNetworkAddress objects that are aggregated by a given MCMEntity object.
delMCMNetworkAddress-FreePartialList (in newNetworkAddressList :	This operation deletes ALL MCMNetworkAddress object instances that are specified in its input parameter that are free-

MCMNetwork-Address[1..*])	standing (i.e., not aggregated by any subclass of MCMEntity). This operation takes a single input parameter, called newNetworkAddressList, which is of type MCMNetworkAddress. This operation is used to delete free-standing MCMNetwork-Address objects; use the delMCMInfoResourceList or delMCMInfoResourcePartialList operations to delete the set of MCMNetworkAddress objects that are aggregated by a given MCMEntity object.
getMCMContactFreeList() : MCMContact[1..1]	This operation returns the set of all MCMContact objects that are free-standing (i.e., they are not aggregated by any subclass of an MCMEntity class). The getMCMInfoResourceList operation is used to retrieve the set of MCMContact objects that are aggregated by a given MCMEntity object. **[D147]** If no MCMContact objects are found, then a NULL MCMContact object **SHOULD** be returned.
setMCMContactFreeList(in newContactList : MCMContact[1..*])	This operation defines a new set of MCMContact to be created that are free-standing (i.e., they are not aggregated by any subclass of MCMEntity). A single input parameter, of type MCMContact, defines an array of one or more MCMContact objects. The operation only defines the MCMContact objects; it does not associate them with an MCMEntity. The setMCMInfoResourceList and setMCMInfoResourcePartialList operations are used to associate an MCMContact to a particular MCMEntity object.
setMCMContactFreePartialList(in newNetAddrFreeList : MCMNetwork-Address[1..*])	This operation defines a set of one or more free-standing MCMContact objects WITHOUT affecting any other existing MCMContact objects. This operation takes a single input parameter, called newNetAddrFreeList, which is an array of one or more MCMContact objects. This operation only defines the MCMContact objects; it does not associate them with an MCMEntity. The setMCMInfoResourceList and setMCMInfoResourcePartialList operations are used to associate an MCMContact to a particular MCMEntity object.
delMCMContactFreeList()	This operation is used to delete all free-standing MCMContact objects; use the delMCMInfoResourceList or delMCMInfoResourcePartialList operations to delete the set of MCMContact objects that are aggregated by a given MCMEntity object.
delMCMContactFreePartialList (in newContactList : MCMContact[1..*])	This operation deletes ALL MCMContact object instances that are specified in its input parameter that are free-standing (i.e., not aggregated by any subclass of MCMEntity). This operation takes a single input parameter, called newContactList, which is of type MCMContact.

	This operation is used to delete free-standing MCMContact objects; use the delMCMInfoResourceList or delMCMInfoResourcePartialList operations to delete the set of MCMContact objects that are aggregated by a given MCMEntity object.

Table 52. Operations of the MCMInformationResource Class

At this time, the MCMInformationResource class defines a single aggregation, called MCMInfoResourceHasMCMMetaData. The multiplicity of this aggregation is 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more MCMMetaData objects can be aggregated by this particular MCMInformationResource object. Note that the cardinality on the part side (MCMMetaData) is 0..*; this enables an MCMInformationResource object to be defined without having to define an associated MCMMetaData object for it to aggregate.

The semantics of this aggregation are defined by the MCMInfoResourceHasMCMMetaDataDetail association class. This enables the management system to control which set of MCMMetaData objects are aggregated by which set of MCMInformationResource objects.

Note that the Policy Pattern may be used to control which specific part objects (i.e., MCMMetaData) are associated with which specific aggregate (i.e., MCMInformationResource) objects, respectively, for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

The MCMInformationResource participates in a second aggregation, called MCMEntityHasMCMInfoResource; please see section 7.10.

7.10.2 MCMNetworkAddress Class Definition

This is an abstract class, and specializes MCMInformationResource. It defines a network address, which is a unique identifier for a node on a network. Such identifiers can be local, private, or public (e.g., globally unique). A network node may have zero or more MCMNetworkAddresses (e.g., a router may have multiple interfaces, and each interface may have a set of MCMNetwork-Addresses). Examples of an MCMNetworkAddress include telephone numbers, IPv4 and IPv6 addresses, MAC addresses, and X.21 or X.25 addresses (in a circuit-switched data network).

No attributes are currently defined for this class.

No operations are currently defined for this class.

No relationships are currently defined for this class.

7.10.3 MCMContact Class Definition

This is a concrete class, and specializes MCMInformationResource. It represents the information needed to communicate with a particular MCMParty or MCMPartyRole. Examples include technical and administrative contacts for Order information and technical implementation work (e.g., the network administrator of an MCMManagementDomain).

No attributes are currently defined for this class.

No operations are currently defined for this class.

At this time, this class participates in a single association, as shown in Figure 23. An MCMPartyRoleDetail is an association class (see section 7.11.2.2) that defines a set of MCMPartyRoles that are used by a given MCMParty. The MCMPartyRoleDetailHasMCMContact is an association between the MCMPartyRoleDetail association class and the MCMContact class. This association defines the set of MCMContacts that are related to this particular MCMPartyRoleDetail object (i.e., the set of MCMParty objects that are playing a specific MCMPartyRole). For example, this association can be used to define the contact information for a set of MCMParty objects that are each playing a set of MCMPartyRoles; a common use is to define the contact information for different types of MCMBuyer and MCMSeller objects.

The multiplicity of this association is 0..* – 0..*. This means that this association is optional (i.e., the “0” part of the 0..* cardinality). If this association is instantiated (e.g., the “0..* cardinality on the MCMPartyHasMCMPartyRoleDetail is greater than 0), then zero or more MCMContact objects can be associated with this particular MCMPartyHasMCMPartyRoleDetail object. Note that the cardinality on the part side (MCMContact) is 0..*; this enables an MCMPartyHasMCMPartyRoleDetail object to be defined without having to define an associated MCMContact object for it to aggregate.

The semantics of this aggregation are defined by the MCMPartyRoleDetailHasMCMContactDetail association class. This enables the management system to control which set of MCMPartyRoleDetail objects are associated to which set of MCMContact objects.

Note that the Policy Pattern may be used to control which specific part objects (i.e., MCMMetaData) are associated with which specific aggregate (i.e., MCMInformationResource) objects, respectively, for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.11 The MCMMetaData Class Hierarchy

Figure 24 shows a portion of the MCMMetaData class hierarchy. This figure will be used to describe the MCMMetaData class and the three aggregations that it participates in.

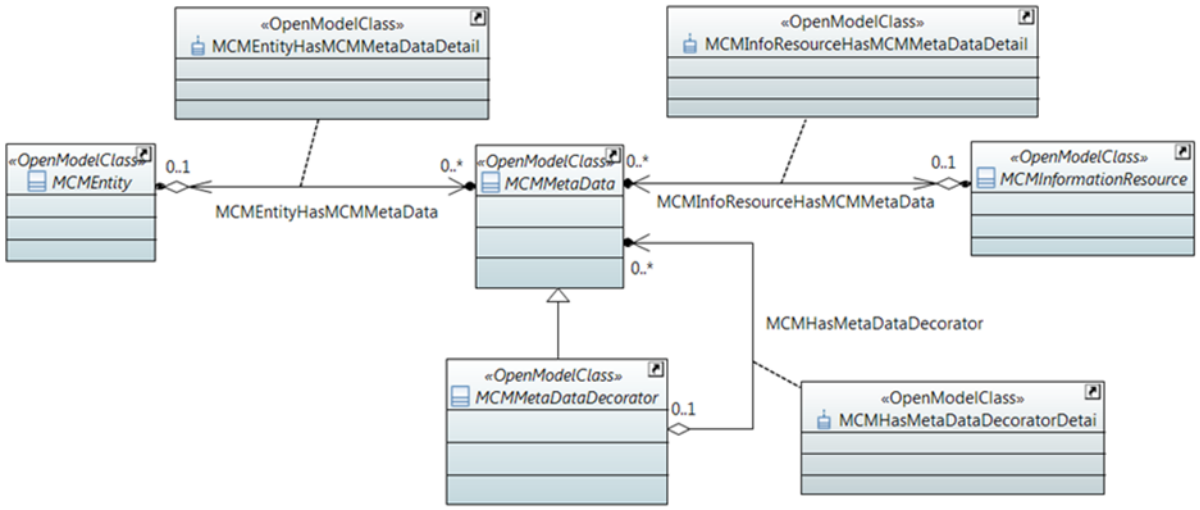

Figure 24. The MCMMetaData Class Hierarchy, Part 1

7.11.1 MCMMetaData Class Definition

This is an abstract class, and specializes MCMRootEntity. It defines prescriptive and/or descriptive information about the object(s) to which it is attached. These descriptive and/or prescriptive characteristics and behavior are not an inherent, distinguishing characteristic or behavior of that object (otherwise, it would be an integral part of that object). Examples of prescriptive and descriptive metadata are the definition of a time period during which specific types of operations are allowed, and documentation about best current practices, respectively.

Table 53 defines following attributes for this class:

Attribute Name	Mandatory?	Description
mcmMetaDataEnableStatus : MCMMetaDataEnableStatus[0..1]	NO	This is an optional enumeration that defines whether the MCMEntity that this MCMMetaData object refers to is enabled for normal operation or not. The values that this attribute can have are defined by the MCMMetaDataEnableStatus enumeration, and include:

		0: ERROR 1: INIT 2: Enabled (ok to use for all operations) 3: Enabled for testing only 4: Disabled (cannot be used) 5: Unknown (e.g., cannot be contacted to ascertain state)
mcmMetaDataCreationTime : TimeAndDate[1..1]	YES	This is a TimeAndDate attribute; it contains a datestamp and a timestamp. It defines the date and time that the MCMMetaData was created. **[D148]** This attribute **SHOULD** have a complete and valid time and/or date. **[O63]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
mcmMetaDataDescriptiveText : String[0..1]	NO	This attribute is a free-form textual string, and is used to contain prescriptive content about the MCMEntity or MCMInformationResource to which it is attached.

Table 53. Attributes of the MCMMetaData Class

Table 54 defines following operations for this class:

Operation Name	Description
getMCMMetaDataEnableStatus() : MCMMetaDataEnableStatus[1..1]	This method returns the value of the mcmMetaDataEnableStatus attribute. The output is an Enumeration of type MCMMetaDataEnableStatus.
setMCMMetaDataEnableStatus (in newStatus : MCMMetaDataEnableStatus1..1])	This method defines the value of the mcmMetaDataEnableStatus attribute. A single input parameter, of type MCMMetaDataEnableStatus, is supplied; the value is set to one of its literal values.
getMCMMetaDataCreationTime() : TimeAndDate[1..1]	This method returns the value of the mcmMetaDataCreationTime attribute. **[D149]** This attribute SHOULD have a complete and valid time and/or date. **[O64]** The implementation **MAY** ensure that the fields in this data type are set to an appropriate default value.
setMCMMetaDataCreationTime(in newTime : TimeAndDate[1..1])	This method defines the value of the mcmMetaDataCreationTime attribute. A single input parameter, of type TimeAndDate, is supplied.

getMCMMetaDataDescriptiveText() : String[1..1]	This method returns the value of the mcmMetaDataDescriptiveText attribute.
setMCMMetaDataDescriptiveText (in newStatus : String[1..1])	This method defines the value of the mcmMetaDataDescriptiveText attribute. A single input parameter, of type String, is supplied.

Table 54. Operations of the MCMMetaData Class

The MCMMetaData class participates in three aggregations: MCMEntityHasMCMMetaData, MCMInfoResourceHasMCMMetaData, and MCMHasMCMMetaDataDecorator. See sections 7.4.1, 7.10, and 7.11.6, respectively.

Figure 25 shows a portion of the MCMMetaData class hierarchy. The following subsections will use this figure to describe the classes in the MCMMetaData class hierarchy in more detail.

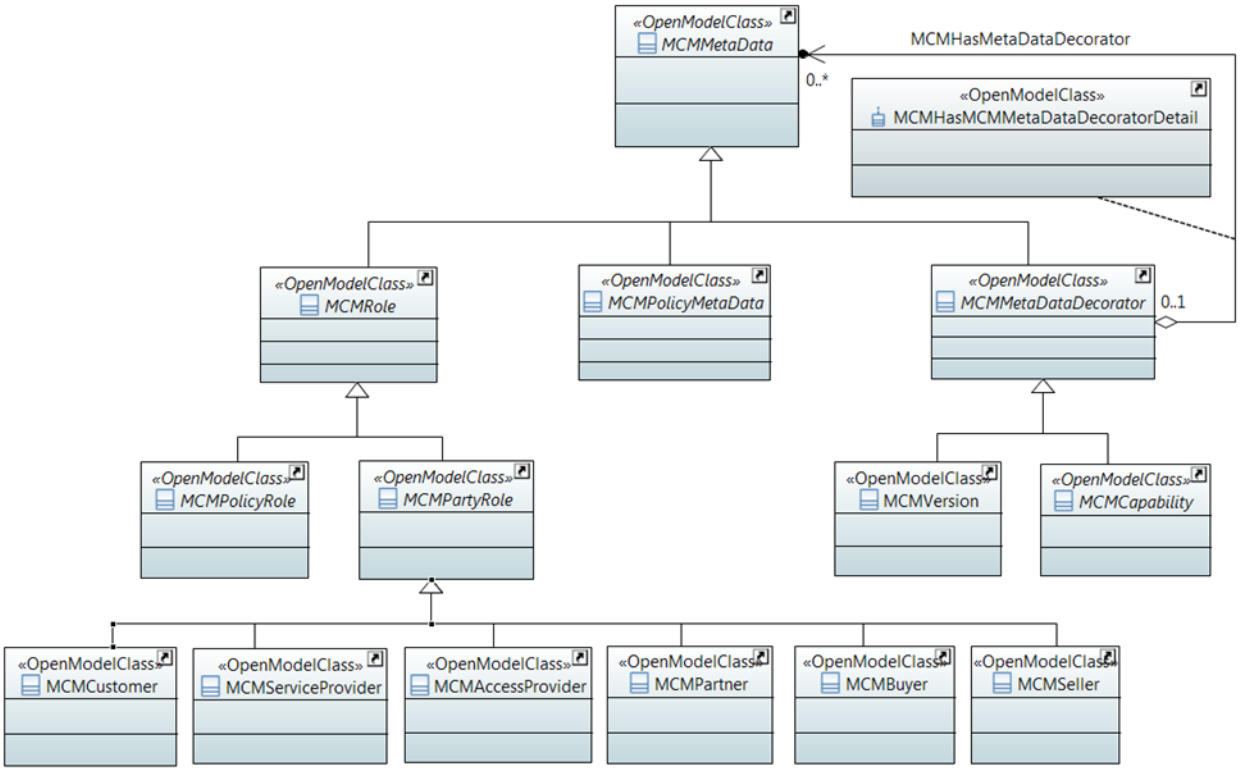

Figure 25. MCMMetaData Class Hierarchy, Part 2

7.11.2 MCMRole Class Hierarchy

This section specifies the class definition of the MCMRole class and its subclasses.

7.11.2.1 MCMRole Class Definition

This is an abstract class, and specializes MCMMetadata. It represents a set of characteristics and behaviors (also referred to as responsibilities) that an object takes on in a particular context. This enables an object to adapt to the needs of different clients through transparently attached role objects (as opposed to having to alter the inherent nature of the object itself). The Role Object pattern models context-specific views of an object as separate role objects that are dynamically attached to and removed from the core object to which the MCMRole objects are attached.

An important concept when using MCMRoles is that of a *role combination*. A role combination defines the set of MCMRoles that are attached to a given object. Data mining mechanisms can be used to optimize the number of roles, permission assignments, and other factors. This subject is beyond the scope of this document; however, this is why the getRoleCombination method is provided by this class.

Table 55 defines the following attributes for this class:

Attribute Name	Mandatory?	Description
mcmRoleName : String[1..1]	YES	This is a string attribute. It contains the name of this Role object. The mcmRoleName attribute is different from the mcmCommonName attribute, because the former defines a user-friendly name that this instance is, while the latter defines a name by which this object is known. **[R76]** The mcmRoleName attribute **MUST NOT** be used as a naming attribute (i.e., to uniquely identify an instance of this object). **[R77]** The mcmRoleName attribute **MUST NOT** be empty or Null.

Table 55. Attributes of the MCMRole Class

Table 56 defines following operations for this class:

Attribute Name	Description
getMCMRoleName : String[1..1]	This method returns the name of this MCMRole object. The mcmRoleName attribute is different from the mcmCommonName attribute, because the former defines a user-friendly name that this instance is, while the latter defines a name by which this object is known. **[R78]** The mcmRoleName attribute **MUST NOT** be used as a naming attribute (i.e., to uniquely identify an instance of this object). **[R79]** The mcmRoleName attribute **MUST NOT** be empty or Null string.
setMCMRoleName (in newRoleName : String[1..1])	This method returns the name of this MCMRole object. The mcmRoleName attribute is different from the mcmCommonName attribute, because the former defines a user-friendly name that this instance is, while the latter defines a name by which this object is known. **[R80]** The mcmRoleName attribute **MUST NOT** be used as a naming attribute (i.e., to uniquely identify an instance of this object). **[R81]** The mcmRoleName attribute **MUST NOT** be empty or Null string.

Table 56. Operations of the MCMRole Class

At this time, no relationships are defined for this class.

7.11.2.2 MCMPartyRole Class Definition

This is an abstract class, and specializes MCMRole. It represents a set of unique behaviors played by an MCMParty in a given context.

[D150] Implementers **SHOULD** use the Role-Object pattern [3] to implement MCMRoles.

At this time, no attributes are defined for this class.

At this time, no operations are defined for this class. Note that the getMCMPartyRoleList, setMCMPartyRoleList, setMCMPartyRolePartialList, delMCMPartyRoleList, and delMCMPartyRolePartialList operations are defined for an MCMParty; see section 7.11.2.2.

The MCMPartyRoleDetailHasMCMContact is an association between the MCMPartyRoleDetail association class and the MCMContact class. This association defines the set of MCMContacts that are related to this particular MCMPartyRoleDetail object (i.e., the set of MCMParty objects that are playing a specific MCMPartyRole). For example, this association can be used to define the contact information for a set of MCMParty objects that are each playing a set of MCMPartyRoles; a common use is to define the contact information for different types of MCMBuyer and MCMSeller objects.

The multiplicity of this association is 0..* – 0..*. This means that this association is optional (i.e., the "0" part of the 0..* cardinality). If this association is instantiated (e.g., the "0..* cardinality on the MCMPartyHasMCMPartyRoleDetail is greater than 0), then zero or more MCMContact objects can be associated with this particular MCMPartyHasMCMPartyRoleDetail object. Note that the cardinality on the part side (MCMContact) is 0..*; this enables an MCMPartyHasMCMPartyRoleDetail object to be defined without having to define an associated MCMContact object for it to aggregate.

The semantics of this aggregation are defined by the MCMPartyRoleDetailHasMCMContactDetail association class. This enables the management system to control which set of MCMPartyRoleDetail objects are associated to which set of MCMContact objects.

Note that the Policy Pattern may be used to control which specific part objects (i.e., MCMMetaData) are associated with which specific aggregate (i.e., MCMInformationResource) objects, respectively, for a given context. See Figure 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.11.2.3 MCMCustomer Class Definition

MCMCustomer is a concrete class, and specializes MCMPartyRole. It represents a particular type of MCMPartyRole that defines a set of people and/or organizations that buy, manage, or use MCMProducts from an MCMServiceProvider. The MCMCustomer is financially responsible for purchasing an MCMProduct. The MCMCustomer is the MCMPartyRole that is purchasing, managing, and/or using Services from an MCMServiceProvider. This definition is based on the definition from [15].

Table 57 defines the attributes of the MCMCustomer class.

Attribute Name	Mandatory?	Description
mcmCustomerStatus : MCMCustomerStatus[1..1]	YES	This attribute defines the current standing of a customer. Values are defined by the MCMCustomerStatus enumeration, and include the following literals: 0: ERROR 1: INIT 2: Active 3: Restricted (active with unpaid bills) 4: Inactive 5: Prospective
mcmCustomerRank : Integer[0..1]	NO	This is a non-negative integer, and defines the current business importance of this Customer. A value of 0 means unimportant, and higher positive values means higher importance.

Table 57. Attributes of the MCMCustomer Class

Table 58 defines following operations for this class:

Attribute Name	Description
getMCMCustomerStatus () : MCMCustomerStatus[1..1]	This method returns the value of the mcmCustomerStatus attribute. The values of this attribute are defined in theMCMCustomerStatus enumeration.
setMCMCustomerStatus (in newStatus: MCMCustomerStatus[1..1])	This method defines the value for the mcmCustomerStatus attribute. Valid values for this attribute are defined in the theMCMCustomerStatus enumeration.
getMCMCustomerRank () : Integer[1..1]	This method returns the value of the mcmCustomerRank attribute.

setMCMCustomer-Rank (in newRank : Integer[1..1])	This method defines the value for the mcmCustomerRank attribute.

Table 58. Operations of the MCMCustomer Class

At this time, no relationships are defined for this class.

7.11.2.4 MCMServiceProvider Class Definition

MCMServiceProvider is a concrete class, and specializes MCMPartyRole. It represents a particular type of MCMPartyRole that provides MCMProducts. This specifically includes MCMServices. This definition is based on the definition from [1].

At this time, no attributes are defined for this class. Most attributes will likely be realized using relationships and/or methods. For example, a query to an instance of the MCMServiceProvider class to provide its set of different contact information will be done by using a class method, since each contact will also use information from a subclass of MCMContact (see section 7.10.3).

At this time, no operations are defined for this class.

At this time, no relationships are defined for this class.

7.11.2.5 MCMAccessProvider Class Definition

MCMAccessProvider is a concrete class, and specializes MCMPartyRole. It represents a particular type of MCMPartyRole that enables MCMPartyRoles (typically MCMCustomers) to gain entrance to a network (e.g., the Internet), by using an MCMProduct. This specifically includes MCMServices.

At this time, no attributes are defined for the MCMAccessProvider class. Most attributes will likely be realized using relationships and/or methods. For example, a query to an instance of the MCMAccessProvider class to provide its set of different contact information will be done by using a class method, since each contact will also use information from a subclass of MCMContact (see section 7.10.3).

At this time, no operations are defined for this class.

At this time, no relationships are defined for this class.

7.11.2.6 MCMPartner Class Definition

MCMPartner is a concrete class, and specializes MCMPartyRole. It represents a particular type of MCMPartyRole that provides MCMProducts and MCMServices to the MCMServiceProvider in order to instantiate and manage MCMService elements, such as MCMServiceComponents, external to the Service Provider's Domain. This definition is based on the definition from [1].

At this time, no attributes are defined for the MCMPartner class. Most attributes will likely be realized using relationships and/or methods. For example, a query to an instance of the MCMPartner class to provide its set of different contact information will be done by using a class method, since each contact will also use information from a subclass of MCMContact (see section 7.10.3).

At this time, no operations are defined for this class.

At this time, no relationships are defined for this class.

7.11.3 MCMPolicyRole Class Definition

This class is defined in the Policy Driven Orchestration specification. It is used to define the descriptive and/or prescriptive characteristics and behavior of a given MCMPolicyRole object.

7.11.4 MCMPolicyMetaData Class Definition

This is an abstract class, and specializes MCMMetaData. It is used to define MetaData for all types of Policy Driven Orchestration Policies (e.g., imperative, declarative, and intent).

7.11.5 MCMGeoSpatialMetaData Class Definition

This is an abstract class, and specializes MCMMetaData. It defines metadata that are applicable to objects that have an explicit or implicit geographic meaning (e.g., they are associated with a particular location, typically on the surface of the Earth). This class will eventually be harmonized with all or some of the information in [14].

Table 59 defines the attributes of this class.

Attribute Name	Mandatory?	Description
mcmIsAbsoluteLocation : Boolean[0..1]	NO	This is a Boolean attribute. If its value is TRUE, then this MCMLocation object instance provides an absolute geo-location. Otherwise, it provides an estimated geo-location.
mcmIsPlannedLocation : Boolean[0..1]	NO	This is a Boolean attribute. If the value of this attribute is TRUE, then an MCMUnManagedEntity is planned to be located here. Otherwise, this is the current location of an MCMLocation object.

mcmGeoMethod : MCMGeoMethod[1..1]	YES	This is an enumerated string attribute, and defines the type of geolocation method used. The values are literals in the MCMGeoMethod enumeration. Values include: - 0: ERROR - 1: INIT - 2: GPS - 3: Differential GPS - 4: Augmented GNSS - 5: Enhanced GNSS - 6: Non-GPS Satellite Navigation - 7: Cellular Navigation - 8: WiFi Positioning - 9: Other Positioning System

Table 59. Attributes of the MCMGeoSpatialMetaData Class

Table 60 defines the operations of this class.

Attribute Name	Description
getMCMIsAbsoluteLocation() : Boolean[1..1])	This method returns the value of the mcmIsAbsoluteLocation attribute. **[D151]** If the value of the mcmIsAbsoluteLocation attribute is empty or NULL, then the implementation **SHOULD** return FALSE. **[D152]** The default value for this attribute **SHOULD** be FALSE.
setMCMIsAbsoluteLocation(in isAbsolute : Boolean[1..1])	This method defines the value for the mcmIsAbsoluteLocation attribute.
getMCMIsPlannedLocation() : Boolean[1..1])	This method returns the value of the mcmIsPlannedLocation attribute. **[D153]** If the value of the mcmIsAbsoluteLocation attribute is empty or NULL, then the implementation **SHOULD** return FALSE. **[D154]** The default value for this attribute **SHOULD** be FALSE.
setMCMIsPlannedLocation(in isPlanned : Boolean[1..1])	This method defines the value for the mcmIsPlannedLocation attribute.
getMCMGeoMethod() : MCMGeoMethod[1..1]	This method returns the value of the mcmGeoMethod attribute. Valid values for this attribute are defined in the MCMGeoMethod enumeration.

setMCMGeoMethod(in newGeoMethod : MCMGeoMethod[1..1])	This method defines the value for the mcmIsAbsoluteLocation attribute. Valid values for this attribute are defined in the MCMGeoMethod enumeration.

Table 60. Operations of the MCMGeoSpatialMetaData Class

At this time, no relationships are defined for this class.

7.11.6 MCMMetaDataDecorator Class Definition

This is an abstract class, and specializes MCMMetaData. It defines the decorator pattern for use with MCMMetaData. This enables all or part of one or more concrete subclasses of MCMMetaDataDecorator to "wrap" another concrete subclass of MCMMetaData.

At this time, no attributes are defined for the MCMMetaDataDecorator class.

Table 61 defines the operations of this class.

Attribute Name	Description
getMCMMetaDecoratorList() : MCMMetaDataDecorator[1..*]	This operation returns the set of MCMMetaDataDecorator objects that are decorating this MCMMetaData object. There are no input parameters. **[D155]** If this MCMMetaData object is not decorated by any MCMMetaDataDecorator objects, then a NULL MCMMetaDataDecorator object **SHOULD** be returned.
setMCMMetaDecoratorList (in newDecoratorList : MCMMetaDataDecorator[1..*])	This operation defines the set of MCMMetaDataDecorator objects that will decorate this MCMMetaData object. This operation takes a single input parameter, called newDecoratorList, which is of type MCMMetaDataDecorator. This operation creates a set of aggregations between this particular MCMMetaData object and the set of MCMMetaDataDecorator objects identified in the input parameter. Note that this operation first deletes any existing MCMMetaDataDecorator objects (and their aggregations and association classes) that decorate this MCMMetaData object, and then instantiates a new set of MCMServiceComponent objects; in doing so, each MCMMetaDataDecorator object is attached to this particular MCMMetaData object by first, creating an instance of the MCMHasMetaDataDecorator aggregation, and second, realizing that aggregation instance as an association class. **[D156]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasMetaDataDecoratorDetail association class).
setMCMMetaDecoratorPartialList (in newDecoratorList : MCMMetaDataDecorator[1..*])	This operation defines a set of one or more MCMMetaDataDecorator objects that will decorate this MCMMetaData object WITHOUT affecting any other existing MCMMetaDataDecorator objects that are decorating this MCMMetaData object. This operation takes a single input parameter, called newDecoratorList, which is an array of one or more MCMMetaDataDecorator objects. This operation creates a

	set of aggregations between this particular MCMMetaData object and the set of MCMMetaDataDecorator objects identified in the input parameter. **[D157]** Each created aggregation **SHOULD** have an association class (i.e., an instance of the MCMHasMetaDataDecoratorDetail association class).
delMCMMetaDecoratorList()	This operation deletes ALL MCMMetaDataDecorator object instances that are decorating this MCMMetaData object. This operation first removes the association class, and second, removes the aggregation, between this MCMMetaData object and each MCMMetaDataDecorator object that is decorating this MCMMetaData object. This operation has no input parameters. This operation does not delete any of the MCMMetaDataDecorator objects; it simply disconnects them from the MCMMetaData that they were decorating.
delMCMMetaDecoratorPartialList (in newDecoratorList : MCMMetaDataDecorator [1..*])	This operation deletes a set of MCMMetaDataDecorator objects that are decorating this particular MCMMetaData object. This operation takes a single input parameter, called newDecoratorList, which is an array of one or more MCMMetaDataDecorator objects. This operation first removes the association class and second, removes the aggregation, between each MCMMetaDataDecorator object specified in the input parameter and this MCMMetaData object. Note that all other aggregations between this MCMMetaData object and other MCMMetaDataDecorator objects that are not specified in the input parameter are NOT affected.

Table 61. Operations of the MCMMetaDataDecorator Class

At this time, a single aggregation is defined for MCMMetaDataDecorator. This aggregation is named MCMHasMetaDataDecorator, and defines the set of concrete subclasses of MCMMetaDataDecorator that wrap (or decorate) a concrete subclass of MCMMetaData. The multiplicity of this aggregation is 0..1 – 0..*. This means that this aggregation is optional (i.e., the "0" part of the 0..1 cardinality). If this aggregation is instantiated (e.g., the "1" part of the 0..1 cardinality), then zero or more concrete subclasses of MCMMetaDataDecorator can decorate (i.e., "wrap") this particular concrete subclass of MCMMetaData. Note that the cardinality on the part side (MCMMetaData) is 0..*; this enables an MCMMetaData object to be defined without having to define an associated MCMMetaDataDecorator object.

The semantics of this aggregation are defined by the MCMHasMetaDataDecoratorDetail association class. This enables the management system to control which set of concrete subclasses of MCMMetaDataDecorator wrap this particular MCMMetaData. The Policy Pattern may be used to control which specific MCMMetaDataDecorator objects wrap a given MCMMetaData for a given

context. See Table 3 for an exemplary illustration of the Policy Pattern. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

7.11.6.1 MCMCapability Class Definition

This is an abstract class, and specializes MCMMetaDataDecorator. It represents a set of features that are available to be used from an MCMEntity. These features may include all, or a subset, of the available features of an MCMEntity. These features may, but do not have to, be used.

At this time, no attributes are defined for the MCMCapability class. Most attributes will likely be realized using relationships and/or methods. For example, the set of mandatory, recommended, and optional capabilities of a given MCMManagedEntity can be gathered and sorted by using an appropriate method.

At this time, no relationships are defined for this class.

At this time, no relationships are defined for this class.

7.11.6.2 MCMNetworkFunction

This section describes the concept of a Network Function. This was originally defined by ETSI NFV, but has been modified to make this concept both more flexible and generic as well as more robust (e.g., in NFV, it is not explicitly modeled).

7.11.6.2.1 Background

This class is derived from the concept of a Network Function as defined in ETSI NFV [17]:

> *Network Function (NF): functional block within a network infrastructure that has well-defined external interfaces and well-defined functional behavior.*

In control theory, a system is made up of functional blocks. A functional block describes a part, or module, of a system. Each functional block defines a collection of structural and/or behavioral features of a module. A transfer function defines the set of outputs for a functional block given a set of inputs and a state. Originally, NFV defined a NetworkFunction as a transfer function. This is no longer true.

7.11.6.2.2 Rationale for Changing the Definition of a NetworkFunction

There are three main reasons for not using the ETSI definition of a NetworkFunction:

- The ETSI NFV information model does not define a superclass for a NetworkFunction
- In ETSI NFV, the concept of a NetworkFunction is limited to a small number of use cases; we want a NetworkFunction to behave as defined in [17], and be used to represent the behavior defined by the combination of a state and a given set of inputs
- The ETSI NFV information model does not specifically model a NetworkFunction

Hence, we have decided to create a new MCM class to better model what a NetworkFunction is.

Accordingly, the name of this class is prefixed with "MCMMEF", to denote that this is the MEF's interpretation of what a NetworkFunction should be.

Note also that, due to this definition, MCMMEFNetworkFunctions may be attached to any MCMManagedEntity. This is not true in NFV.

Finally, we have modeled an MCMMEFNetworkFunction as a type of MCMMetaData. This is because:

- A NetworkFunction is not required to be used
- A NetworkFunction may change its behavior, adding or removing capabilities dynamically at runtime

Therefore, we have modeled an MCMMEFNetworkFunction as a subclass of MCMCapability (since it represents the capabilities of an MCMEntity), which is in turn a subclass of MCMMetaDataDecorator (since it can be dynamically changed at runtime).

7.11.6.2.3 MCMMEFNetworkFunction Class Definition

This is a concrete class, and specializes MCMCapability. It represents the features and behavior of an MCMManagedEntity that may be used for a given set of external interfaces while in a particular state. It may specify attributes and methods, as well as define nested MCMMEFNetworkFunctions. It may also enumerate the actors that use it.

At this time, no attributes are defined for the MCMMEFNetworkFunction class.

At this time, no operations are defined for the MCMMEFNetworkFunction class.

At this time, no relationships are defined for the MCMMEFNetworkFunction class.

7.11.6.3 MCMMEFDescriptor

This section describes the concept of a Descriptor. This was originally defined by ETSI NFV, but has been modified to make this concept both more flexible and generic as well as more robust (e.g., in NFV, there are many different types of Descriptors, but each is modeled as an individual object and does not have a superclass).

7.11.6.3.1 Background

A Descriptor is loosely described as a "template", as in this example from [17]:

> *network service descriptor: template that describes the deployment of a Network Service including service topology (constituent VNFs and the relationships between them, Virtual Links, VNF Forwarding Graphs) as well as Network Service characteristics such as SLAs and any other artefacts necessary for the Network Service on-boarding and lifecycle management of its instances*

There are a large number of descriptors defined in ETSI NFV.

7.11.6.3.2 Rationale for Changing the Defintiion of a Descriptor

First, there is a large amount of commonality, in both function and purpose, among the many different types of Descriptors used in ETSI NFV. Unfortunately, the NFV information model treats them as individual objects with no common inheritance. This needs to be fixed to follow accepted object-oriented practice, and to make the models more robust and easier to maintain.

Second, many NFV descriptors contain other NFV descriptors – this is another reason to enforce inheritance and represent descriptors using a class hierarchy. In addition, the use of various patterns, such as the composite pattern [2][4], would significantly simplify the resulting design as well as improve its robustness and decrease its fragility.

Third, some types of NFV descriptors contain instance-specific information (e.g., link data) that is fragile and will change during normal operations (e.g., a VM migration). In order to be compatible with this approach, the MCM models descriptors as a type of metadata that can be dynamically attached and detached using the decorator pattern.

Fourth, many types of NFV descriptors combine metadata with other types of data. The MCM separates these two types of data into separate class hierarchies (e.g., MCMEntity vs MCMMetaData), but enables them to be associated with each other. This provides a better and more consistent implementation approach.

Finally, descriptors are used inconsistently in NFV. While NFV is basically a resource-oriented model, not all resources have descriptors. In addition, descriptors should be able to be used for other entities, such as Services. This is the prime motivation for subclassing MCMMEFDescriptor from MCMCapability (which is a type of MCMMetaDataDecorator – that way, a Descriptor can change dynamically to suit the needs of what it is describing).

7.11.6.3.3 MCMMEFDescriptor Class Definition

An MCMMEFDescriptor is a set of related metadata that can be applied to describe and/or prescribe the characteristics and behavior of an MCMManagedEntity. Note that this class can be used in conjunction with an appropriate subclass of an MCMDefinition class to provide a completely generic mechanism for defining the salient characteristics and behavior of a Descriptor. In addition, the flexibility of the MCM enables the application developer to tailor application-specific definitions of Descriptors to Products, Services, and/or Resources. For example, the MCMMEFDescriptor class can be attached to an MCMService class, which is defined by a set of MCMDefinitions.

There are several significant problems with the NFV definition of a descriptor. First, it is restricted to the deployment view, yet is often used as part of the design process. Second, it combines metadata and non-metadata information. For example, the definition of a VNF Descriptor (VNFD) in NFV is a “configuration template that describes a VNF in terms of its deployment and operational behavior, and is used in the process of VNF on-boarding and managing the lifecycle of a VNF instance”. Hence, the name of this class is prefixed with “MCMMEF”, to denote that this is the MEF’s interpretation of what a Descriptor should be.

At this time, no attributes are defined for the MCMMEFDescriptor class.

At this time, no operations are defined for the MCMMEFDescriptor class.

At this time, no relationships are defined for the MCMMEFDescriptor class.

7.11.6.4 MCMVersion Class Definition

This is a concrete class that specializes MCMMetaDataDecorator. It defines versioning information, in the form of metadata, that can be added to an MCMManagedEntity. This enables all or part of a standardized description and/or specification of version information for a given MCMManagedEntity to be easily changed at runtime by wrapping an object instance of the MCMManagedEntity class (or its subclass) with all or part of this object.

Version information is defined in a generic format based on the Semantic Versioning 2.0.0 Specification [16] as follows:

<major>.<minor>.<patch>[<pre-release>][<build-metadata>]

where the first three components (major, minor, and patch) MUST be present, and the latter two components (pre-release and build-metadata) MAY be present. A version number MUST take the form <major>.<minor>.<patch>, where <major>, <minor>, and <patch> are each non-negative integers that MUST NOT contain leading zeros.

In addition, the value of each of these three elements MUST increase numerically. In this approach:

- mcmVersionMajor denotes a new release; this number MUST be incremented when either changes are introduced that are not backwards-compatible, and/or new functionality not previously present is introduced
- mcmVersionMinor denotes a minor release; this number MUST be incremented when new features and/or bug fixes to a major release that are backwards-compatible are introduced, and/or if any features are marked as deprecated
- mcmVersionPatch denotes a version that consists ONLY of bug fixes, and MUST be incremented when these bug fixes are NOT backwards-compatible

When multiple versions exist, the following rules define their precedence:

1) Precedence MUST be calculated by separating the version into major, minor, patch, and pre-release identifiers, in that order. Note that build-metadata is NOT used to calculated precedence.

2) Precedence is determined by the first difference when comparing each of these identifiers, from left to right, as follows:

   a) Major, minor, and patch versions are always compared numerically (e.g., 1.0.0 < 2.0.0 < 2.1.0 < 2.1.1)

b) When major, minor, and patch are equal, a pre-release version has LOWER precedence than a normal version (e.g., 1.0.0-alpha < 1.0.0)

c) Precedence for two pre-release versions with the same major, minor, and patch version MUST be determined by comparing each dot separated identifier from left to right until a difference is found as follows:

i) identifiers consisting only of digits are compared numerically and identifiers with letters and/or hyphens are compared lexically in ASCII sort order

ii) Numeric identifiers always have lower precedence than non-numeric identifiers

iii) A larger set of pre-release fields has a higher precedence than a smaller set, if all of the preceding identifiers are equal

Example:

1.0.0-alpha < 1.0.0-alpha.1 < 1.0.0-alpha-beta < 1.0.0-beta < 1.0.0-beta.2 < 1.0.0-rc.1 < 1.0.0.

Table 62 defines the attributes of the MCMVersion class.

Attribute Name	Mandatory?	Description
mcmVersionMajor : String[1..1]	YES	This is a mandatory string attribute, and contains a string representation of an integer that is greater than or equal to zero. It indicates that a significant increase in functionality is present in this version. Improvements to each starting initial version, before they are released to the public, are denoted by incrementing the minor and patch version numbers. **[O65]** A major version **MAY** indicate that this version has changes that are NOT backwards-compatible; this **MAY** be denoted using attached MCMMetaData and/or using the mcmVersionBuildMetaData class attribute. **[R82]** The special string "0.1.0" is for initial development that **MUST NOT** be considered stable. **[R83]** The major version X (i.e., X.y.z, where X > 0) **MUST** be incremented if any backwards incompatible changes are introduced. **[O66]** A major version **MAY** include minor and patch level changes. **[R84]** The minor and patch version numbers **MUST** be reset to 0 when the major version number is incremented.

mcmVersionMinor : String[1..1]	YES	This is a mandatory string attribute, and contains a string representation of an integer that is greater than or equal to zero. A minor version indicates that this release contains a set of features and/or bug fixes that are backwards-compatible. **[R85]** A minor version indicates that this release contains a set of features and/or bug fixes that **MUST** be backwards-compatible. **[R86]** The minor version Y (i.e., x.Y.z, where x > 0) **MUST** be incremented if new, backwards-compatible changes are introduced. **[R87]** The minor version **MUST** be incremented if any features are marked as deprecated. **[O67]** The minor version **MAY** be incremented if new functionality or improvements are introduced. **[O68]** The minor version **MAY** include patch level changes. **[R88]** The patch version number **MUST** be reset to 0 when the minor version number is incremented.
mcmVersionPatch : String[1..1]	YES	This is a mandatory string attribute, and contains a string representation of an integer that is greater than or equal to zero. A patch version indicates that this version ONLY contain bug fixes. A bug fix is defined as an internal change that fixes incorrect behavior. **[R89]** A patch version indicates that this version **MUST** ONLY contain bug fixes. **[R90]** The patch version Z (i.e., x.y.Z, where x > 0) **MUST** be incremented if new, backwards-compatible changes are introduced.
mcmVersionPreRelease : String[0..1]	NO	This is an optional string attribute, and contains a string defining the pre-release version. A pre-release version is denoted by appending a hyphen and a series of dot-separated identifiers immediately following the patch version. A pre-release version indicates that the version is unstable and might not satisfy the intended compatibility requirements as denoted by its associated normal version. Pre-release versions have a lower precedence than the associated normal version. Examples include: 1.0.0-alpha, 1.0.0-alpha.1, 1.0.0-0.3.7, and 1.0.0-x.7.z.92. **[R91]** Identifiers **MUST** comprise only ASCII alphanumerics and a hyphen.

		[R92] Identifiers **MUST NOT** be empty. **[R93]** Numeric identifiers **MUST NOT** include leading zeroes.
mcmVersionBuildMetaData : String[0..1]	NO	This is an optional string attribute, and contains a string defining the build metadata. Build metadata is denoted by appending a plus sign and a series of dot-separated identifiers immediately following the patch or pre-release version. Examples include: 1.0.0.-alpha+1, 1.0.0+20130313144700, and 1.0.0-beta+exp.sha.5114f85. **[R94]** Identifiers **MUST** be made up of only ASCII alphanumerics and a hyphen. **[R95]** Identifiers **MUST NOT** be empty. **[D158]** Build metadata **SHOULD** be ignored when determining version precedence.

Table 62. Attributes of the MCMVersion Class

Table 63 defines the following operations for this class.

Operation Name	Description
getMCMVersionMajor() : String[1..1]	This method returns the value of the mcmVersionMajor attribute. This value is a string representation of an integer that is greater than or equal to zero. It indicates that a significant increase in functionality is present in this version. Improvements to each starting initial version, before they are released to the public, are denoted by incrementing the minor and patch version numbers. **[R96]** The value of this attribute **MUST NOT** be a NULL or an empty string value.
setMCMVersionMajor(in newVersionMajor : String[1..1])	This method defines the value of the mcmVersionMajor attribute. A single input parameter, of type String, is provided. This value is a string representation of an integer that is greater than or equal to zero. It indicates that a significant increase in functionality is present in this version. Improvements to each starting initial version, before they are released to the public, are denoted by incrementing the minor and patch version numbers. **[R97]** The value of this attribute **MUST NOT** be a NULL or an empty string value.
getMCMVersionMinor() : String[1..1]	This method returns the value of the mcmVersionMinor attribute. This value a string representation of an integer that is greater than

	or equal to zero. A minor version indicates that this release contains a set of features and/or bug fixes that are backwards-compatible. **[R98]** The value of this attribute **MUST NOT** be a NULL or an empty string value.
setMCMVersionMinor(in newVersionMinor : String[1..1])	This method defines the value of the mcmVersionMinor attribute. A single input parameter, of type String, is provided. This value is a string representation of an integer that is greater than or equal to zero. A minor version indicates that this release contains a set of features and/or bug fixes that are backwards-compatible. **[R99]** The value of this attribute **MUST NOT** be a NULL or an empty string value.
getMCMVersionPatch() : String[1..1]	This method defines the value of the mcmVersionPatch attribute. This value is a string representation of an integer that is greater than or equal to zero. A patch version indicates that this version ONLY contain bug fixes. A bug fix is defined as an internal change that fixes incorrect behavior. **[R100]** The value of this attribute **MUST NOT** be a NULL or an empty string value.
setMCMVersionPatch(in newVersionPatch : String[1..1])	This method defines the value of the mcmVersionPatch attribute. A single input parameter, of type String, is provided. This value is a string representation of an integer that is greater than or equal to zero. A patch version indicates that this version ONLY contain bug fixes. A bug fix is defined as an internal change that fixes incorrect behavior. **[R101]** The value of this attribute **MUST NOT** be a NULL or an empty string value.
getMCMVersionPreRelease() : String[1..1]	This method defines the value of the mcmVersionPreRelease attribute. This value is a string defining the pre-release version. A pre-release version is denoted by appending a hyphen and a series of dot-separated identifiers immediately following the patch version. A pre-release version indicates that the version is unstable and might not satisfy the intended compatibility requirements as denoted by its associated normal version. Pre-release versions have a lower precedence than the associated normal version. Examples include: 1.0.0-alpha, 1.0.0-alpha.1, 1.0.0-0.3.7, and 1.0.0-x.7.z.92. **[R102]** Identifiers **MUST NOT** be empty. **[R103]** The value of this attribute **MUST NOT** be a NULL or an empty string value.

setMCMVersionPreRelease(in newVersionPreRelease : String[1..1])	This method defines the value of the mcmVersionPreRelease attribute. A single input parameter, of type String, is provided. This value is a string defining the pre-release version. A pre-release version is denoted by appending a hyphen and a series of dot-separated identifiers immediately following the patch version. A pre-release version indicates that the version is unstable and might not satisfy the intended compatibility requirements as denoted by its associated normal version. Pre-release versions have a lower precedence than the associated normal version. Examples include: 1.0.0-alpha, 1.0.0-alpha.1, 1.0.0-0.3.7, and 1.0.0-x.7.z.92. **[R104]** Identifiers **MUST NOT** be empty. **[R105]** The value of this attribute **MUST NOT** be a NULL or an empty string value.
getMCMVersionBuildMetaData : String[1..1]	This method defines the value of the mcmVersionBuildMetaData attribute. This value is a string defining the build metadata. Build metadata is denoted by appending a plus sign and a series of dot-separated identifiers immediately following the patch or pre-release version. Examples include: 1.0.0.-alpha+1, 1.0.0+20130313144700, and 1.0.0-beta+exp.sha.5114f85. **[R106]** Identifiers **MUST** be made up of only ASCII alphanumerics and a hyphen. **[R107]** Identifiers **MUST NOT** be empty. **[R108]** The value of this attribute **MUST NOT** be a NULL or an empty string value.
setMCMVersionBuildMetaData(in newVersionBuild : String[1..1])	This method defines the value of the mcmVersionBuildMetaData attribute. A single input parameter, of type String, is provided. This value is a string defining the build metadata. Build metadata is denoted by appending a plus sign and a series of dot-separated identifiers immediately following the patch or pre-release version. Examples include: 1.0.0.-alpha+1, 1.0.0+20130313144700, and 1.0.0-beta+exp.sha.5114f85. **[R109]** Identifiers **MUST** be made up of only ASCII alphanumerics and a hyphen. **[R110]** Identifiers **MUST NOT** be empty. **[R111]** The value of this attribute **MUST NOT** be a NULL or an empty string value.

Table 63. Operations of the MCMVersion Class

At this time, no relationships are defined for this class.

8 References

[1] MEF Forum, Lifecycle Service Orchestration: Reference Architecture and Framework, MEF 55, March 2016

[2] Gamma, E., Helm, R. Johnson, R., Vlissides, J., “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley, Nov, 1994. ISBN 978-0201633610

[3] Bäumer, D. Riehle, W. Siberski, M. Wulf, “The Role Object Pattern”, Proceedings of the 1997 Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '97), ACM Press, 1997, Page 218-228

[4] Riehle, D., “Composite Design Patterns”, Proceedings of the 1997 Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '97), ACM Press, 1997, Page 218-228

[5] Liskov, B.H., Wing, J.M., “A Behavioral Notion of subtyping”, ACM Transactions on Programming languages and Systems 16 (6): 1811 - 1841, 1994

[6] Martin, R.C., "Agile Software Development, Principles, Patterns, and Practices", Prentice-Hall, 2002, ISBN: 0-13-597444-5

[7] Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels”, BCP 14, RFC 2119, March 1997

[8] Meyer, B., "Object-Oriented Software Construction”, Prentice Hall, second edition, 1997 ISBN 0-13-629155-4

[9] Schmidt, D.C., “Model-Driven Engineering”, IEEE Computer, 2006

[10] Strassner, J., Halpern, J., and van der Meer, S., “Generic Policy Information Model for Simplified Use of Policy Abstractions (SUPA)”, draft-ietf-supa-generic-policy-info-model-03, May 2017

[11] Object Management Group, OMG Unified Modeling Language TM (OMG UML), Version 2.5.1, December 2017.

[12] Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mee, R., Stafford, R., “Patterns of Enterprise Application Architecture”, Addison-Wesley, November, 2002

[13] MEF, “Ethernet Services Attributes Phase 3”, Technical Specification MEF 10.3, October 2013

[14] ISO, “Geographic Information – Metadata”, ISO 19115:2113

[15] ATIS and MEF, “Ethernet Ordering Technical Specification” (revision of MEF57/J-Spec-001), atis_mef_201700022R006, December 2017

[16] https://semver.org/

[17] ETSI, “ETSI GS NFV 003 v1.3.1; Network Functions Virtualisation; Terminology for Main Concepts in NFV”, January 2018

[18] Internet Engineering Task Force RFC 8174, “Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words”, May 2017

Appendix A Basic Mapping between the MCM and TMF Models

The following table defines a simplified association between concrete MCM classes and those used in TMF625 (TMF API Data Model, version 16.0.1, which is the current latest release). This mapping will be detailed and enhanced in a future release of the MCM.

MCM Concrete Class	Equivalent TMF Class	Comments
MCMContact	Contact	Significant semantic differences (e.g., MCMContact is an MCMInformation-Resource)
MCMCatalog	Catalog	Significant semantic differences (e.g., MCMCatalog aggregates MCMCata-logItems, which can be any type of MCMManagedEntity; contact info for Catalogs is significantly different)
MCMCatalogItem	No Equivalent	No Equivalent
MCMCustomer	Customer	Significant semantic differences (e.g., this uses the Role-Object pattern, TMF doesn't; compliant with MEF57.1, TMF isn't).
MCMServiceProvider	No Equivalent	No Equivalent
MCMAccessProvider	No Equivalent	No Equivalent
MCMPartner	PartnershipType and RoleType and unnamed composition	Significant semantic differences (e.g., MCMPartner uses the Role-Object pattern; TMF625 doesn't have a Part-ner class)
MCMBuyer	PartnershipType and RoleType and unnamed composition	TMF625 does not define this as a ded-icated class, as required in MEF57.1
MCMSeller	PartnershipType and RoleType and unnamed composition	TMF625 does not define this as a ded-icated class, as required in MEF57.1
MCMPerson	Individual	Significant semantic differences (e.g., MCMPerson is part of a Composite pattern; not used in TMF)
MCMOrganization	Organization	Same as MCMPerson
MCMProductFeature	No Equivalent	No Equivalent
MCMServiceFeature	No Equivalent	No Equivalent
MCMResourceFeature	No Equivalent	No Equivalent
MCMBusinessTerm	No Equivalent	No Equivalent
MCMProductOffer	ProductOffering	Requires use of the TMF specification pattern, which is not supported in the MCM
MCMResourceOffer	No Equivalent	No Equivalent

MCMServiceOffer	No Equivalent	No Equivalent
MCMProductAtomic	Product	Semantically different, since MCM uses the composite pattern and TMF625 doesn't
MCMProductComposite	Product plus ProductRelationship plus unnamed composition	Significant semantic differences, since MCM uses the composite pattern and TMF625 doesn't
MCMOrderedService	No Equivalent	No Equivalent
MCMInternalService	No Equivalent	No Equivalent
MCMServiceComponent	No Equivalent	No Equivalent
MCMServiceEndpoint	No Equivalent	No Equivalent
MCMServiceInterface	No Equivalent	No Equivalent
MCMMgmtDomainAtomic	No Equivalent	No Equivalent
MCMMgmtDomainComposite	No Equivalent	No Equivalent
MCMOrderAtomic	ProductOrder	Semantically different, since MCM is not restricted to ordering Products
MCMOrderItem	OrderItem	Semantically different, since MCM lacks many of the compositions that are in TMF625 (but these have been rejected in the latest MEF Ordering model)
MCMMEFNetworkFunction	No Equivalent	No Equivalent
MCMMEFDescriptor	No Equivalent	No Equivalent
MCMVersion	No Equivalent	No Equivalent

Table 64. Brief Comparison of MCM and TMF625 Classes