ETSI GS ENI 019 V3.1.1 (2023-06)

Experiential Networked Intelligence (ENI); Representing, Inferring, and Proving Knowledge in ENI

Disclaimer

The present document has been produced and approved by the Experiential Networked Intelligence (ENI) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
DGS/ENI-0029_ENI_Models

Keywords
data models, information model, ontology,
semantic reasoning

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (ISG) Experiential Networked Intelligence (ENI).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The purpose of the present document is to define the ENI information model. The present document will also provide at least two different examples of how to derive technology-specific data models from the ENI information model. Finally, it will explain how ontologies can be incorporated to augment, enhance, and specify meaning and different relationships between modelled entities. This latter is critical to provide semantic reasoning. The present document is specific to and enhances the current ENI System Architecture.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI GS ENI 001 (V3.1.1): "Experiential Networked Intelligence (ENI); ENI use cases".

[2] ETSI GS ENI 002 (V3.1.1): "Experiential Networked Intelligence (ENI); ENI requirements".

[3] ETSI GS ENI 005 (V3.1.1): "Experiential Networked Intelligence (ENI); System Architecture".

[4] MEF 78.1: "MEF Core Model (MCM)", Strassner J., editor, July 2020.

[5] MEF 95: "MEF Policy Driven Orchestration", Strassner J., editor, July 2021.

[6] OMG: "OMG Meta Object Facility (MOF) Core Specification", version 2.5.1, October 2019.

[7] MEF 95.0.1: "Amendment to MEF 95: Policy Driven Orchestration", October 2022.

[8] The Semantic Versioning specification.

[9] NIST SP 1800-161: "Cybersecurity Supply Chain Risk Management Practices for Federal Information Systems and Organizations", May 2022.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] Strassner. J.: "Knowledge Representation, Processing, and Governance in the FOCALE Autonomic Architecture", book chapter, 2011, Elsevier.

[i.2] Strassner, J.: "Policy-Based Network Management", Morgan Kaufman, ISBN 978-1558608597, September 2003.

[i.3] Strassner, J., de Souza, J.N., Raymer, D., Samudrala, S., Davy, S., Barrett, K.: "The Design of a Novel Context-Aware Policy Model to Support Machine-Based Learning and Reasoning", Journal of Cluster Computing, Vol 12, Issue 1, pages 17-43, March, 2009.

[i.4] Strassner, J., van der Meer, S., O'Sullivan, D., and Dobson, S.: "The Use of Context-Aware Policies and Ontologies to Facilitate Business-Aware Network Management", Journal of Network and Systems Management 17(3), pages 255-284, 2009.

[i.5] ETSI GR ENI 016 (V2.1.1): "Experiential Networked Intelligence (ENI); Functional Concepts for Modular System Operation".

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

[i.7] Gamma, E., Helm, R. Johnson, R., Vlissides, J.: "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Nov, 1994. ISBN 978-0201633610.

[i.8] 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, Pages 218-228.

[i.9] ETSI TR 102 748 (V1.1.1): "Electromagnetic compatibility and Radio spectrum Matters (ERM); Impact of the trend towards flexibility in spectrum usage on the principles for drafting Harmonized Standards and the ETSI work programme for Harmonized Standards".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms given in ETSI GS ENI 005 [3] and the following apply:

behaviour: way in which a set of objects function

NOTE: This includes how the object reacts in a particular situation given one or more events.

camelCase: naming convention in which the first letter of each word in a compound word is capitalized, except for the first word

NOTE: This is also called lowerCamelCase.

formal: study of (typically linguistic) meaning of an object by constructing formal mathematical models of that object and its attributes and relationships:

- **formal grammar:** set of structural rules that define how to form valid strings from a language's alphabet that obey the syntax of the language
- **formal semantics:** set of tools that define grammatical meaning in a language

identity: the set of data and information that allow an object to be disambiguated from all other objects in a system, including objects of the same type:

- **digital identity:** set of data and information used by a computer system to represent an actor, such as a person, device, or application
- **contextual identity:** digital identity of an object for a particular context

inheritance: defining the characteristics and behaviour of an entity on another entity

NOTE 1: The following definition of inheritance is from ETSI TR 102 748 [i.9]: "typical property related to the concept of classes: when building a new derived class it will inherit properties from one or more previously-defined base classes, while possibly allowing for redefining or adding new properties". The ENI Information Model does not allow redefinition of properties, as this breaks information encapsulation. See ETSI GR ENI 016 [i.5], clause 4.1.2.

- **class-based inheritance:** defining the characteristics and behaviour of a subclass on a superclass

NOTE 2: Class-based inheritance defines a subclass of a superclass as a class that keeps all of the characteristics and behaviour of its superclass, and refines that definition in one or more of the following ways:

1) the subclass changes the class from abstract to concrete;

2) the subclass adds one or more attributes, operations, constraints, and/or relationships to its definition.

- **multiple inheritance:** object or class may inherit characteristics and/or behaviour from more than one superclass

NOTE 3: This style of inheritance is not used in ENI.

- **single inheritance:** object or class shall inherit characteristics and/or behaviour from only one superclass

model element: classes, attributes, operations, constraints, relationships and stereotypes used in constructing a model

override: changing of an attribute or operation implementation by a subclass that is already provided by its superclasses

NOTE: The ENI information model does not use overriding, because this alters the semantics of the class.

PascalCase: naming convention in which the first letter of all words in a compound word is capitalized

NOTE: This is also called UpperCamelCase.

refine: addition of attributes, operations, constraints, and/or relationships to an inherited class

NOTE: Refinement is only added to the semantics of the class. Refinement is not used to delete or override attributes, operations, constraints, and/or relationships that it inherits.

semantics: study of the meaning of something (e.g. a sentence or a relationship in a model)

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
DSL	Domain Specific Language
ECA	Event-Condition-Action
ID	IDentification
IP	Internet Protocol
LDAP	Lightweight Directory Access Protocol
MAC	Media Access Control
MBM	MEF Business Model
MCM	MEF Core Model
MEF	MEF (a standards body)
MIME	Multipurpose Internet Mail Extension
MPM	MEF Policy Model

NFV	Network Functions Virtualisation
OCL	Object Constraint Language
ONF	Open Networking Foundation
PDO	Policy Driven Orchestration
POC	Proof Of Concept
RDBMS	Relational Database Management System
RMM	Remote Monitoring and Management
SDO	Standards Defining Organisation
SNMP	Simple Network Management Protocols
SQL	Structured Query Language
TMF	TeleManagement Forum
TTL	Time-To-Live
UML	Unified Modeling Language
YANG	Yet Another Next Generation

4 Representing, Inferring, and Proving Knowledge in ENI

4.1 Introduction

A model is a representation of the entities of a system, including their relationships and dependencies, using an established set of rules and concepts. It is a vital part of a larger knowledge management infrastructure, which includes a set of interoperable hardware, software, and other artifacts, along with a set of standardized procedures to use them for various tasks. Models include information and data models, as well as ontologies, as described in the subclauses below. Artifacts include the representation and storage of data, information, knowledge, and wisdom (see ETSI GR ENI 016 [i.5], clause 4.4), as well as inferences made by the ENI System (see ETSI GR ENI 016 [i.5], clause 4.5 and ETSI GS ENI 005 [3], clauses 6.3.4, 6.3.6 and 6.3.7).

4.2 Definitions

4.2.1 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. The purpose of an information model is to represent *facts* in a *technology-neutral manner*.

4.2.2 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. The purpose of a data model is to represent *facts* in a *technology-specific manner*.

4.2.3 Ontology

An ontology is a language, consisting of a vocabulary and a set of primitives, that enable the semantic characteristics of a domain to be modelled. The purpose of an ontology is to represent *facts and meaning* in a *technology-neutral manner*.

4.3 Information Model Usage in ENI

4.3.1 Purpose

The purpose of an Information Model is to define and manage objects and their relationships at a conceptual level, independent of any specific implementations or protocols used to transport the data. ETSI GR ENI 016 [i.5] provides a set of important principles for designing modular systems.

Application interoperability requires a set of consensual high level patterns for the creation, manipulation, and use of descriptive, machine understandable information. In particular, it is not enough to define data with a simple MIME type (or its equivalent), as this *does not provide needed semantics or metadata.* The ENI information model defines a rich metadata hierarchy (see clause 5.2 of the present document) and relationships that enable any class to optionally have metadata attached to it. Examples include both descriptive (e.g. best current practices) and prescriptive (e.g. minimum version of an object, or valid time interval of a policy) information [4].

An Information Model provides a standardized framework for organizing data. Its structure encourages reusability and facilitates searching.

NOTE 1: Metadata may also provide advanced functionality by defining additional metadata information to search on.

MEF 78.1 [4] and ETSI GR ENI 016 [i.5] provide an overview of the MEF Core Model (MCM) and a set of design principles that use modelling, respectively. Key requirements in this approach follow (see clause 7.1 for more detailed information):

- **Inheritance:** The ENI information model shall use single inheritance. This simplifies implementation, as not all systems are able to implement multiple inheritance (MEF 78.1 [4]).
- **Overriding attributes:** The ENI information model shall not override an inherited attribute or operation. This is because the semantics of the object containing the overridden attribute or operation are now changed (MEF 78.1 [4]).
- **Information Hiding:** Information hiding prevents implementation details from being unnecessarily exposed. This makes the design more robust, since implementations details are hidden behind a stable interface that does not change. It also supports modular decomposition of a system into smaller, reusable components. The ENI information model shall use information hiding to ensure that clients of an object do not have to change when the implementation of an object changes. An important corollary is that different model elements can be developed independently and do not have to know how other software entities work (ETSI GR ENI 016 [i.5]).
- **Encapsulation:** Encapsulation is an implementation mechanism that defines the boundaries of a model element (e.g. a class is a collection of attributes, operations, and relationships that are part of a single object) (ETSI GR ENI 016 [i.5]).
- **Single Responsibility:** Classes should be designed having one responsibility; this means that the only reason for a class to change is if its responsibility changes (ETSI GR ENI 016 [i.5]).
- **Liskov Substitution:** Subclasses should be able to be substituted for superclasses without affecting the behaviour of the system (ETSI GR ENI 016 [i.5]).
- **Loose Coupling:** Each element of a loosely coupled system should depend on as little knowledge as possible of other elements. This means that classes should not depend on other classes. Rather, different types of associations should be used to realize those dependencies (ETSI GR ENI 016 [i.5]).

NOTE 2: Many models use class attributes to contain values of one or more attributes of one or more other classes.

EXAMPLE: A Person class may contain one or more phone number attributes. This violates loose coupling. A better design is to define a relationship between the Person class and a ContactInfo class. This way, if one or more phone numbers change, the Person class is not affected.

- **Design by Contract:** Software entities whose behaviour needs to be managed should be designed using formal specifications that use pre- and post-conditions, as well as invariants, to specify the behaviour of the entity (ETSI GR ENI 016 [i.5]).

4.3.2 Use of an Information Model as a Blueprint for Entity Definitions

The use of an information model is similar to how master data management is used. The information model provides a consensual set of definitions for the model elements of a system. This in turn ensures accuracy, uniformity, and consistency of semantics for objects represented in the information model. It also provides a consistent set of identifiers that can unambiguously identify different instances of the same object.

4.3.3 Use of an Information Model to Define a Lexicon and Grammar

An information model defines the meaning of concepts of interest to the managed environment in the form of classes. Classes may have zero or more attributes, operations, and relationships [i.5]. When instantiated, classes become objects.

In linguistics a lexicon contains syntactic and semantic information about words used in a particular context. The lexicon serves as a consensual reference for different users with varying skills and terminologies. It is thus analogous to a *lingua franca*, serving to enable communication between diverse groups of people who do not have a common set of vocabularies that can describe the salient features and behaviour of objects in the managed environment.

In an ENI System, the ENI Information Model helps enforce a lexicon that is readable for humans and machines. It provides definitions of classes and relationships between classes, and through associated metadata (see clause 5.2.2.6), provides additional descriptive and/or prescriptive information associated with those objects.

4.4 Data Model Usage in ENI

4.4.1 Purpose

The purpose of a Data Model is to define and manage objects and their relationships at a conceptual level. Data models are dependent of platform, language and/or protocol. Since a data model is a specific technology-dependent instantiation of a common information model, data models enable application-specific needs to be represented in an optimal form. In this way, a common concept can be translated into multiple application-specific forms. For example, a customer could be represented as a set of relative distinguished names in a directory and as a set of tables in an RDBMS. Since both data models were derived from the same information model, common characteristics and behaviour is embedded in both. This ensures that both applications have understand what the customer is.

Data modeling tools and techniques translate complex system designs into clearly understood representation of data flows and processes.

4.4.2 Use of a Data Model as a Blueprint for System Data

Data modeling creates a UML definition of all or part of a system. This includes graphical representations of the design as well as textual definitions of the objects. It may also include different types of UML diagrams that show different aspects of the interaction between objects. In addition, code may be generated from the UML model in most UML tools. Data models are living documents that evolve along with changing business needs.

Information models are built around business needs. Data models instantiate those models into artefacts (e.g. data structures and code) particular to the needs of an application. Data modeling employs standardised schemas and formal techniques that are specific to a particular platform, language, and/or protocol. This provides a common and consistent mechanism of defining and managing data resources across an organization, or even beyond.

4.4.3 Derivation of Data Models from an Information Model

ENI specifies the use of a single information model, along with zero or more data models, as shown in Figure 4-1.

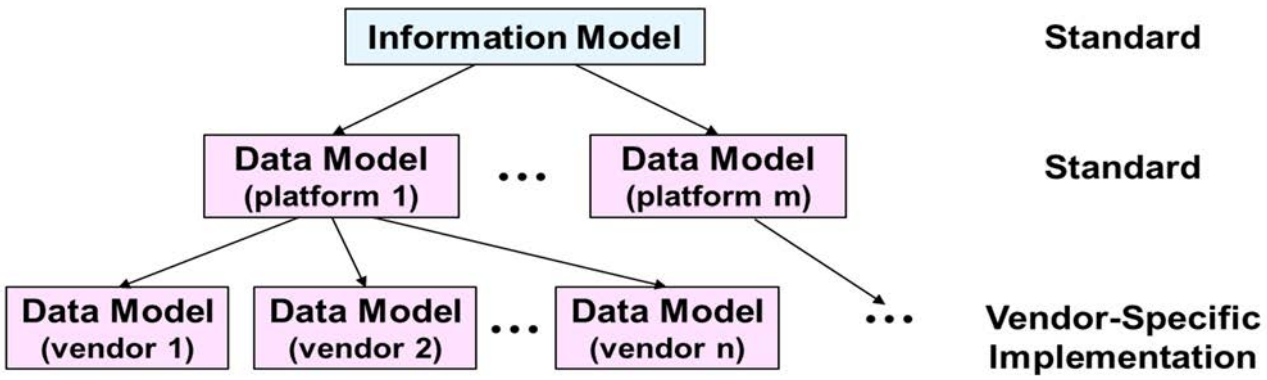

Figure 4-1: Modelling Tiers

In Figure 4-1, the information model serves as the definitive source for specifying model elements in a technologically neutral fashion. This enables different data models (e.g. a directory and a relational database) to be derived from the information model without introducing bias in the mapping.

EXAMPLE: A relational database uses a much more powerful language (SQL) than a directory, and has more powerful relationships (e.g. SQL enables very complex relationships to be defined that have no equivalent in directories).

If a data model was used for defining concepts, each concept would have an associated bias from the dependencies of that data model on data repository, data definition language, query language, implementation language, and protocol. In contrast, using an information model enables each concept to be defined without such biases.

Accordingly, Figure 4-1 illustrates one form of model mapping for possible use in ENI. In this Figure, a particular Data Model (for example, a Directory) is produced from an information model. This mapping produces a directory implementation that is conformant with the appropriate standards (e.g. LDAP or X.500). This is different than another data model, which for example is for a Relational Database, and is conformant with SQL92. The second tier of mapping accounts for the fact that different vendors provide varying degrees of compliance with a standard. In addition, some vendors provide features that are not yet standardized. This second tier of mapping enables these differences to be normalized, so that different implementations of the same data model can better interoperate. Thus, our hierarchy shows an information model standard being mapped to the appropriate data model standard, from which various vendor-specific implementations are built.

The reason for having a single information model is best understood via an analogy. Imagine that a new word was introduced in a document for which there were three dictionaries. This word had three different definitions, one in each dictionary. The reader is now confused, because the reader is not sure which definition is correct.

The above simple analogy is exacerbated if there are multiple information models, which contain large number of objects, attributes, operations, and relationships to other objects. More importantly, consider Figure 4-2, which shows two information models that have concepts that appear to be similar.

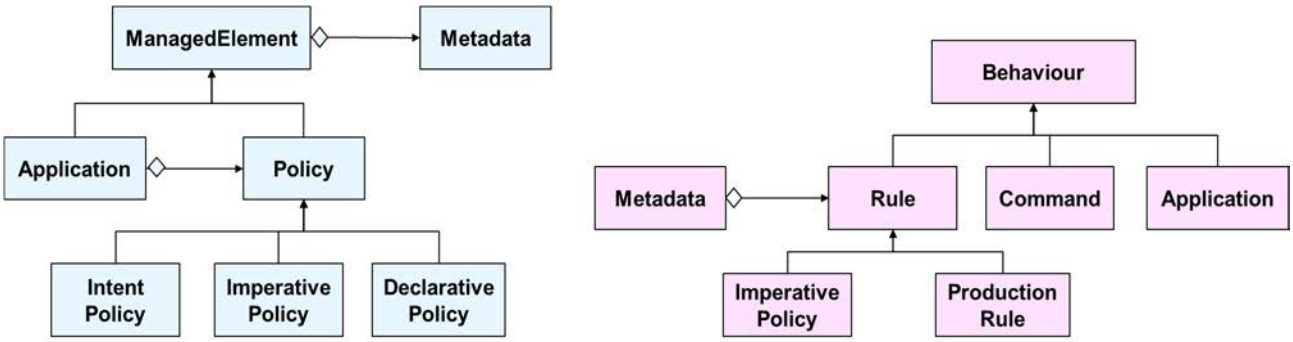

Figure 4-2: High-Level Functional Architecture of ENI When an API Broker is Not Used

The information model on the left is significantly different than the one on the right of this figure. For example, in the left model, all of the subclasses of ManagedElement inherit an aggregation to Metadata. In contrast, the right model only enables the subclasses of Rule to inherit a similar aggregation. Furthermore, even though there are two instances of an entity called IntentPolicy, this inheritance means that the two are semantically dissimilar. In addition, the IntentPolicy entity on the left cannot be "copied" to the model on the right, because there is no common root class between the two models. Even if there was, the IntentPolicy class has two significant differences: inheritance and aggregations. Even if Policy was introduced as a subclass of Rule (to enable the introduction of IntentPolicy in the model on the right), it is still lacking two features from the model on the left:

1) applications can aggregate Policy; and

2) applications also have Metadata.

This does not cover all of the differences (e.g. the left model enables Applications to aggregate different types of Policies, while the right model does not offer this capability); this capability would also need to be copied to the model on the right.

4.5 Ontology Usage in ENI

4.5.1 Introduction

Information and data models are widely used to represent managed objects. As such, they are natural to use for building APIs, since the models describe the objects that are being communicated and manipulated by the API.

However, information and data models do not have standardized *formal semantics*. For the purposes of the present document, all ontologies shall have a formal grammar and associated formal semantics.

In an ENI System, models represent facts, and ontologies add meaning to those facts. This forms a semantic network, as described in clause 6.3.4.4.1 of ETSI GS ENI 005 [3]. This combination of models and ontologies enable semantic tools and applications to be built that may infer knowledge and gain insight for operational purposes.

In an ENI System, models should be constructed using the principles set forth in ETSI GR ENI 016 [i.5]. This significantly simplifies the creation of semantic links between a set of model elements and a set of ontological concepts.

4.5.2 Use of Ontologies to Enable Formal Reasoning and Learning

An ontology is a consensual formal description of knowledge within a domain. This knowledge consists of both objects and the relationships that hold between them. It therefore ensures a common understanding of information. Relationships between concepts in an ontology enable automated reasoning. More importantly, ontologies are able to add new and change existing knowledge about a domain dynamically.

In an ENI System, ontologies should be used to specify common representations of knowledge from external and internal systems. The reason an ENI System will use both models and ontologies is twofold. First, data models of managed objects relevant to an ENI System are readily available. Since ontologies are available in a significantly limited number and for a smaller set of domains, models are realized first and form the "backbone" of knowledge management in an ENI System. Second, given the limitations of models discussed in clause 4.5.1 of the present document and in ETSI GS ENI 005 [3] (but especially in clause 6.3.4 of ETSI GS ENI 005 [3]), ontologies are needed to add and formalize knowledge. This shifts the focus from managing and using *data* to managing and using *knowledge*. This enables interoperability and enhanced search (e.g. based on the meaning of what is being searched for, not just names and attributes).

Typical use cases for an ENI System include:

- Augment and enhance model elements with meaning.
- Discover new concepts through inference.
- Discover new relationships between concepts through inference.
- Ensure that knowledge in an ENI System is always up-to-date by learning experientially.

4.6 Model Augmentation

4.6.1 Introduction

The information model is the "ultimate source of truth". However, ENI is an experiential system, and as such, is able to discover new data, information, and knowledge, as well as changes in existing data, information, and knowledge. This is enabled by noting such changes in the set of data models that are deployed, and sending these changes to the Knowledge Management Functional Block. This is captured in clause 5.6 of ETSI GS ENI 005 [3] and discussed more in clause 6.3.4 of ETSI GS ENI 005 [3].

It is therefore recommended that the information model, and all derived data models, be each augmented with ontological information. The above reasons result in many of the requirements in clause 5.6 of ETSI GS ENI 005 [3].

4.6.2 Augmentation of an Information Model using Ontologies

Whilst information models are indispensable, they do have some key technical limitations. Three such limitations that are critical to ENI are:

1) Information models do not provide the ability to robustly represent semantics.

2) Information models do not provide the ability to conduct semantic searches.

3) Information models do not provide the ability to formally prove hypotheses.

Information models work by defining facts using model elements (e.g. classes, attributes, operations, relationships, and constraints). This means that information models can not inherently *infer* new facts. For example, if an information model defines the concepts of father and brother, it is unable to infer that an uncle is the brother of a father (this example ignores more complex definitions, which suffer from the same inability as that identified). In contrast, ontologies use formal logic, and can therefore infer this relationship without having to predefine a new object called uncle. This enables ontological systems to *dynamically add and change information*.

A semantic search is one in which the subject of the search is based on the meaning, behaviour, or processes involved concerning the subject (as opposed to being limited to just keywords or phrases). For example, it is very difficult for an information model to find telemetry data that depends on a particular measurement operation, or to compare telemetry data that use similar measurement operations.

The Unified Modeling Language (UML) standard lacks the ability to use formal logic as part of its definitions of model elements. This means that facts cannot be proven. For example, a relationship is created between two objects because that relationship is asserted as a fact. Formal logic is able to mathematically prove whether in fact such a relationship exists, and more importantly, where its ideal location is (e.g. between one or more superclasses or subclasses of the objects in question).

Therefore, semantic relationships are used to attach meaning (from the ontologies) to technology-neutral facts (from the information model). This is discussed more in clause 6.3.4 of [i.1].

4.6.3 Augmentation of Data Models Using Ontologies

The reasoning in the clause 4.3.2 also applies to data models, since data models are built from information models. More importantly, the purpose of a data model is to describe the structure of the data required for a particular application or service. In contrast, an ontology represents knowledge, using formal logic, that can be applied to multiple applications and services. The main difference is that semantic relationships are now used to attach meaning (from the ontologies) to technology-specific facts (from the data model).

4.7 Synchronizing and Reconciling Modelled Data

This is for further study in Release 4 of the present document.

4.8 Securing Modelled Data

This is for further study in Release 4 of the present document.

4.9 Decision-Making

4.9.1 Introduction

This is for further study in Release 4 of the present document.

4.9.2 Control Loops

This is for further study in Release 4 of the present document.

4.9.3 Traditional Learning and Reasoning

This is for further study in Release 4 of the present document.

4.9.4 Semantic Learning and Reasoning

This is for further study in Release 4 of the present document.

4.9.5 Cognitive Learning and Reasoning

This is for further study in Release 4 of the present document.

4.10 Model-Driven DSLs

4.10.1 Introduction

This is for further study in Release 4 of the present document.

4.10.2 Constructing Model-Driven DSLs

This is for further study in Release 4 of the present document.

4.11 Model-Driven APIs

A preliminary example of how models were used to build ENI APIs is described in clause 8 of ETSI GS ENI 005 [3].

NOTE: This clause will be updated in the next version of the present document.

5 ENI Information Model

5.1 Introduction

The present document shall use Unified Modeling Language (UML) [6] to describe the salient characteristics and behaviour of objects of interest to the managed environment, and how these objects may interact with each other. Interaction may be a feature of objects (e.g. dependencies between objects) or externally defined (e.g. ENI Policies are used to change the state of a managed object).

5.2 The Design of the ENI Extended Core Model

5.2.1 Introduction

The design of the MCM is based on a single-rooted model. Many models do not use a single root hierarchy. A single root enables all subclasses to have a common interface, and be defined as the same type of object. This also means that all objects in a type hierarchy have a set of guaranteed functionality.

5.2.2 The MCM (MEF Core Model)

5.2.2.1 Introduction

The MCM is an object-orientated information model that defines objects and their relationships that represent entities and concepts important to a managed environment. It uses a set of naming rules, defined in clause 5.2.2.2, to guarantee naming consistency in model elements. This also facilitates creating namespaces to provide the developer additional flexibility in constructing applications.

The MCM has been constructed using classification theory [i.6]. This defines class hierarchies, where each class hierarchy is a family of types. The type of an object determines both a set of legal values and an interface with its environment (through calls on its operations). In each hierarchy, higher level supertypes define a set of behaviours that all of their subtypes inherit. All subtypes preserve the behaviour of the supertype operations and the attributes of its supertype.

For further information on the MCM, see [4]. The rest of clause 5.2.2 and its subclauses summarise the salient information from [4] that is used to build the ENI Extended Core Model.

5.2.2.2 Naming Rules

The following lists the MCM naming rules, adapted to ETSI terminology:

1) The MCM uses the following rules to define the names of its model elements.

2) Class names **shall** be in UpperCamelCase (i.e. the first letter is capitalized). Class names **shall not** begin with any non-alphabetic character, and no spaces are allowed.

3) Attribute names **shall** be in lowerCamelCase (i.e. the first letter is lower case); attribute names **shall not** begin with any non-alphabetic character except for the underscore, and no spaces are allowed.

NOTE: Attribute names that begin with an underscore are private attributes that reference an end of an association.

4) Relationship names **shall** be in UpperCamelCase (i.e. the first letter is capitalized). Relationship names **shall not** begin with any non-alphabetic character, and no spaces are allowed.

5) Each class **shall** 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.

6) Each attribute **shall** be prefixed with "mcm".

EXAMPLE 1: 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).

7) Each relationship **shall** be prefixed with "MCM".

EXAMPLE 2: The aggregation "EntityHasMetaData" is named "MCMEntityHasMCMMetaData".

8) All association classes **shall** be suffixed with the word "Detail".

EXAMPLE 3: 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:

1) 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) **shall** be prefixed with "MCMMEF". For example, the concept of a Descriptor from ETSI NFV is named "MCMMEFDescriptor".

2) All classes that model a concept from another SDO *exactly as it is defined* in that SDO **shall** 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:

1) 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.

2) Aggregation and composition relationships **should not** have a 0..* - 0…*.

3) 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.

4) 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 shall exist, and it makes no sense to have one side of a relationship exist while the other side does not.

5.2.2.3 MCM Superstructure

5.2.2.3.1 Overview

Figure 5-1 shows the MCM Superstructure. This consists of MCMRootEntity, which is the root of the entire MCM, and its three subclasses.

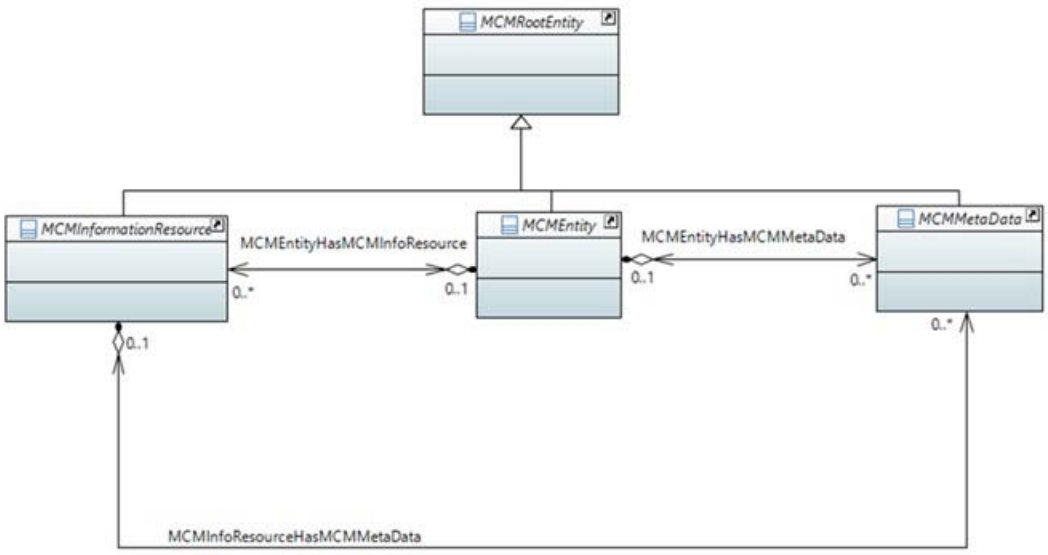

Figure 5-1: The MCM Superstructure

5.2.2.3.2 MCMRootEntity

5.2.2.3.2.1 Overview

MCMRootEntity defines a set of attributes that enable all objects to be unambiguously named, described, and identified in a managed environment. It defines three subclasses, which provide three class hierarchies to represent different objects and concepts important to a managed environment. 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.

5.2.2.3.2.2 Class Definition

MCMRootEntity is an abstract class that defines the top of the MEF Core Model (MCM) class hierarchy It specifies a set of attributes, methods, and relationships that other classes in the MCM inherit.

5.2.2.3.2.3 Attribute Definition

Table 5-1 defines the attributes of the MCMRootEntity class.

Table 5-1: Attributes of the MCMRootEntity Class

Attribute Name	Description
mcmCommonName : String[0..1]	This is a string attribute that represents a user-friendly name for this object. This attribute **shall not** be used as a naming attribute.
mcmDescription: String[0..1]	This is a string attribute that defines a free-form description of the object.
mcmObjectIDContent: String[1..1]	This attribute is a string that, with the mcmObjectIDFormat attribute, define a composite objectID. This class attribute contains the value of the objectID. The objectID enables the developer to customise the content and format to represent a unique ID of an object. The value of this attribute **shall not** be a NULL or EMPTY string.
mcmObjectIDFormat: String[1..1]	This attribute is a string that, with the mcmObjectIDFormat attribute, define a composite objectID. This class attribute defines the format of the objectID. The objectID enables the developer to customise the content and format to represent a unique ID of an object. The value of this attribute **shall not** be a NULL or EMPTY string.

5.2.2.3.2.4 Operation Definition

Table 5-2 defines the operations for the MCMRootEntity class. there are no individual getters and setters for the mcmObjectIDContent and mcmObjectIDFormat attributes because they form a tuple.

Table 5-2: Operations of the MCMRootEntity Class

Operation Name	Description
getMCMCommonName() : String[1..1]	This operation returns this object's mcmCommonName attribute as a String. If this attributes does not have a value, then this 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. The mcmCommonName String parameter attribute contains a new value for this attribute. An empty string **should** be used to define an empty value for this attribute.
getMCMObjectID() : String[2..2]	This operation returns this object's mcmObjectID attribute as a String of multiplicity [2], where the first element contains the mcmObjectIDContent attribute, and the second element contains the mcmObjectIDFormat attribute. If either returned parameter is NULL or an empty string, then an exception **shall** be thrown.

Operation Name	Description
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 defines the new value of the mcmObjectIDContent attribute. The second input parameter defines the new value of the mcmObjectIDFormat attribute. Both parameters **shall** not be NULL or EMPTY strings.
getMCMDescription() : String[1..1]	This operation returns this object's mcmDescription attribute as a String. If this attributes does not have a value, then this operation **should** return an empty string.
setMCMDescription(in inputString : String[1..1])	This operation sets the current value of the mcmDescription attribute. Its String parameter contains the new value of the mcmDescription attribute. An empty string **should** be used to define an empty value for this attribute.

5.2.2.3.2.5 Relationship Definition

No relationships (i.e. associations, aggregations, or compositions) are defined that involve RootEntity. This is because any such relationships would apply to all MCM classes, which would in turn violate software architecture principles.

5.2.2.4 MCMEntity Hierarchy

5.2.2.4.1 Overview

The MCMEntity hierarchy defines the five major different types of entities that are of interest to the managed environment. An MCMEntity defines the externally visible characteristics and behaviour of the managed environment in more detail. Each MCMEntity has a separate and distinct existence (i.e. an MCMEntity is not just a collection of attributes or an abstraction of behaviour). The MCMEntity class has five abstract subclasses, as shown in Figure 5-2.

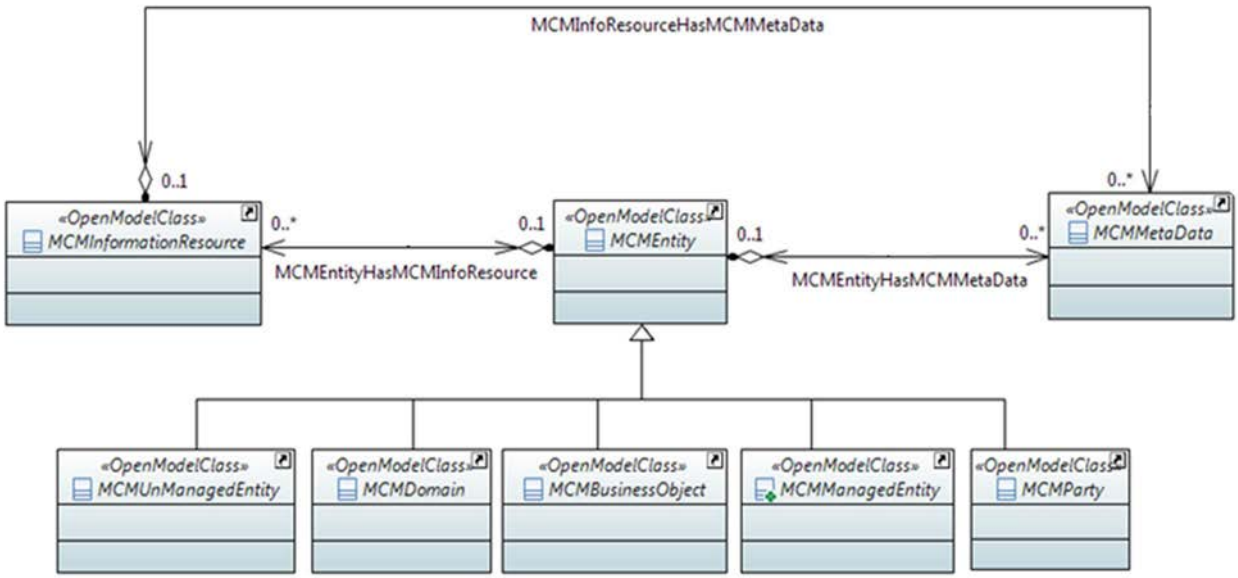

Figure 5-2: The MCMEntity Hierarchy

5.2.2.4.2 MCMEntity

5.2.2.4.2.1 Overview

Any object that is monitored or configured is typically a subclass of MCMEntity. An MCMEntity inherits the attributes, operations, and relationships of MCMRootEntity. This means that any subclass of MCMEntity can aggregate zero or more MCMMetaData objects (see clause 5.2.2.5) and zero or more MCMInformationResource objects (see clause 5.2.2.6).

5.2.2.4.2.2 Class Definition

An MCMEntity is an abstract class that represents objects that are important to the managed environment. Each of its five subclasses form a unique class hierarchy that 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).

5.2.2.4.2.3 Attribute Definition

This class does not currently define any attributes, because there is no attribute that is common to all of its subclasses.

5.2.2.4.2.4 Operation Definition

Table 5-3 defines the operations for the MCMEntity class.

Table 5-3: Operations of the MCMEntity Class

Operation Name	Description
getMCMMetaDataList() : MCMMetaData[1..*]	This operation returns the set of MCMMetaData objects that are attached to this particular MCMEntity object. If this object does not have any attached MCMMetaData, then a NULL MCMMetaData object **should** be returned.
setMCMMetaDataList(in attachedMetaDataList : MCMMetaData[1..*])	This operation defines the set of MCMMetaData objects that will be attached to this particular MCMEntity object. The single input parameter, called attachedMetaDataList, 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. Any existing aggregations not specified in the input parameter are deleted. Each aggregation created by this operation **should** have an instance of the MCMEnttyHasMCMMetaDataDetail 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 creates a set of aggregations between this particular MCMEntity object and the set of MCMMetaData objects identified in the input parameter. Each aggregation created by this operation **should** have an instance of the MCMEntityHasMCMMetaDataDetail class.
delMCMMetaDataList()	This operation deletes **all** instances of attached MCMMetaData for this particular MCMEntity. This operation removes the association class and the aggregation between this MCMEntity object and each MCMMetaData object that is attached to this MCMEntity object.
delMCMMetaDataPartialList(in attachedPartialMetaDataList : MCMMetaData[1..*])	This operation deletes a set of MCMMetaData objects from this particular MCMEntity. This operation removes the association class and its aggregation between each MCMMetaData object specified in the input parameter and this MCMEntity. All other aggregations between this MCMEntity and other MCMMetaData objects that are not specified in the input parameter **shall not** be affected.
getMCMInfoResourceList() : MCMInformationResource[1..*]	This operation returns the set of MCMInformationResource objects that are currently attached to this particular MCMEntity object as an array of one or more MCMInformationResource objects. If this object does not have any attached MCMInformationResource objects, then a NULL MCMInformationResource object **should** be returned.

Operation Name	Description
setMCMInfoResourceList(in attachedInfoResourceList : MCMInfoResource[1..*])	This operation defines the set of MCMInformationResource objects that will be attached to this particular MCMEntity object. This operation deletes any existing attached MCMInformationResource objects and then instantiates a new set of MCMInformationResource objects; in doing so, each MCMInformationResource object is attached to this particular MCMEntity object by creating an instance of the MCMEntityHasMCMInfoResource aggregation and realizing that aggregation instance as an association class. Each aggregation created by this operation should have an instance of the MCMEntityHasMCMInfoResourceDetail class.
setMCMInfoResourcePartialList(in attachedInfoResource-PartialList : 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 creates a set of aggregations between this particular MCMEntity object and the set of MCMInformationResource objects identified in the input parameter. Each aggregation created by this operation **should** have an instance of the MCMEntityHasMCMInfoResourceDetail class.
delMCMInfoResourceList()	This operation deletes **all** instances of attached MCMInformationResource objects for this particular MCMEntity. This operation removes the association class and its aggregation between this MCMEntity object and each MCMInformationResource object that is attached to this MCMEntity object.
delMCMInfoResourcePartialList(in attachedPartial-MetaData : MCMInformation-Resource[1..*])	This operation deletes a set of MCMInformationResource objects from this particular MCMEntity. This operation removes the association class and its aggregation between each MCMInformationResource object specified in the input parameter and this MCMEntity. All other aggregations between this MCMEntity and other MCMInformationResource objects that are not identified in the input parameter **shall not** be affected.

5.2.2.4.2.5 Relationship Definition

MCMEntity defines two relationships, called MCMEntityHasMCMInfoResource and MCMEntityHasMCMMetaData, as shown in Figure 5-2.

MCMEntityHasMCMInfoResource is an optional aggregation that defines the set of MCMInformationResource objects that are associated with this particular set of MCMEntity objects. Its multiplicity is 0..1 - 0..*. If this aggregation is instantiated, then zero or more MCMInformationResource objects can be aggregated by this particular MCMEntity object. The cardinality on the part side is 0..*, which enables an MCMEntity object to be defined without having to define an MCMInformationResource object for it to be associated with. 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 behaviour 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 optional aggregation that defines the set of MCMMetaData objects that are associated with this particular set of MCMEntity objects. Its multiplicity is 0..1 - 0..*. If this aggregation is instantiated, then zero or more MCMMetaData objects can be aggregated by this particular MCMEntity object. The cardinality on the part side is 0..*; this enables an MCMEntity object to be defined without having to define an MCMMetaData object for it to be associated with. 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 behaviour 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 to define policy rules that constrain which part objects (i.e. MCMMetaData) are attached to which object. MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

5.2.2.4.3 MCMUnManagedEntity Hierarchy

5.2.2.4.3.1 Overview

A simplified view of the MCMUnManagedEntity hierarchy is shown in Figure 5-3.

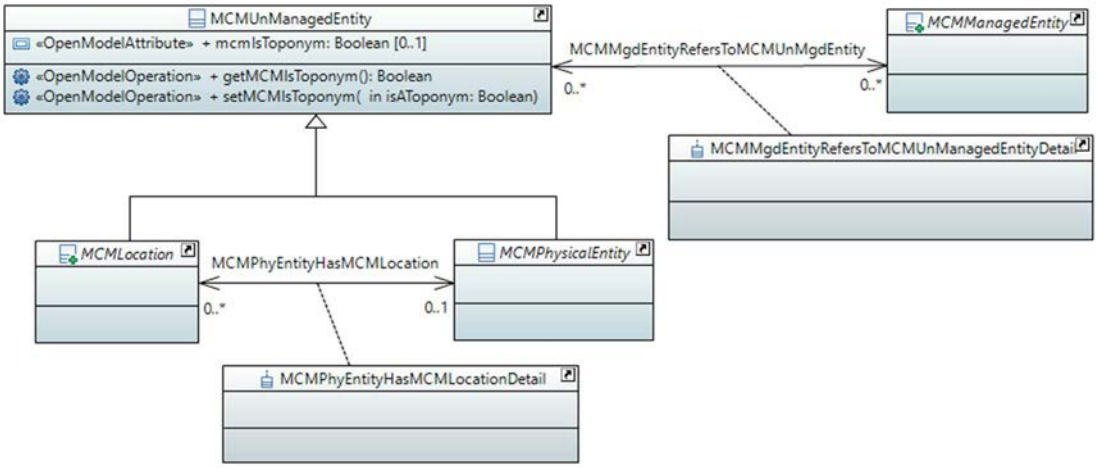

Figure 5-3: The MCMUnManagedEntity Hierarchy

The purpose of the MCMUnManagedEntity hierarchy is to model the different types of MCMEntities that cannot be intrinsically managed, yet are of interest to the managed environment. 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).

NOTE: The use of subclasses of MCMUnManagedEntity in the ENI Extended Core Model are for further study.

5.2.2.4.3.2 Class Definition

MCMUnMangedEntity 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.

Two main subclasses of MCMUnManagedEntity, called MCMLocation and MCMPhysicalEntity, are defined.

5.2.2.4.3.3 Attribute Definition

This class defines a single attribute, which is shown in Table 5-4.

Table 5-4: Attributes of the MCMUnManagedEntity Class

Attribute Name	Description
mcmIsToponym[0..1]	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 "Corporate Headquarters" and "CustomerSiteLocation".

5.2.2.4.3.4 Operation Definition

The operations of this class are shown in Table 5-5.

Table 5-5: Operations of 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).
setMCMIsToponym(in isAToponym : Boolean[1..1])	This operation sets the current value of the mcmIsToponym attribute. It contains a single input parameter, of type Boolean, to define the new value of this attribute.

5.2.2.4.3.5 Relationship Definition

No relationships are defined for the MCMUnManagedEntity class. It participates in one relationship, called MCMMgdEntityRefersToMCMUnManagedEntity, which is defined in clause 5.2.2.4.4.5.

5.2.2.4.4 MCMManagedEntity Hierarchy

5.2.2.4.4.1 Overview

The MCMManagedEntity class has six abstract subclasses, as shown in Figure 5-4.

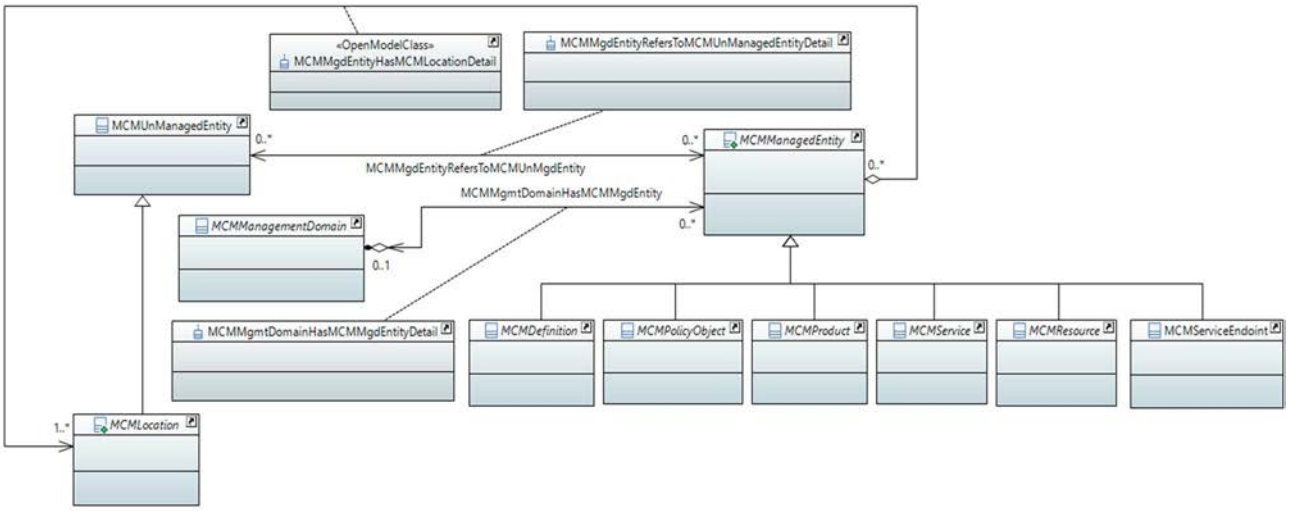

Figure 5-4: The MCMManagedEntity Hierarchy

The MCMManagedEntity hierarchy represents and manages Products, Services, and Resources. As such, the MCMDefinition hierarchy is used to specify common characteristics and behaviour of these three concepts, and the MCMPolicyObject hierarchy is used to manage these three concepts.

5.2.2.4.4.2 Class Definition

MCMManagedEntity 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.

This is the base class for defining Product, Resource, Service, and Policy hierarchies.

5.2.2.4.4.3 Attribute Definition

Table 5-6 defines the attributes of this class.

Table 5-6: The Attributes of the MCMManagedEntity Class

Attribute Name	Description
mcmAdminState : MCMAdminState[1..1]	This is a mandatory enumeration that defines the set of states for what the IETF and ITU-T call "AdminStatus". The MCM extends this concept to include locked, in test, and enabled states. The values that this attribute can have are defined by the MCMAdminState enumeration.
mcmOperState : MCMOperState[1..1]	This is a mandatory enumeration that defines the set of states for what the IETF and ITU-T call "OperStatus". The MCM version extends this concept to include operating, not operating, enabled, installed, and other states. The values that this attribute can have are defined by the MCMOperState enumeration.
mcmMgdEntityCreationDate : TimeAndDate[1..1]	This is a TimeAndDate attribute. It defines the time and date that this object instance was created. This attribute **should** have a valid time and/or date. The implementation **may** ensure that the fields in this data type are set to an appropriate default value.
mcmExternalID-AttrName : String[0..1]	This attribute is a string, and defines the name of an objectID that an external Application is using. This attribute **shall not** be used as a naming attribute. If an object does not have a value for this class attribute, then an empty string **should** be used.

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. This is the purpose of the mcmExternalIDAttrName defined in Table 5-6.

The mcmExternalIDAttrName attribute **shall** 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 in an interoperable manner.

5.2.2.4.4.4 Operation Definition

Table 5-7 defines the operations of this class. These operations include getters and setters that manipulate the behaviour provided in relationships that MCMManagedEntity participates in.

Table 5-7: Operations of the MCMManagedEntity Class

Operation Name	Description
getMCMAdminState() : MCMAdminState[1..1]	This operation returns the value of the mcmAdminState attribute. 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. The newAdminState input parameter defines the new value to be used.
getMCMOperState() : MCMOperState[1..1]	This operation returns the value of the mcmOperState attribute. 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. The newOperState input parameter contains the value to be used.
getMCMMgdEntityCreation-Date() : TimeAndDate[1..1]	This operation returns the value of the mcmMgdEntityCreationDate attribute. The value returned is a TimeAndDate attribute. This attribute **should** have a complete and valid time and/or date. The implementation **may** ensure that the fields in this data type are set to an appropriate default value.
setMCMMgdEntityCreation-Date(in newTimeAndDate : TimeAndDate [1..1])	This operation defines a new value for the mcmMgdEntityCreationDate attribute. The newTimeAndDate attribute defines the value to be used. This attribute **should** have a complete and valid time and/or date.
getMCMExternalIDAttrName() : String[1..1]	This operation retrieves the value of the mcmExternalIDAttrName attribute. This class attribute **shall not** be used as a naming attribute. 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. The newAttrName input parameter defines the new name of this attribute. This class attribute **shall not** be used as a naming attribute.

Operation Name	Description
getMCMParentDomain() : MCMManagement-Domain[1..1])	This operation retrieves the MCMManagementDomain that contains this MCMManagedEntity. If this MCMManagedEntity has no containing MCMManagementDomain, then it **should** return a NULL MCMManagementDomain object.
setMCMParentDomain (in newMgmtDomain : MCMManagement-Domain[1..1])	This operation defines a new MCMManagementDomain to contain this particular MCMManagedEntity. The newMgmtDomain attribute defines the new parent. If this MCMManagedEntity object already has a parent, then its existing parent 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. This MCMManagementDomain object **shall not** have more than one parent.
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.
getReferredMCMUnManaged-EntityList() : MCMUnManagedEntity[1..*]	This operation retrieves the set of MCMUnManagedEntity objects that refer to this MCMManagedEntity object. If this MCMManagedEntity object does not refer to an MCMUnManagedEntity object, then it **should** return a NULL MCMUnManagedEntity object.
setReferredMCMUnManaged-EntityList(in newUnMgdEntityList : MCMUnManagedEntity[1..*])	This operation defines a new set of MCMUnManagedEntity objects that refer to this particular MCMManagedEntity object. The newUnMgdEntityList input parameter 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. Every association created **should** have a new association class created to realize the semantics of that association.
setReferredMCMUnManaged-EntityPartialList(in newUnMgdEntityList : MCMUnManagedEntity[1..*])	This operation defines a new set of MCMUnManagedEntity objects that refer to this particular MCMManagedEntity object. The newUnMgdEntityList input parameter 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. Every association created **should** have a new association class created to realize the semantics of that association.
delReferredMCMUnManaged-Entity()	This operation removes the association, and its association class, for all MCMUnManagedEntity objects that this particular MCMManagedEntity refers to. This operation does **not** affect any of the MCMUnManagedEntity objects or the MCMManagedEntity object; it just deletes all associations between this MCMManagedEntity object and this MCMUnManagedEntity object.
delReferredMCMUnManaged-EntityPartial(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 object. The unMgdEntityList input parameter defines the set of MCMUnManagedEntity objects that will be unlinked from this particular MCMManagedEntity object. 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. Any association between this MCMManagedEntity object and other MCMUnManagedEntity objects that are not specified in the unMgdEntityList **shall not** be affected.

5.2.2.4.4.5 Relationship Definition

The MCMManagedEntity class defines a single optional association, called MCMMgdEntityRefersToMCMUnMgdEntity. This association enables an MCMManagedEntity to refer to a set of MCMUnManagedEntities, and vice versa. The value of this relationship is that it can link content of the MCMUnManagedEntity (e.g. location or address) to the MCMManagedEntity.

EXAMPLE 1: An MCMService could be associated with the location of an MCMPhysicalEntity at a particular MCMLocation.

The multiplicity of this relationship is 0..* - 0..*. If this association is instantiated, then zero or more MCMUnManagedEntity objects can be associated with this particular MCMManagedEntity object. 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.

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 behaviour of this association class.

EXAMPLE 2: This association class can be used to define which MCMUnManagedEntity objects are allowed to be associated with which MCMManagedEntity objects (or vice-versa).

The Policy Pattern 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). The MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

The MCMManagedEntity class also participates in an aggregation, called MCMMgmtDomainHasMCMMgdEntity. This is described in clause 5.2.2.4.11.6.1.

5.2.2.4.5 MCMDefinition Hierarchy

5.2.2.4.5.1 Overview

This is 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, capabilities, and constraints will be invariant over all instances of each concrete subclass of MCMDefinition.

The use of MCMDefinition objects is critical to enabling scalable and consistent creation of Product, Service, and Resource hierarchies that share common properties and behaviour.

5.2.2.4.5.2 Class Definition

MCMDefinition is an abstract class, and specializes MCMManagedEntity. It is a template that defines the salient characteristics, behaviour, capabilities, and constraints of concrete subclasses of an MCMManagedEntity. When concrete subclasses of MCMDefinition are instantiated, these characteristics, capabilities, and constraints will be invariant over all instances of each concrete subclass of MCMDefinition.

5.2.2.4.5.3 Attribute Definition

No attributes are defined for the MCMDefinition class. This is because MCMDefinition objects can be created for other types of MCMEntity objects (e.g. an Inventory or a ProductOrder) that are not part of the MCMManagedEntity class hierarchy. While MCMDefinition objects are manageable, they may be used to structure information that is not manageable. This makes it impossible to define attributes that apply to managed and unmanaged objects that use MCMDefinition objects.

5.2.2.4.5.4 Operation Definition

No operations are defined for the MCMDefinition class for the same reason as that in clause 5.2.2.4.5.3.

5.2.2.4.5.5 Relationship Definition

No operations are defined for the MCMDefinition class for the same reason as that in clause 5.2.2.4.5.3.

5.2.2.4.5.6 MCMDefinition Subclasses

5.2.2.4.5.6.1 MCMDefinitionDecorator

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

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 clause 5.2.2.4.5.6.3) and MCMOfferHasMCMDefinitionDecorator (see clause 5.2.2.4.5.6.4).

5.2.2.4.5.6.2 MCMBusinessTerm

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; see clause 5.2.2.4.5.6.4 for the definition of an MCMOffer.

Table 5-8 defines the attributes of the MCMBusinessTerm class.

Table 5-8: Attributes of the MCMBusinessTerm Class

Attribute Name	Description
mcmBusTermRMM : String[0..1]	This is a string attribute 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.
mcmBusTermServiceDesk : String[0..1]	This is a string attribute that defines the type of problem management and remediation services that are available to MCMCustomers that purchase this MCMOffer. It serves as a single point of contact for all end-user issues.
mcmBusTermVendorMgmt : String[0..1]	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.

Table 5-9 defines the operations of the MCMBusinessTerm class.

Table 5-9: Operations of the MCMBusinessTerm Class

Operation Name	Description
getMCMBusTermRMM() : MCMString[1..1]	This operation returns the value of the mcmBusTermRMM attribute. The value returned describes the remote monitoring and management capabilities of this MCMBusinessTerm. 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. The input parameter contains the text that describes the remote monitoring and management capabilities of this MCMBusinessTerm. The newString attribute **may** contain an empty string (e.g. for clearing this field).
getMCMBusTermServiceDesk() : String[1..1]	This operation returns the value of the mcmBusTermServiceDesk attribute. The value returned describes the problem management and remediation services of this MCMBusinessTerm. If the mcmBusTermRMM attribute is empty, then an empty string **should** be returned.

Operation Name	Description
setMCMBusTermServiceDesk (in newString : String[1..1])	This operation defines the new value for the mcmBusTermServiceDesk attribute. The input parameter contains a description of the problem management and remediation services of this MCMBusinessTerm. The newString attribute **may** contain an empty string (e.g. for clearing this field).
getMCMBusTermVendorMgmt() : String[1..1]	This operation returns the value of the mcmBusTermVendorMgmt attribute. The value returned is a String that describes the type of vendor management that is included for Buyers that purchase an MCMOffer that has this MCMBusinessTerm. If the mcmBusTermVendorMgmt attribute is empty, then an empty string **should** be returned.
setMCMBusTermVendorMgmt (in newString : String[1..1])	This operation defines a new value for the mcmBusTermVendorMgmt attribute. The input parameter contains a description of the type of vendor management that is included for Buyers that purchase an MCMOffer that has this MCMBusinessTerm. 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 currently decorate this MCMBusinessTerm object. The return value is an array of one or more MCMFeature objects. 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 operation decorates this particular MCMBusinessTerm object with the set of MCMFeature objects identified in the input parameter. This operation 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. 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. 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 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 **shall be** affected. 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. 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 by deleting their associations. It does not delete the MCMFeature objects. 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. Implementations **may** remove the decorating object any way they wish, including deleting the object.

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

5.2.2.4.5.6.3 MCMFeature

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

An MCMFeature is a salient characteristic or behaviour of an object that it describes. An MCMFeature may be related to one or more MCMCapability objects via the MCMEntityHasMCMMetaData aggregation. This enables a list of used and unused capabilities to augment the definition of each MCMFeature object.

MCMFeature is the superclass for the MCMProductFeature, MCMServiceFeature, and MCMResourceFeature classes. 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).

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 optional aggregation, called MCMFeatureDecoratesMCMDefinition, is defined for the MCMFeature class. This aggregation defines the set of MCMFeatures that wrap (or decorate) this particular MCMDefinition object. The multiplicity of this aggregation is 0..1 - 0..*. 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 of ETSI GS ENI 005 [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.

5.2.2.4.5.6.4 MCMOffer

MCMOffer 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 has 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. The MCMOfferHasMCMDefinitionDecorator aggregation is part of a pattern that enables MCMOffers to be made up of a combination of different MCMFeatures and MCMBusinessTerms. Both MCMFeature and MCMBusinessTerm 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 5-10 defines the operations of the MCMBusinessTerm class.

Table 5-10: Operations of the MCMOffer Class

Operation Name	Description
getMCMBusTermRMM() : MCMString[1..1]	This operation returns the value of the mcmBusTermRMM attribute. 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. The input parameter contains the text that describes the remote monitoring and management capabilities of this MCMBusinessTerm. The newString attribute **may** contain an empty string (e.g. for clearing this field).
getMCMBusTermServiceDesk() : String[1..1]	This operation returns the value of the mcmBusTermServiceDesk attribute. If the mcmBusTermRMM attribute is empty, then an empty string **should** be returned.

Operation Name	Description
setMCMBusTermServiceDesk (in newString : String[1..1])	This operation defines the new value for the mcmBusTermServiceDesk attribute. The input parameter contains a description of the problem management and remediation services of this MCMBusinessTerm. The newString attribute **may** contain an empty string (e.g. for clearing this field).
getMCMBusTermVendorMgmt() : String[1..1]	This operation returns the value of the mcmBusTermVendorMgmt attribute. If the mcmBusTermVendorMgmt attribute is empty, then an empty string **should** be returned.
setMCMBusTermVendorMgmt (in newString : String[1..1])	This operation defines a new value for the mcmBusTermVendorMgmt attribute. The input parameter contains a description of the type of vendor management that is included for Buyers that purchase an MCMOffer that has this MCMBusinessTerm. 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 currently decorate this MCMBusinessTerm object. 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 operation decorates this particular MCMBusinessTerm object with the set of MCMFeature objects identified in the input parameter. This operation 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. 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. 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 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 **shall** be affected. 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. 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. 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. Implementations **may** remove the decorating object any way they wish, including deleting the object.

A single optional aggregation, called MCMOfferHasMCMDefinitionDecorator, is defined for the MCMOffer class. This aggregation defines the set of MCMDefinitionDecorators that are contained by this particular MCMOffer object. This is an optional aggregation. Its multiplicity is 0..1 – 0..*. 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. 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 ETSI GS ENI 005 [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.

5.2.2.4.6 MCMPolicyObject

5.2.2.4.6.1 Overview

This MCM class is the root of the MPM (MEF Policy Model). This provides a set of abstractions for viewing any type of Policy, regardless of its programming paradigm (e.g. imperative, declarative, or intent). More importantly, this means that any MCMManagedEntity can be the target of an MCMPolicy, and that other MCM objects (e.g. MCMPartyRole, see clause 5.2.2.6) can the subject of an MCMPolicy. See clauses 5.3.2.6.7.28 and 5.3.2.6.7.27, respectively.

5.2.2.4.6.2 Class Definition

This is an abstract class that is the root of the MPM. Its purpose is to ensure that MCMPolicy objects can interact with other objects of the MCM via relationships. The MPM is discussed in clause 5.3.2.

5.2.2.4.6.3 Attribute Definition

No attributes are defined for the MCMPolicyObject class. This is because MPM models policies as consisting of a set of MPMPolicyStructure and MPMPolicyComponentStructure objects. No common attribute exists for these subclasses that is not already inherited by MCMPolicyObject.

5.2.2.4.6.4 Operation Definition

No operations are defined for the MCMPolicyObject class for the same reason as that in clause 5.2.2.4.6.3.

5.2.2.4.6.5 Relationship Definition

No relationships are defined for the MCMPolicyObject class for the same reason as that in clause 5.2.2.4.6.3.

5.2.2.4.7 MCMProduct Hierarchy

NOTE: Should this version of the ENI Extended Core Model use Product classes?

5.2.2.4.7.1 Overview

An MCMProduct defines the set of goods and services offered to a market by an MCMParty that is playing an appropriate MCMPartyRole (e.g. MCMServiceProvider). MCMProducts are purchased by an MCMCustomer, which is also a type of MCMPartyRole. Each such purchased Product is based on an MCMProductOffer, and results in a separate instance of the MCMProduct class.

The MCMProduct class hierarchy is shown in Figure 5-5.

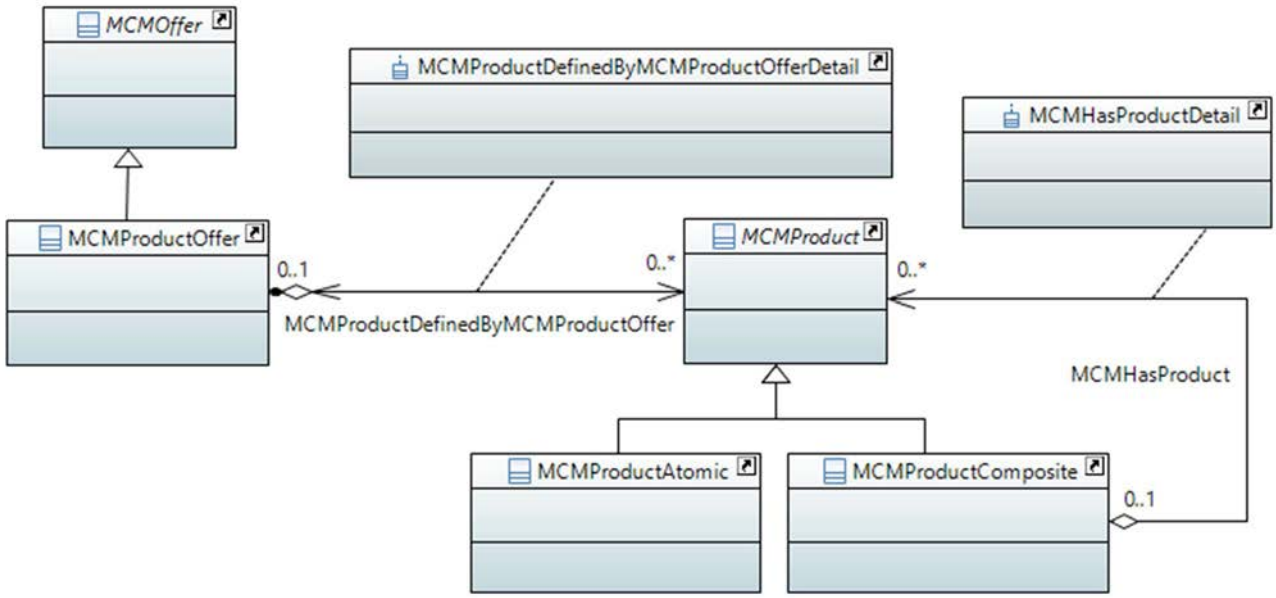

Figure 5-5: The MCMProduct Hierarchy

5.2.2.4.7.2 Class Definition

MCMProduct is an abstract class that that specializes MCMManagedEntity. An MCMProduct defines the set of goods and services, offered to a market, by a set of MCMPartyRoles. MCMProducts are purchased by an MCMCustomer, which is another 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.

An MCMProduct **may** exist in a purchased or unpurchased state.

5.2.2.4.7.3 Attribute Definition

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 downloads to 1 Gb/month, and the current usage is 750 Mb); 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 does not "know" how much usage is incurred, but can find out (e.g. by using an operation).

5.2.2.4.7.4 Operation Definition

Table 5-11 defines the operations of the MCMProduct class.

Table 5-11: Operations of the MCMProduct Class

Operation Name	Description
getMCMProductParent() : MCMProductComposite[1..1]	This operation returns the parent of this MCMProduct object. 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 input parameter is of type MCMProductComposite. If this MCMProduct object already has a parent, then an exception **should** be raised. This MCMProduct object **shall not** have more than one parent.

5.2.2.4.7.5 Relationship Definition

The MCMProduct class participates in two aggregations, which are shown in Figure 5-5.

NOTE: These will be defined if it is decided to use MCMProduct and its subclasses.

5.2.2.4.8 MCMService Hierarchy

5.2.2.4.8.1 Overview

An MCMService represents functionality that can be used by different internal and external users (e.g. a management system and a Customer, respectively). Services may be used by other Services, but not by Resources. The MCMService class hierarchy is shown in Figure 5-6.

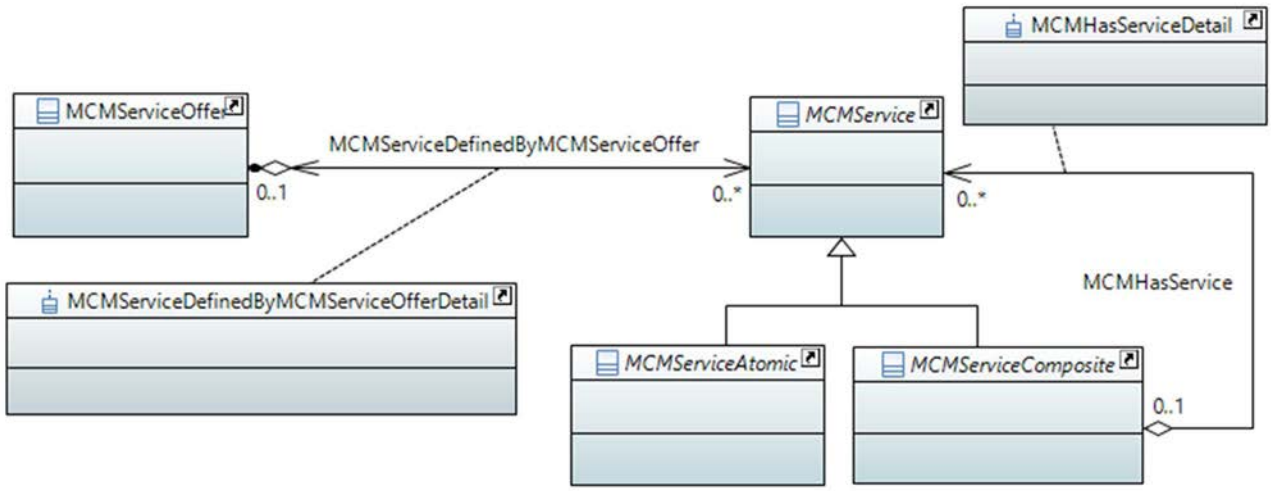

Figure 5-6: The MCMService Hierarchy

5.2.2.4.8.2 Class Definition

MCMService 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.

5.2.2.4.8.3 Attribute Definition

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 1 Gb/months, and the current usage is 750 Mb); and

2) how much time is left (e.g. the MCMService is being used on a time-limited subscription).

In the above examples, the MCMService itself does not "know" how much usage is incurred, but can find out (e.g. by using an operation). As another example, an MCMManagedEntity may need to know the status of all of the MCMServiceEndpoints and MCMServiceComponents 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.

5.2.2.4.8.4 Operation Definition

Table 5-12 defines the operations of the MCMProduct class.

Table 5-12: Operations of the MCMService Class

Operation Name	Description
getMCMServiceParent() : MCMServiceComposite[1..1]	This operation returns the parent of this MCMService object. The parent **shall** be of type MCMServiceComposite. 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 input parameter **shall** be of type MCMServiceComposite. If this MCMService object already has a parent, then an exception **should** be raised. This MCMService object **shall not** have more than one parent.

5.2.2.4.8.5 Relationship Definition

The MCMService class participates in three aggregations, as shown in Figure 5-6.

NOTE: These will be defined if it is decided to use MCMService subclasses.

5.2.2.4.8.6 MCMService Subclasses

5.2.2.4.8.6.1 MCMServiceAtomic Class Definition

MCMServiceAtomic is an abstract class that specializes MCMService. This class represents stand-alone MCMService objects that **shall 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.

5.2.2.4.8.6.2 MCMServiceComposite Class Definition

MCMServiceComposite is an abstract class, and specializes MCMService. This class represents a set of MCMServiceComposite objects that are organized into a tree structure. 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 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 5-13 defines following operations for this class.

Table 5-13: Operations of the MCMServiceComposite Class

Operation Name	Description
getMCMServiceList() : MCMService[1..*]	This operation returns the set of all MCMService objects that are contained in this specific MCMServiceComposite object. This operation returns a list of zero or more MCMService objects. 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 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. 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 creates a set of aggregations between this particular MCMServiceComposite object and each of the MCMService objects identified in the childObjectList. 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 removes both the association class and the aggregation between this MCMServiceComposite object and each MCMService object that is contained in this MCMServiceComposite object.
delMCMServicePartialList (in childObjectList : MCMService[1..*])	This operation deletes a set of MCMService objects from this particular MCMServiceComposite object. This has the effect of removing both the association and the aggregation between each MCMService object specified in the input parameter and this MCMServiceComposite object. All other aggregations between this MCMServiceComposite and other MCMService objects that are not identified in the input parameter **shall not** be affected.

The MCMServiceComposite class defines a single optional aggregation, called MCMHasService. This aggregation is used to define the set of MCMServices that are contained within this particular MCMServiceComposite object. Its multiplicity is defined to be 0..1 - 0..*. 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 of ETSI GS ENI 005 [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.

5.2.2.4.9 MCMResource Hierarchy

5.2.2.4.9.1 Overview

An 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, and may be logical or virtual in nature. 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.

A simplified view of part of the MCMResource hierarchy is shown in Figure 5-7.

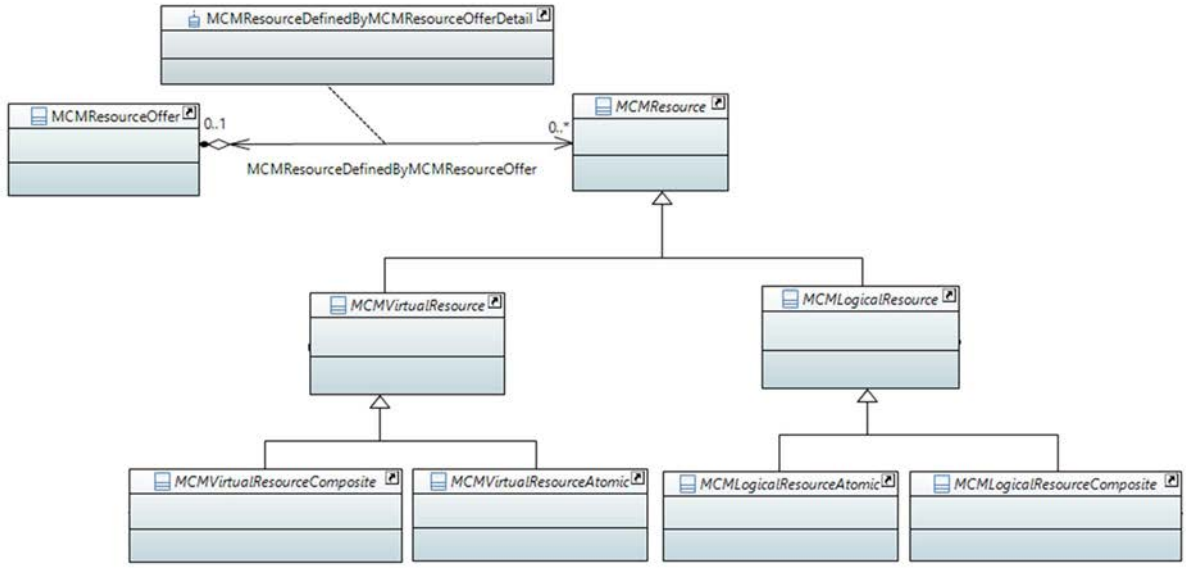

Figure 5-7: The MCMResource Hierarchy

5.2.2.4.9.2 Class Definition

MCMResource is an abstract class that specializes MCMManagedEntity. It provides capabilities that may be consumed by other MCMResources and/or MCMServices. In addition, an MCMResource, which may be logical or virtual in nature, may consume other MCMResources. An MCMResource has a distinct state. MCMResources are typically limited in quantity and/or availability. 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 clause 5.2.2.4.3).

5.2.2.4.9.3 Attribute Definition

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

5.2.2.4.9.4 Operation Definition

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

5.2.2.4.9.5 Relationship Definition

The MCMResource class participates in a single aggregation, called MCMResourceDefinedByMCMResourceOffer, as shown in Figure 5-7.

NOTE: This will be defined if it is decided to use MCMOffer subclasses.

5.2.2.4.9.6 MCMResource Subclasses

5.2.2.4.9.6.1 MCMVirtualResource Class Definition

MCMVirtualResource is an abstract class that 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 behaviour 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 5-14 defines following operations for this class.

Table 5-14: Operations of the MCMVirtualResource Class

Operation Name	Description
getMCMVirtualResourceParent() : MCMVirtualResource[1..1]	This operation returns the parent of this MCMVirtualResource object. 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. If this MCMVirtualResource object already has a parent, then an exception **should** be raised. This MCMVirtualResource object **shall not** have more than one parent.

The MCMVirtualResource class participates in one aggregation, called MCMHasVirtualResource; see clause 5.2.2.4.9.6.3.

5.2.2.4.9.6.2 MCMVirtualResourceAtomic Class Definition

MCMVirtualResourceAtomic is an abstract class that specializes MCMVirtualResource. It represents an MCMResource that is modeled as a single, stand-alone, manageable entity that is virtualized (i.e. it is not directly associated with the underlying physical hardware).

This object **shall 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.

5.2.2.4.9.6.3 MCMVirtualResourceComposite Class Definition

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

An MCMVirtualResourceComposite object **may** contain zero or more MCMVirtualResourceAtomic and/or zero or more MCMVirtualResourceComposite objects.

At this time, no attributes are defined for the MCMVirtualResourceComposite class. Most attributes will likely be realized using relationships and/or operations. For example, to find the set of MCMVirtualResources contained in an MCMVirtualResourceComposite object uses class operations to query each of its contained MCMVirtualResources (which will in turn call their operations to acquire their MCMVirtualResources), aggregate and organize the information, and provide that information in its operation response.

Table 5-15 defines following operations for this class.

Table 5-15: Operations of the MCMVirtualResourceComposite Class

Operation Name	Description
getMCMVirtualResourceList() : MCMVirtualResource[1..*]	This operation returns the set of all MCMVirtualResource objects that are contained in this specific MCMVirtualResourceComposite object. This operation returns a list of zero or more MCMVirtualResource objects. If this MCMVirtualResourceComposite object has no children, then it **should** return a NULL MCMVirtualResource object.

Operation Name	Description
setMCMVirtualResourceList (in childObjectList : MCMVirtualResource [1..*])	This operation defines a set of MCMVirtualResource objects that are contained by this particular MCMVirtualResourceComposite object. This operation deletes any existing contained MCMVirtualResource objects, 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. 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 creates a set of aggregations between this particular MCMVirtualResourceComposite object and each of the MCMVirtualResource objects identified in the childObjectList. 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 removes both the association class and the aggregation between this MCMVirtualResourceComposite object and each MCMVirtualResource object that is contained in this MCMVirtualResourceComposite object.
delMCMVirtualResourcePartialList (in childObjectList : MCMVirtualResource [1..*])	This operation deletes a set of MCMVirtualResource objects from this particular MCMVirtualResourceComposite object. This removes both the association class and the aggregation between each MCMVirtualResource object specified in the input parameter and this MCMVirtualResourceComposite object. All other aggregations between this MCMVirtualResourceComposite and other MCMVirtualResource objects that are not identified in the input parameter **shall not** be affected.

The MCMVirtualResourceComposite class defines a single optional aggregation called MCMHasVirtualResource. The multiplicity of this aggregation is 0..1 - 0..*. 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. 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 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern. The MCMPolicyStructure object is an abstract class that is the superclass of imperative, declarative, and intent policy rules.

NOTE: The use of other MCMVirtualResource subclasses besides those defined in the present document is for further study.

5.2.2.4.9.6.4 MCMLogicalResource Class Definition

MCMLogicalResource is an abstract class that specializes MCMResource. It represents MCMResources that have inherent digital communication and management capabilities and are neither physical nor virtual in nature. Examples include operating systems, application and management software, protocols, and the logic required to perform forwarding, routing, and other functions. It is shown in Figure 5-7.

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

The operations for this class are shown in Table 5-16.

Table 5-16: Operations of the MCMLogicalResource Class

Operation Name	Description
getMCMLogicalResourceParent() : MCMLogicalResource[1..1]	This operation returns the parent of this MCMLogicalResource object. If this MCMLogicalResource object has no parent, then a NULL MCMLogicalResource object **should** be returned.
setMCMLogicalResourceParent (in newParent) : MCMLogicalResourceComposite[1..1]	This operation defines the parent of this MCMLogicalResource object. This MCMLogicalResource object **shall not** have more than one parent. If this MCMLogicalResource object already has a parent, then an exception **should** be raised.

The MCMLogicalResource class participates in one aggregation, called MCMHasLogicalResource; see clause 5.2.2.4.9.6.6.

5.2.2.4.9.6.5 MCMLogicalResourceAtomic Class Definition

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

This object **shall 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.

5.2.2.4.9.6.6 MCMLogicalResourceComposite Class Definition

MCMLogicalResourceComposite is an abstract class that specializes MCMLogicalResource. It represents MCMResources that are composite in nature. An MCMLogicalResourceComposite represents a whole-part relationship; this produces a tree-structured class hierarchy. A composite object defines three types of objects: the whole, the part, and the assembly of the whole with its parts.

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 operations. 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 1 Gb/months, and the current usage is 750 Mb); 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 does not "know" how much usage is incurred, but can find out (e.g. by using a operation). Hence, class operations will likely be added to provide more detailed information for instances of this class.

Table 5-17 defines following operations for this class.

Table 5-17: Operations of the MCMLogicalResourceComposite Class

Operation Name	Description
getMCMLogicalResourceList() : MCMLogicalResource[1..*]	This operation returns the set of all MCMLogicalResource objects that are contained in this specific MCMLogicalResourceComposite object. This operation returns a list of zero or more MCMLogicalResources. If this MCMLogicalResourceComposite object has no children, then it **should** return a NULL MCMLogicalResource object.

Operation Name	Description
setMCMLogicalResourceList (in childObjectList : MCMLogicalResource [1..*])	This operation defines a set of MCMLogicalResource objects that will be contained by this particular MCMLogicalResourceComposite object. This operation 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. 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 objects or the objects that are contained in them. This operation creates a set of aggregations between this particular MCMLogicalResourceComposite object and each of the MCMLogicalResource objects identified in the childObjectList. 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 removes both the association class and its aggregation between each MCMLogicalResource object specified in the input parameter and this MCMLogicalResourceComposite object. All other aggregations between this MCMLogicalResourceComposite and other MCMLogicalResource objects that are not identified in the input parameter **shall not** be affected.
NOTE: The use of other MCMLogicalResource subclasses besides those defined in the present document is for further study.	

5.2.2.4.10 MCMServiceEndpoint

5.2.2.4.10.1 Overview

An MCMServiceEndpoint represents the (logical) point of delivery of a Service to an MCMConsumer.

5.2.2.4.10.2 Class Definition

This is a concrete class that specializes MCMManagedEntity. It represents a (logical) point of delivery of an MCMService to an MCMConsumer, as viewed by the MCMService that the MCMConsumer is using.

An MCMServiceEndpoint that is in use **shall** 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 state, but it is not yet instantiated.

5.2.2.4.10.3 Attribute Definition

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

5.2.2.4.10.4 Operation Definition

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

5.2.2.4.10.5 Relationship Definition

A single optional aggregation, named MCMServiceEndpointHasMCMServiceInterface, is defined. This aggregation defines the set of MCMServiceInterfaces that are associated with this particular MCMServiceEndpoint object. The multiplicity of this aggregation is 0..1 - 1. If this aggregation is instantiated (e.g. the "1" part of the 0..1 cardinality), then only one MCMServiceInterface object **shall** be allowed to 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. MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

5.2.2.4.11 MCMParty

NOTE: Should this version of the ENI Extended Core Model use MCMParty Objects?

5.2.2.4.11.1 Overview

The MCMParty class has two subclasses, as shown in Figure 5-8.

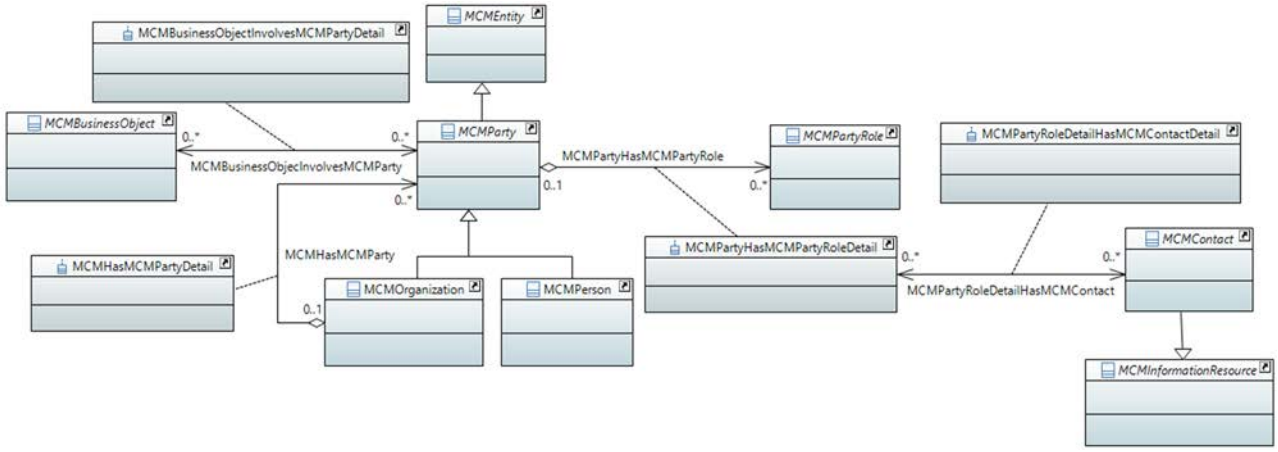

Figure 5-8: The MCMParty Hierarchy

5.2.2.4.11.2 MCMParty Class Definition

This is an abstract class that specializes MCMEntity. It represents either an individual person or a group of people that function 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. For example, an MCMParty that takes on the role HelpDesk can be used to represent any group or individual that performs a HelpDesk function.

5.2.2.4.11.3 Attribute Definition

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

5.2.2.4.11.4 Operation Definition

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 MCMManagedEntity. Since MCMPartyRoles can change dynamically at runtime, an attribute cannot accurately reflect this. In contrast, a operation can simply look for instantiated aggregations of type MCMPartyHasMCMPartyRole (see clause 5.2.2.4.11.5); it can even look at the MCMPartyHasMCMPartyRoleDetail association class, and/or associated MCMMetaData objects, if it needs further detail.

Table 5-18 defines following operations for this class.

Table 5-18: Operations of the MCMParty Class

Operation Name	Description
getMCMPartyParent() : MCMOrganization[1..1]	This operation returns the parent of this MCMParty object. 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.. This MCMParty object **shall not** have more than one parent. If this MCMParty 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. For each instance of this aggregation, this operation then adds each MCMPartyRole defined in this aggregation to an array that is returned by this operation. 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 creates a set of aggregations between this particular MCMParty object and the set of MCMPartyRole objects identified in the input parameter. This operation 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 creating an instance of the MCMPartyHasMCMPartyRole aggregation, and realizing that aggregation instance as an association class. Each created aggregation **should** have an association class (i.e. an instance of the MCMPartyHasMCMPartyRoleDetail association class).
setMCMPartyRolePartialList (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 creates a set of aggregations between this particular MCMParty object and the set of MCMPartyRole objects identified in the input parameter. This operation **shall not** affect any existing aggregated MCMPartyRole objects. 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 removes both the association class and the aggregation between this MCMParty object and each MCMPartyRole object that is aggregated by this MCMParty object. This operation does not delete any of the MCMPartyRole objects; it simply disconnects them from the MCMParty that they were aggregating. This operation **shall not** delete any MCMPartyRole object that were aggregated by this MCMParty. If no MCMPartyRole objects are aggregated by this MCMParty, then an exception **should** be raised.
delMCMPartyRolePartialList (in newPartyRoleList : MCMPartyRole[1..*])	This operation deletes a set of MCMPartyRole objects that are aggregated by this particular MCMParty object. This operation removes both the association class and the aggregation between each MCMPartyRole object specified in the input parameter and this MCMParty object. All other aggregations between this MCMParty object and other MCMPartyRole objects that are not specified in the input parameter are NOT affected. This operation **shall not** delete any MCMPartyRole object that is not named in the input parameter. If no MCMPartyRole objects are aggregated by this MCMParty, then an exception **should** be raised.

5.2.2.4.11.5 Relationship Definition

The MCMParty class defines one aggregation and participates in two other aggregations.

MCMPartyHasMCMPartyRole is an optional aggregation that defines the set of MCMPartyRoles that this particular MCMParty can take on. The multiplicity of this aggregation is 0..1 - 0..*. 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. 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. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

The MCMParty class participates in the MCMHasMCMParty (see clause 5.2.2.4.11.6.1). The MCMParty class also participates in the MCMPartyRoleDetailHasMCMContact aggregation (see clause 5.2.2.5.6.2).

5.2.2.4.11.6 MCMParty Subclasses

5.2.2.4.11.6.1 MCMOrganization Class Definition

This is a concrete class, and specializes MCMParty. An MCMOrganization defines a group of people (e.g. a set of MCMPerson, MCMOrganization, and/or their subclasses) 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.

An MCMOrganization object can interact with other MCMOrganization and MCMPerson objects directly or through their MCMPartyRole(s). Behaviour 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 behaviour of the Entity being modeled from its responsibilities and functions. This provides a more accurate and extensible model.

Table 5-19 defines following attributes for this class.

Table 5-19: Attributes of the MCMOrganization Class

Attribute Name	Description
mcmIsLegalEntity : Boolean[0..1]	This is a Boolean attribute. If its value is TRUE, then this organization is a legal entity.
mcmIsTempOrg : Boolean[0..1]	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).
mcmIsVirtualOrg : Boolean[0..1]	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).
mcmOrgName : String[1..1]	This is a string attribute, and contains the name of this MCMOrganization.

Table 5-20 defines following operations for this class.

Table 5-20: Operations of the MCMOrganization Class

Operation Name	Description
getMCMIsLegalEntity() : Boolean[1..1]	This operation returns the value of the mcmIsLegalEntity attribute.
setMCMIsLegalEntity(in isLegal : Boolean[1..1])	This operation defines the value of the mcmIsLegalEntity attribute. 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.
setMCMIsTempOrg (in isTemp : Boolean[1..1])	This operation defines the value of the mcmIsTempOrg attribute. 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.
setMCMIsVirtualOrg(in isVirtualOrg : Boolean[1..1])	This operation defines the value of the mcmIsLegalEntity attribute. If the value of this attribute is TRUE, then this MCMOrganization is a virtual organization.
getMCMOrgName() : String[1..1]	This operation returns the value of the mcmOrgName attribute.
setMCMOrgName(in newOrgName : String[1..1])	This operation defines the value of the mcmOrgName attribute.

At this time, a single optional aggregation named MCMHasMCMParty is defined for the MCMOrganization class. The multiplicity of this aggregation is 0..1 - 0..*. 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 MCMOrganization object. The cardinality on the part side (MCMParty) is 0..*; this enables an MCMOrganization 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. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

5.2.2.4.11.6.2 MCMPerson Class Definition

This is a concrete class that 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). Behaviour 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 behaviour of the individual and his or her responsibilities being modeled from its responsibilities and functions.

This provides a more accurate and extensible model.

Table 5-21 defines following attributes for this class.

Table 5-21: Attributes of the MCMPerson Class

Attribute Name	Description
mcmBirthDate : TimeAndDate[1..1]	This is a TimeAndDate attribute that contains the date (and optionally, the time) that this MCMPerson was born. If the value of this attribute is not known, then an accepted value to denote this **should** be used.
mcmBirthPlace : String[0..1]	This is a String attribute, and contains the name of the place that this MCMPerson was born. Due to variations in formatting, this is a simple string and not a type of Location object. If the value of this attribute is not known, then an empty string **should** be used.
mcmFirstName : String[0..1]	This is a String attribute, and contains the first name of this MCMPerson. If the value of this attribute is not known, then an empty string **should** be used.
mcmLastName : String[1..1]	This is a String attribute, and contains the last name of this MCMPerson. If the value of this attribute is not known, then an empty string **should** be used.

Table 5-22 defines following operations for this class.

Table 5-22: Operations of the MCMPerson Class

Operation Name	Description
getMCMBirthDate() : TimeAndDate[1..1]	This operation returns the value of the mcmBirthDate attribute.
setMCMBirthDate (in birthdate : TimeAndDate[1..1])	This operation defines the value of the mcmBirthDate attribute. A default TimeAndDate value for signifying an unknown Date **may** be used.
getMCMBirthPlace() : String[1..1]	This operation returns the value of the mcmBirthPlace attribute.
setMCMBirthPlace (in birthPlace : String[1..1])	This operation defines the value of the mcmBirthPlace attribute.
getMCMFirstName() : String[1..1]	This operation returns the value of the mcmFirstName attribute.
setMCMFirstName (in firstName : String [1..1])	This operation defines the value of the mcmFirstName attribute.
getMCMLastName() : String[1..1]	This operation returns the value of the mcmastName attribute.
setMCMLastName (in lastName : String [1..1])	This operation defines the value of the mcmastName attribute.

No relationships are currently defined for this class.

5.2.2.4.12 MCMDomain Hierarchy

5.2.2.4.12.1 Overview

A *domain* is defined as a collection of entities that share a common purpose. An MCMDomain constrains this definition and requires that each constituent MCMEntity in an MCMDomain is both uniquely addressable and uniquely identifiable within that MCMDomain. An MCMDomain (or its subclasses) should be used to provide access to both Internal and External Reference Points as well as APIs.

The MCMDomain class has a single subclass, as shown in Figure 5-9.

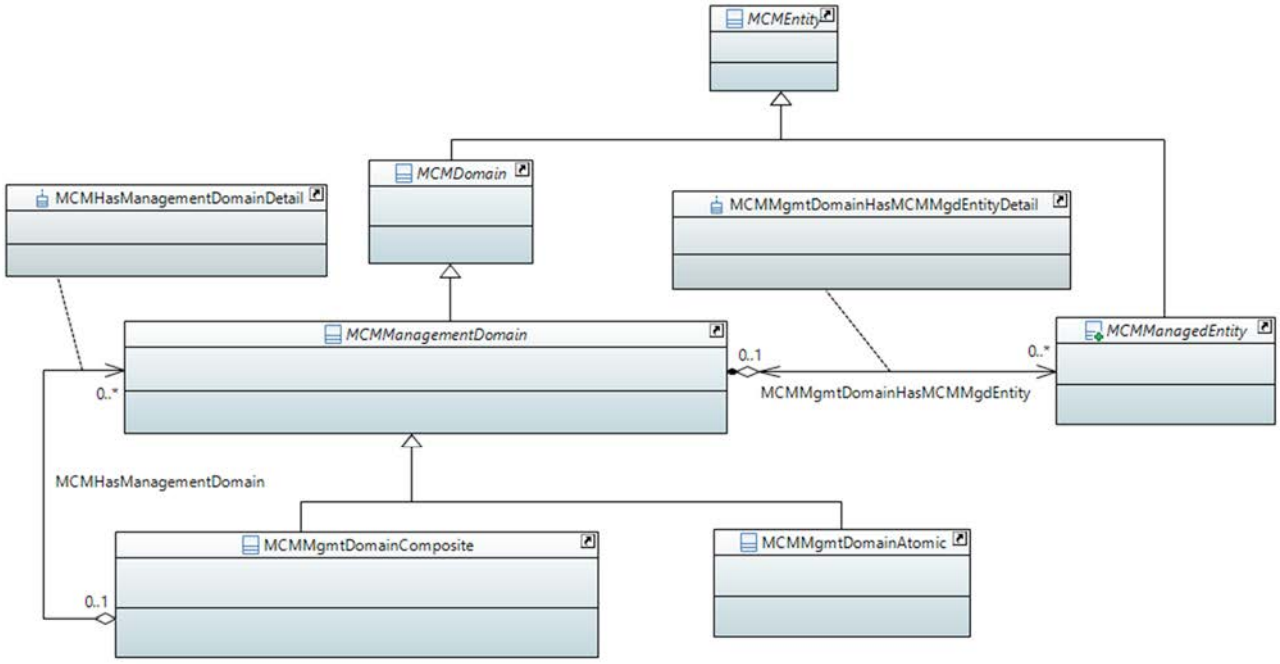

Figure 5-9: The MCMDomain Hierarchy

5.2.2.4.12.2 Class Definition

An MCMDomain is an abstract class that 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.

An MCMDomain **should not** be used to contain MCMManagedEntity objects. Instead, an MCMManagementDomain object **shall** be used.

5.2.2.4.12.3 Attribute Definition

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

5.2.2.4.12.4 Operation Definition

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

5.2.2.4.12.5 Relationship Definition

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

5.2.2.4.12.6 MCMDomain Subclasses

5.2.2.4.12.6.1 MCMManagementDomain Class Definition

MCMManagementDomain is a concrete class that specializes MCMDomain. An MCMManagementDomain is used to contain zero or more MCMManagedEntities and zero or more MCMEntities. It therefore refines the notion of a Domain by adding several important behavioural features, as specified in the following requirements:

1) Each MCMManagedEntity contained in an MCMManagementDomain **shall** be uniquely identifiable for management purposes.

2) An MCMManagementDomain **should** define a set of administrators that collectively govern the set of MCMManagedEntities that it contains. That is, all contained MCMManagedEntities are governed by at least one administrator.

3) An administrator **may** be restricted to execute a subset of operations for a given MCMManagementDomain. For example, if administrators A and B can managed a given MCMManagementDomain, then their respective management operations can be different.

4) An MCMManagementDomain **should** define a set of applications that are responsible for all envisaged governance operations, such as monitoring and configuration.

5) Different applications **may** be responsible for different governance operations. For example, monitoring and configuration operations may be performed by the same or different applications).

6) An MCMManagementDomain **should** define a common set of management mechanisms, such as policy rules, that are used to govern the behaviour of MCMManagedEntities contained in the MCMManagementDomain.

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

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

This association **may** also be further refined using OCL.

Currently, no attributes are defined for this class.

Table 5-23 defines the operations for the MCMManagementDomain class.

Table 5-23: Operations of the MCMManagementDomain Class

Operation Name	Description
getMCMMgmtDomainParent() : MCMMgmtDomainComposite[1..1]	This operation returns the parent of this MCMDomain object. If this MCMDomain object has no parent, then a NULL MCMDomainComposite object **should** be returned.
setMCMMgmtDomainParent(in newParent : MCMMgmtDomainComposite[1..1])	This operation defines the parent of this MCMDomain object. If this MCMDomain object already has a parent, then an exception **should** be raised. This MCMLocation object **shall not** have more than one parent.

At this time, a single optional 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..*. 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. 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. The semantics of the MCMMgmtDomainHasMCMManagedEntity aggregation is realized using the MCMMgmtDomainHasMCMMgdEntityDetail association class. 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. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

This class also participates in a second aggregation, called MCMHasManagementDomain; this is defined in clause 5.2.2.4.12.6.3.

5.2.2.4.12.6.2 MCMManagementDomainAtomic Class Definition

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

This class represents stand-alone MCMManagementDomain objects (i.e. they **shall 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.

5.2.2.4.12.6.3 MCMManagementDomainComposite Class Definition

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

Each MCMMgmtDomainComposite **may** contain zero or more MCMMgmtDomainAtomic and/or zero or more MCMMgmtDomainComposite objects.

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

Table 5-24 defines the operations for the MCMMgmtDomainComposite class.

Table 5-24: Operations of the MCMManagementDomainComposite Class

Operation Name	Description
getMCMMgmtDomainChildList() : MCMManagementDomain [1..*]	This operation returns the set of all MCMMgmtDomain objects that are contained in this specific MCMMgmtDomainComposite object. This operation returns a list of zero or more MCMMgmtDomain objects. If this MCMMgmtDomainComposite object has no child objects, then a NULL MCMMgmtDomain object **should** be returned.
setMCMMgmtDomainChildList (in childObjectList : MCMManagementDomain [1..*])	This operation defines a set of MCMMgmtDomain objects that will be contained by this particular MCMMgmtDomainComposite object. This operation deletes any existing contained MCMMgmtDomain objects (and their aggregations and association classes), and then instantiates a new set of aggregations between each MCMMgmtDomain object in the childObjectList and this particular MCMMgmtDomainComposite object ; in doing so, each MCMMgmtDomain object is contained within this particular MCMMgmtDomainComposite object by creating an instance of the MCMHasManagementDomain aggregation and realizing that aggregation instance as an association class. Each created aggregation **should** have an association class (i.e. an instance of the MCMHasManagementDomainDetail association class).
setMCMMgmtDomainPartialChild-List (in childObjectList : MCMManagementDomain [1..*])	This operation defines a set of one or more MCMMgmtDomain objects that are contained within this particular MCMMgmtDomainComposite object **without** affecting any other existing contained MCMMgmtDomain objects or the objects that are contained in them. This operation creates a set of aggregations between this particular MCMMgmtDomainComposite object and each of the MCMMgmtDomain objects identified in the childObjectList. Each created aggregation **should** have an association class (i.e. an instance of the MCMHasManagementDomainDetail class).
delMCMMgmtDomainChildList ()	This operation deletes **all** contained MCMMgmtDomain objects of this particular MCMMgmtDomainComposite object. This operation removes both the association class and the aggregation between this MCMMgmtDomainComposite object and each MCMMgmtDomain object that is contained in this MCMMgmtDomainComposite object.
delMCMMgmtDomainPartialChild-List (in childObjectList : MCMManagementDomain[1..*])	This operation deletes a set of MCMMgmtDomain objects from this particular MCMMgmtDomainComposite object. This operation removes both the association class and the aggregation, between each MCMMgmtDomain object specified in the input parameter and this MCMMgmtDomainComposite object. All other aggregations between this MCMMgmtDomainComposite and other MCMMgmtDomain objects that are not identified in the input parameter **shall not** be affected.

The MCMHasManagementDomainComposite class defines a single optional aggregation, called MCMHasManagementDomain. This aggregation is used to define the set of MCMManagementDomains that are contained within this particular MCMMgmtDomainComposite. Its multiplicity is defined to be 0..1 - 0..*. 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 (MCMManagementDomain) 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 MCMHasManagementDomain 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. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

5.2.2.4.13 MCMBusinessObject Hierarchy

5.2.2.4.13.1 Overview

The MCMBusinessObject class hierarchy defines the abstract concept of business objects that are types of MCMEntities, but not types of MCMManagedEntities. Examples include Order, TroubleTicket, and Report.

The utility of this class hierarchy is to connect business objects with the network, business applications, and/or Customers for modelling the full lifecycle of business operations that results in network services.

The MCMBusinessObject class has two subclasses, as shown in Figure 5-10.

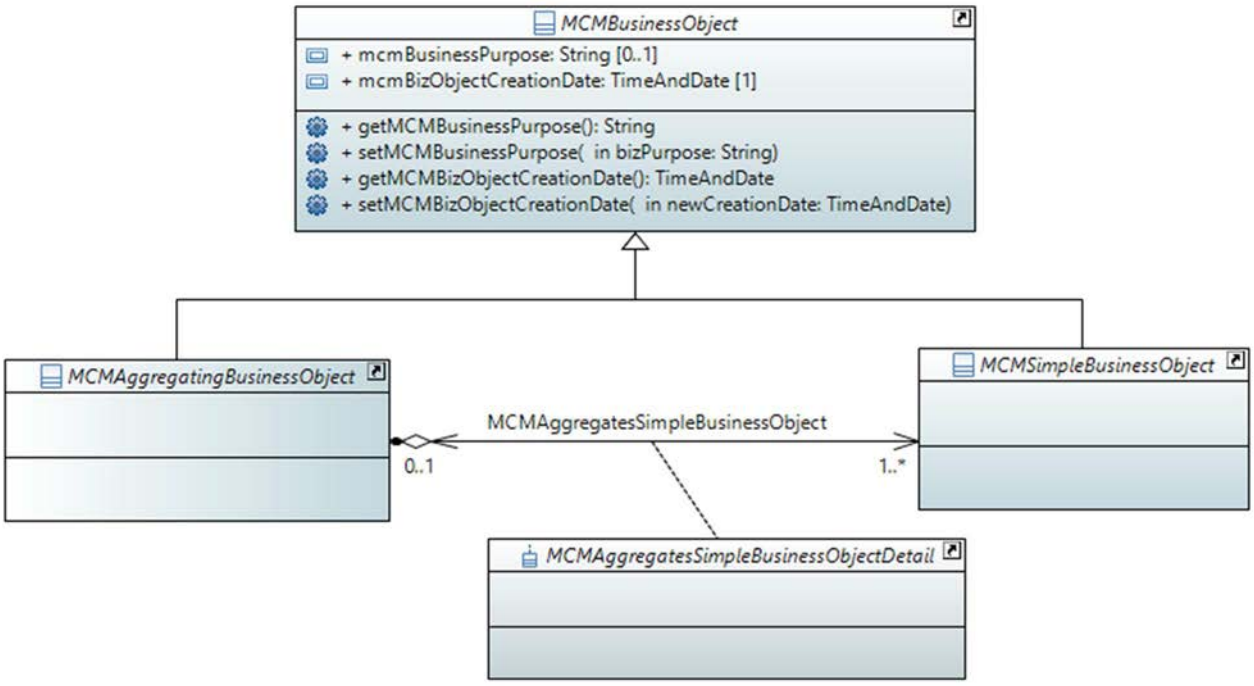

Figure 5-10: The MCMBusiness Hierarchy

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, POCs, and Quotes. The full model of business entities is defined in the MEF Business Model (MBM). In the MBM, all business objects that contain other objects, such as Orders, are subclasses from MCMAggregatingBusinessObject. Similarly, all business objects that are contained by another business object, such as OrderItems, are subclassed from MCMSimpleBusinessObject.

NOTE: Should this version of the ENI Extended Core Model use MCMBusiness Objects?

5.2.2.4.13.2 Class Definition

MCMBusinessObject 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, Inventory, and Reports.

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 as attributes of the MCMBusinessObject class. More specifically, the former is provided by MCMEntityHasMCMMetaData, since MCMBusinessObject is a subclass of MCMEntity and therefore inherits this aggregation. The latter is an attribute that is already defined in MCMMetaData.

5.2.2.4.13.3 Attribute Definition

Table 5-25 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 will be defined using a combination of relationships and operations.

Table 5-25: Attributes of the MCMBusinessObject Class

Attribute Name	Description
mcmBusinessPurpose : String[0..1]	This is a string attribute that contains a description of the business purpose of this MCMBusinessObject. If an object does not have a value for the mcmCommonName attribute, then an empty string **should** be used.
mcmBizObjCreationDate : TimeAndDate[0..1]	This is a TimeAndDate attribute that contains the date and time when this object was produced. This attribute **should** have a complete and valid time and/or date. The implementation **may** ensure that the fields in this data type are set to an appropriate default value.

5.2.2.4.13.4 Operation Definition

Table 5-26 defines the operations of the MCMBusinessObject class.

Table 5-26: Operations of the MCMBusinessObject Class

Operation Name	Description
getMCMBusinessPurpose() : String[1..1]	This operation returns the mcmBusinessPurpose textual attribute for this particular MCMBusinessObject. 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. If a business purpose is not defined, then an empty string **should** be used.
getMCMDateCreated() : TimeAndDate[1..1]	This is a TimeAndDate attribute that contains the date and time that this object was created. This attribute **should** have a complete and valid time and/or date. The implementation **may** ensure that the fields in this data type are set to an appropriate default value.
setMCMDateCreated (in newCreationDate: TimeAndDate[1..1])	This is a TimeAndDate attribute that contains the date and time that this object was created. This attribute **should** have a complete and valid time and/or date.

5.2.2.4.13.5 Relationship Definition

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

5.2.2.4.13.6 MCMBusinessObject Subclasses

5.2.2.4.13.6.1 MCMAggregatingBusinessObject Class Definition

This is an abstract class that specializes MCMBusinessObject. Its purpose is to represent a commonly occurring business pattern of an object that contains different items that have a numerical significance. These contained items are called "line items". Examples include Order and OrderItems, Quote and QuoteItems, and ProductOfferQualification (POC) and POCItems objects.

There are two ways to realize this restriction. The first is to apply OCL to the aggregation, while the second is to use the association class to restrict which types of part components (e.g. an OrderItem) can be aggregated by which type of aggregating object (i.e. an Order in this example).

This class currently has no attributes. This is because its purpose is to enable its concrete subclasses to aggregate (i.e. contain) concrete subclasses of the MCMSimpleObject class.

Similarly, this class currently has no operations.

This class participates in one optional relationship, called MCMAggregatesSimpleBusinessObject. This defines the set of concrete subclasses of MCMSimpleBusinessObject that a concrete subclass of this MCMAggregatingBusinessObject can contain.

All subclasses of MCMAggregatingBusinessObject inherit this relationship. The multiplicity of this aggregation is 0..1 - 1..*. If this aggregation is instantiated (e.g. the "1" part of the 0..1 cardinality), then one or more MCMSimpleBusinessObject objects are aggregated by this particular MCMAggregatingBusinessObject.

The semantics of this aggregation are defined by the MCMAggregatesSimpleBusinessObjectDetail association class. This enables a particular set of MCMSimpleBusinessObjects to be contained by a given MCMAggregatingBusinessObject. The MCMAggregatesSimpleBusinessObjectDetail association class is abstract; this enables the developer to build concrete subclasses of this association class to define details specific to different combinations of MCMAggregatingBusinessObject and MCMSimpleBusinessObject. For example, a concrete association class could be defined to restrict which particular subclasses of MCMSimpleBusinessObject may be aggregated by a particular concrete subclass of MCMAggregatingBusinessObject. The Policy Pattern may be used to control which specific MCMOrderItem objects are attached to a given MCMOrderStructure object for a given context. Note that MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

5.2.2.4.13.6.2 MCMSimpleBusinessObject Class Definition

This is an abstract class that represents different types of objects that enumerate different items of an aggregating object. Examples, including Order Items, Quote Items, and POC Item objects, which can each be contained by Order, Quote, and POC objects, respectively.

This class currently has no attributes. This is because its purpose is to enable its concrete subclasses to be aggregated by (i.e. contained by) concrete subclasses of the MCMAggregatingBusinessObject class.

Similarly, this class currently has no operations.

Similarly, this class currently has no relationships.

5.2.2.5 MCMInformationResource Hierarchy

5.2.2.5.1 Overview

Figure 5-11 shows the MCMInformationResource class hierarchy, along with some important aggregations that it participates in. The following clauses describe the classes in the MCMInformationResource class hierarchy in more detail.

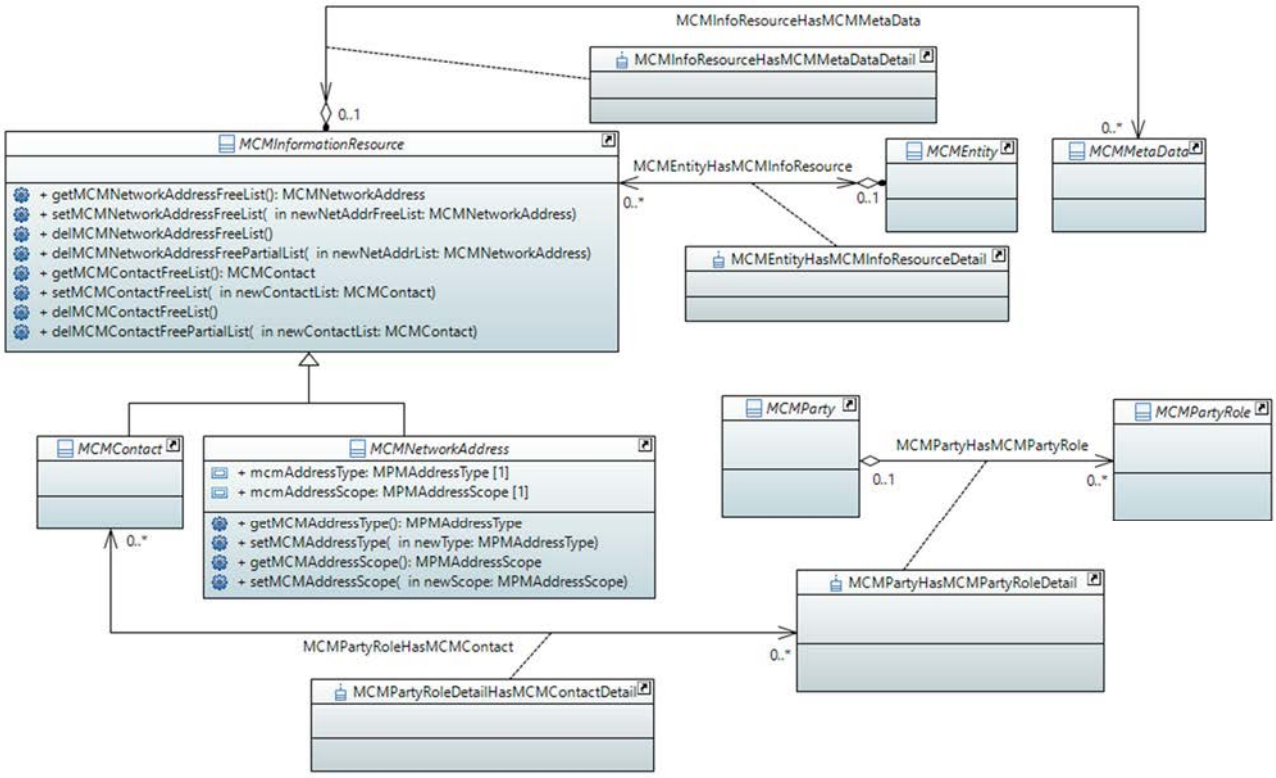

Figure 5-11: The MCMInformationResource Hierarchy

NOTE: Should this version of the ENI Extended Core Model use Business Objects?

5.2.2.5.2 MCMInformationResource Class Definition

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

5.2.2.5.3 Attribute Definition

There are no attributes defined for this class. This is because its two subclasses are semantically different, and common information, such as version and time created, can be supplied by metadata.

5.2.2.5.4 Operation Definition

Table 5-27 defines following operations for this class.

Table 5-27: Operations of the MCMInformationResource Class

Operation Name	Description
getMCMNetworkAddress-FreeList() : MCMNetworkAddress[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). If no MCMNetworkAddress objects are found, then a NULL MCMNetworkAddress object **should** be returned.

Operation Name	Description
setMCMNetworkAddressFreeList (in newNetAddrFreeList : MCMNetworkAddress[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). Each element in the input parameter is used to create an MCMNetworkAddress. This operation does not associate any MCMIPAddress with an MCMEntity. This operation **shall not** affect any other MCMNetworkAddress object that are aggregated by any MCMEntity class. If an MCMNetworkAddress defined in the input parameter already exists, then an exception **should** be raised.
delMCMNetworkAddressFreeList()	This operation is used to delete all free-standing MCMNetworkAddress objects. This operation **shall not** affect any other MCMNetworkAddress object that are aggregated by any MCMEntity class.
delMCMNetworkAddressFree-PartialList (in newNetworkAddressList : MCMNetworkAddress[1..*])	This operation deletes **all** MCMNetworkAddress object instances that are specified in its input parameter that are free-standing (i.e. not aggregated by any subclass of MCMEntity). This operation **shall not** affect any MCMNetworkAddress object that is not specified in its input parameter. This operation **shall not** affect any other MCMNetworkAddress object that are aggregated by any MCMEntity class.
getMCMContactFreeList() : MCMContact[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. 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). This operation only defines the MCMContact objects; it does not associate them with an MCMEntity. This operation **shall not** affect any other MCMContact object that are aggregated by any MCMEntity class.
delMCMContactFreeList()	This operation is used to delete all free-standing MCMContact objects. This operation **shall not** affect any other MCMContact object that are aggregated by any MCMEntity class.
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). Each element in the input parameter is used to delete an MCMContact. If an MCMContact corresponding to an element in the input parameter is not found, then an error **shall** be raised. This operation **shall not** affect any MCMContact object that is not specified in its input parameter. This operation **shall not** affect any other MCMContact object that are aggregated by any MCMEntity class.

5.2.2.5.5 Relationship Definition

At this time, the MCMInformationResource class defines a single optional aggregation, called MCMInfoResourceHasMCMMetaData. The multiplicity of this aggregation is 0..1 - 0..*. 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. 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. The MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

The MCMInformationResource participates in a second aggregation, called MCMEntityHasMCMInfoResource (see clause 5.2.2.4.2.5).

5.2.2.5.6 MCMInformationResource Subclasses

5.2.2.5.6.1 MCMNetworkAddress Class Definition

This is an abstract class that 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 MCMNetworkAddresses). 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).

Table 5-28 defines the attributes for this class.

Table 5-28: Attributes of the MCMNetworkAddress Class

Attribute Name	Description
mcmAddressType: MPMAddressType[1..1]	This attributes defines the type of network address that this instance is. Valid values are defined in the MPMAddressType enumeration.
mcmAddressScope : MPMAddressScope[1..1]	This attributes defines the scope for this network address. Valid values are defined in the MPMAddressScope enumeration.

Table 5-29 defines the operations for this class.

Table 5-29: Operations of the MCMNetworkAddress Class

Operation Name	Description
getMCMNetworkAddress() : MPMAddressType[1..1]	This operation returns the value of the mcmAddressType attribute. If a NULL or empty value is found, then an error **shall** be raised.
setMCMNetworkAddress(in newType : MPMAddressType [1..1])	This operation defines a new value for the mcmAddressType attribute.
getMCMNetworkScope() : MPMAddressScope[1..1]	This operation returns the value of the mcmAddressScope attribute. If a NULL or empty value is found, then an error **shall** be raised.
setMCMNetworkScope(in newScope : MPMAddressScope[1..1])	This operation defines a new value for the mcmAddressType attribute.

No relationships are currently defined for this class.

5.2.2.5.6.2 MCMContact Class Definition

This is a concrete class that 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 or MCMManagedEntity).

No attributes are currently defined for this class. Extensions to the MCM are encouraged to define supporting classes that collect different contact methods, such as types of emails, phone numbers, and other information; relationships could then be defined between selected supporting classes and the MCMContact class to generalise these settings. This is because there is no specific email or phone number that can always be used for a given MCMContact. In addition, an MCMParty that can have multiple MCMPartyRoles (e.g. Employee, HelpDesk, and HelpDeskSupervisor) would likely need different contact information for each of those MCMPartyRoles.

No operations are currently defined for this class.

This class participates in a single optional association, called MCMPartyRoleDetailHasMCMContact. This is an association between the MCMPartyRoleDetail association class and the MCMContact class. An MCMPartyRoleDetail is an association class that defines a set of MCMPartyRoles that are used by a given MCMParty. Hence, the MCMPartyRoleDetailHasMCMContact association defines the set of MCMContacts that are related to this particular MCMPartyRoleDetail object. Put another way, it defines the set of MCMParty objects that are playing one or more specific 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..*. 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. 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. The MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. 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 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

5.2.2.6 MCMMetaData Hierarchy

5.2.2.6.1 Overview

The purpose of the MCMMetaData hierarchy is to define objects that describe and/or prescribe the characteristics and/or behaviour of MCMEntity and MCMInformationResource objects. This is done by instantiating an association between the MCMEntity or MCMInformation resource object and the appropriate set of MCMMetaData objects.

Figure 5-12 shows the top portion of the MCMMetaData class hierarchy.

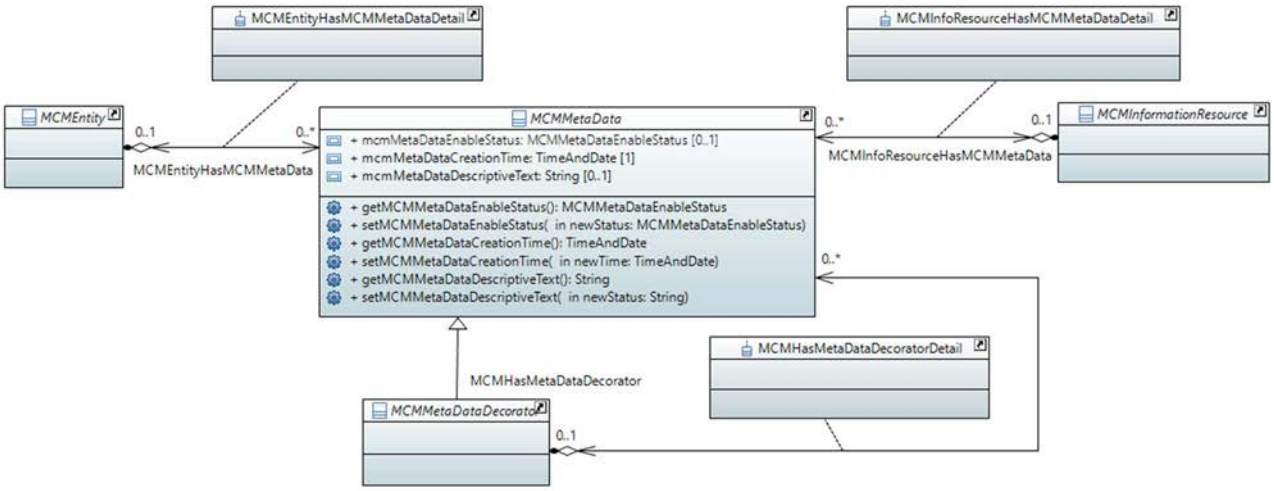

Figure 5-12: The MCMMetaData Hierarchy, Top Level View

5.2.2.6.2 MCMMetaData Class Definition

This is an abstract class that specializes MCMRootEntity. It defines prescriptive and/or descriptive information about the MCMEntity or MCMInformationResource object(s) to which it is attached. These descriptive and/or prescriptive characteristics and behaviour **shall not** be an inherent, distinguishing characteristic or behaviour 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.

5.2.2.6.3 Attribute Definition

Table 5-30 defines following attributes for this class.

Table 5-30: Attributes of the MCMMetaData Class

Attribute Name	Description
mcmMetaDataEnableStatus : MCMMetaDataEnableStatus[0..1]	This is an optional enumeration that defines whether the MCMEntity that this MCMMetaData object refers to is enabled for normal operation or not.
mcmMetaDataCreationTime : TimeAndDate[1..1]	This contains a date stamp and a timestamp that defines the date and time that this MCMMetaData object was created. This attribute **should** have a complete and valid time and/or date. The implementation **may** ensure that the fields in this data type are set to an appropriate default value.
mcmMetaDataDescriptiveText : String[0..1]	This attribute contains descriptive content about the MCMEntity or MCMInformationResource to which it is attached.

5.2.2.6.4 Operation Definition

Table 5-31 defines following operations for this class.

Table 5-31: Operations of the MCMMetaData Class

Operation Name	Description
getMCMMetaDataEnableStatus() : MCMMetaDataEnableStatus[1..1]	This operation returns the value of the mcmMetaDataEnableStatus attribute. If this object does not have a value for the mcmLocationDataList attribute, then a NULL string **should** be returned by the getMCMLocationDataList operation.
setMCMMetaDataEnableStatus (in newStatus : MCMMetaDataEnableStatus1..1])	This operation defines the value of the mcmMetaDataEnableStatus attribute.
getMCMMetaDataCreationTime() : TimeAndDate[1..1]	This operation returns the value of the mcmMetaDataCreationTime attribute. This attribute **should** have a complete and valid time and/or date. 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 operation defines the value of the mcmMetaDataCreationTime attribute. The implementation **may** ensure that the fields in this data type are set to an appropriate default value.
getMCMMetaDataDescriptiveText() : String[1..1]	This operation returns the value of the mcmMetaDataDescriptiveText attribute. If this object does not have a value for this attribute, then a NULL string **should** be returned.
setMCMMetaDataDescriptiveText (in newStatus : String[1..1])	This operation defines the value of the mcmMetaDataDescriptiveText attribute. A NULL string **should not** be used.

5.2.2.6.5 Relationship Definition

The MCMMetaData class participates in three aggregations. The first two, MCMEntityHasMCMMetaData and MCMInfoResourceHasMCMMetaData, have been previously defined in clauses 5.2.2.4.2.5 and 5.2.2.5.4, respectively. The third, MCMHasMCMMetaDataDecorator, is defined in clause 5.2.2.6.6.2.3.

5.2.2.6.6 MCMMetaData Subclasses

5.2.2.6.6.1 Overview

Figure 5-13 shows the important subclasses of MCMMetaData.

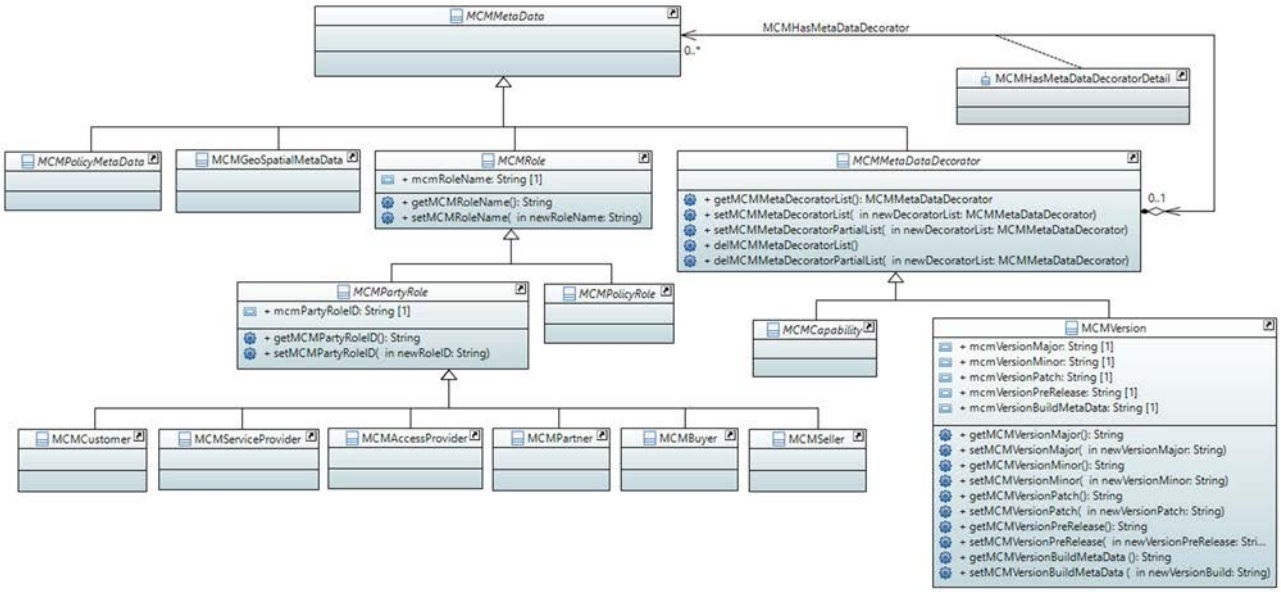

Figure 5-13: The MCMMetaData Important Subclasses

5.2.2.6.6.2 MCMRole Class Hierarchy

5.2.2.6.6.2.1 MCMRole Class Definition

This is an abstract class that specializes MCMMetadata. It represents a set of characteristics and behaviours (also referred to as responsibilities throughout this document) 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 [i.8] 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 the present document; however, this is why the getRoleCombination operation is provided by this class.

Implementers **should** use the Role-Object pattern to implement MCMRoles.

Table 5-32 defines the following attributes for this class.

Table 5-32: Attributes of the MCMRole Class

Attribute Name	Description
mcmRoleName : String[1..1]	This is a string attribute that contains the name of this Role object. This is different from the mcmCommonName attribute, which defines a name by which this object is known. The mcmRoleName attribute **shall not** be used as a naming attribute (i.e. to uniquely identify an instance of this object). The mcmRoleName attribute **shall not** be empty or Null.

Table 5-33 defines following operations for this class.

Table 5-33: Operations of the MCMRole Class

Attribute Name	Description
getMCMRoleName : String[1..1]	This operation returns the name of this MCMRole object. The mcmRoleName attribute **shall not** be used as a naming attribute (i.e. to uniquely identify an instance of this object). The mcmRoleName attribute **shall not** be empty or Null string.
setMCMRoleName (in newRoleName : String[1..1])	This operation defines the name of this MCMRole object. The mcmRoleName attribute **shall not** be used as a naming attribute (i.e. to uniquely identify an instance of this object). The mcmRoleName attribute **shall not** be empty or Null string.

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

5.2.2.6.6.2.2 MCMPartyRole Class Definition

This is an abstract class that specializes MCMRole. It represents a set of unique behaviours played by an MCMParty in a given context. Table 5-34 defines the attributes for this class:

Implementers s**hould** use the Role-Object pattern to implement MCMPartyRoles.

Table 5-34: Attributes of the MCMPartyRole Class

Attribute Name	Description
mcmPartyRoleID : String[1..1]	This is a mandatory string attribute that contains a unique value that enables instances of this MCMPartyRole to be disambiguated from other MCMPartyRoles (including MCMPartyRoles of the same object type). This attribute **shall not** be used as an objectID, since one is inherited from MCMRootEntity. The value of this attribute **shall not** be a NULL or EMPTY string.

Table 5-35 defines following operations for this class.

Table 5-35: Operations of the MCMPartyRole Class

Attribute Name	Description
getMCMPartyRoleID : String[1..1]	This operation returns the ID of this MCMPartyRole object. The mcmPartyRoleID attribute **shall not** be used as a naming attribute (i.e. to uniquely identify an instance of this object). The mcmPartyRoleID attribute **shall not** be a NULL or empty string.
setMCMPartyRoleID (in newRoleID : String[1..1])	This operation defines the ID of this MCMPartyRole object. The mcmPartyRoleID attribute **shall not** be used as a naming attribute (i.e. to uniquely identify an instance of this object). The mcmPartyRoleID attribute **shall not** be a NULL or empty string.

MCMParty defines five important operations: getMCMPartyRoleList, setMCMPartyRoleList, setMCMPartyRolePartialList, delMCMPartyRoleList, and delMCMPartyRolePartialList; see clause 5.2.2.4.11.

The MCMPartyRoleDetailHasMCMContact was defined in clause 5.2.2.5.6.2.

5.2.2.6.6.2.3 MCMMetaDataDecorator Class Definition

This is an abstract class that 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 5-36 defines the operations of this class.

Table 5-36: Operations of the MCMMetaDataDecorator Class

Attribute Name	Description
getMCMMetaDecoratorList() : MCMMetaDataDecorator[1..*]	This operation returns the set of MCMMetaDataDecorator objects that are decorating this MCMMetaData object. 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 decorate this MCMMetaData object. This operation creates a set of aggregations between this particular MCMMetaData object and the set of MCMMetaDataDecorator objects identified in the input parameter. 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 creating an instance of the MCMHasMetaDataDecorator aggregation and realizing that aggregation instance as an association class. Each created aggregation **should** have an association class (i.e. an instance of the MCMHasMetaDataDecoratorDetail association class).
setMCMMetaDecoratorPartial-List (in newDecoratorList : MCMMetaDataDecorator[1..*])	This operation defines a set of one or more MCMMetaDataDecorator objects that decorate this MCMMetaData object without affecting any other existing MCMMetaDataDecorator objects that are decorating this MCMMetaData object. This operation creates a set of aggregations between this particular MCMMetaData object and the set of MCMMetaDataDecorator objects identified in the input parameter. 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 removes the association class and the aggregation between this MCMMetaData object and each MCMMetaDataDecorator object that is decorating this MCMMetaData object. This operation does not delete any of the MCMMetaDataDecorator objects; it simply disconnects them from the MCMMetaData that they were decorating.
delMCMMetaDecoratorPartial-List (in newDecoratorList : MCMMetaDataDecorator [1..*])	This operation deletes a set of MCMMetaDataDecorator objects that are decorating this particular MCMMetaData object. This operation removes both the association class and the aggregation between each MCMMetaDataDecorator object specified in the input parameter and this MCMMetaData object. All other aggregations between this MCMMetaData object and other MCMMetaDataDecorator objects that are not specified in the input parameter **shall not** be affected.

A single optional aggregation, called MCMHasMetaDataDecorator, is defined for MCMMetaDataDecorator. This 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..*. 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. The MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

5.2.2.6.6.2.4 MCMCapability Class Definition

This is an abstract class that specializes MCMMetaDataDecorator. It represents a set of features (i.e. attributes, operations, and relationships) 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 be realized using relationships and/or operations. For example, the set of mandatory, recommended, and optional capabilities of a given MCMManagedEntity can be gathered and sorted by using an appropriate operation.

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

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

5.2.2.6.6.2.5 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 MCMEntity or MCMInformationResource. This approach has two main benefits:

1) It enables a standardized form of versioning to be attached to any MCMEntity or MCMInformationResource.

2) It enables a standardized form of version information to be easily changed at runtime by wrapping an object instance of the MCMEntity or MCMInformationResource class (or its subclasses) with all or part of this object.

Version information is defined in a generic format based on the Semantic Versioning 2.0.0 [8] specification as follows:

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

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

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

1) mcmVersionMajor denotes a new release. This number **shall** be incremented when either changes are introduced that are not backwards-compatible, and/or new functionality not previously present is introduced.

2) mcmVersionMinor denotes a minor release. This number **shall** 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.

3) mcmVersionPatch denotes a version that consists ONLY of bug fixes. It **shall** be incremented when these bug fixes are NOT backwards-compatible.

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

1) Precedence **shall** 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 **shall** be determined by the first difference when comparing each of these identifiers, from left to right, as follows:

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

   - 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).

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

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

2) Numeric identifiers always have lower precedence than non-numeric identifiers.

3) 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 5-37 defines the attributes of this class.

Table 5-37: Attributes of the MCMVersion Class

Attribute Name	Description
mcmVersionMajor : String[1..1]	This is a mandatory string attribute that 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. A major version **may** indicate that this version has changes that are NOT backwards-compatible. The lack of backwards-compatibility **may** be denoted using attached MCMMetaData and/or using the mcmVersionBuildMetaData class attribute. The major version X (i.e. X.y.z, where X > 0) **shall** be incremented if any backwards incompatible changes are introduced. The major version X (i.e. X.y.z, where X > 0) **shall** be incremented if any new functionality is introduced. The minor and patch version numbers **shall** be reset to 0 when the major version number is incremented.
mcmVersionMinor : String[1..1]	This is a mandatory string attribute that indicates that this release contains a set of features and/or bug fixes that are backwards-compatible. The minor version Y (i.e. x.Y.z, where x > 0) **shall** be incremented if new, backwards-compatible changes are introduced. A minor version indicates that this release contains a set of features and/or bug fixes that **shall** be backwards-compatible. The special string "0.1.0" is for initial development and **shall not** be considered stable. The minor version **shall** be incremented if any features are marked as deprecated. The minor version **may** be incremented if improved functionality is introduced. The patch version number **shall** be reset to 0 when the minor version number is incremented.
mcmVersionPatch : String[1..1]	This is a mandatory string attribute that indicates this version ONLY contain bug fixes. A bug fix is defined as an internal change that fixes incorrect behaviour. A patch version indicates that this version **shall** only contain bug fixes. The patch version Z (i.e. x.y.Z, where x > 0) **shall** be incremented if new, backwards-compatible changes are introduced.
mcmVersionPreRelease : String[0..1]	This is an optional string attribute that indicates that the version is unstable and might not satisfy the intended compatibility requirements as denoted by its associated normal version. Identifiers **shall** comprise only ASCII alphanumerics and a hyphen. Identifiers **shall not** be empty. Numeric identifiers **shall not** include leading zeroes.
mcmVersionBuildMetaData : String[0..1]	This is an optional string attribute that contains build metadata. These are metadata that explain changes made to the current non-production release. Examples include: 1.0.0.-alpha+1, 1.0.0+20130313144700, and 1.0.0-beta+exp.sha.5114f85. Identifiers **shall** be made up of only ASCII alphanumerics and a hyphen. Identifiers **shall not** be empty. Build metadata **should** be ignored when determining version precedence.

Table 5-38 defines the operations of this class.

Table 5-38: Operations of the MCMVersion Class

Operation Name	Description
getMCMVersionMajor() : String[1..1]	This operation returns the value of the mcmVersionMajor attribute. The value of this attribute **shall not** be a NULL or an empty string value.
setMCMVersionMajor(in newVersionMajor : String[1..1])	This operation defines the value of the mcmVersionMajor attribute. The value of this attribute **shall not** be a NULL or an empty string value.
getMCMVersionMinor() : String[1..1]	This operation returns the value of the mcmVersionMinor attribute. The value of this attribute **shall not** be a NULL or an empty string value.
setMCMVersionMinor(in newVersionMinor : String[1..1])	This operation defines the value of the mcmVersionMinor attribute. The value of this attribute shall not be a NULL or an empty string value.
getMCMVersionPatch() : String[1..1]	This operation defines the value of the mcmVersionPatch attribute. The value of this attribute **shall not** be a NULL or an empty string value.

Operation Name	Description
setMCMVersionPatch(in newVersionPatch : String[1..1])	This operation defines the value of the mcmVersionPatch attribute. The value of this attribute **shall not** be a NULL or an empty string value.
getMCMVersionPreRelease() : String[1..1]	This operation defines the value of the mcmVersionPreRelease attribute. Identifiers **shall not** be empty. The value of this attribute **shall not** be a NULL or an empty string value.
setMCMVersionPreRelease(in newVersionPreRelease : String[1..1])	This operation defines the value of the mcmVersionPreRelease attribute. Identifiers **shall not** be empty. The value of this attribute **shall not** be a NULL or an empty string value.
getMCMVersionBuildMetaData : String[1..1]	This operation defines the value of the mcmVersionBuildMetaData attribute. Identifiers **shall** be made up of only ASCII alphanumerics and a hyphen. Identifiers **shall not** be empty. The value of this attribute **shall not** be a NULL or an empty string value.
setMCMVersionBuildMetaData(in newVersionBuild : String[1..1])	This operation defines the value of the mcmVersionBuildMetaData attribute. Identifiers **shall** be made up of only ASCII alphanumerics and a hyphen. Identifiers **shall not** be empty. The value of this attribute **shall not** be a NULL or an empty string value.

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

5.2.3 ENI Extensions to the MCM

5.2.3.1 Introduction

The MCM provides an extensible framework that is organized into three class hierarchies: Entity, InformationResource, and MetaData. This clause defines extensions to the Entity class hierarchy in order to represent key concepts that are needed for an ENI System to achieve its functional requirements as described in clause 5 of ETSI GS ENI 005 [3].

5.2.3.2 Naming Rules

The naming rules of ENI extensions to the MCM model follow the naming rules of the MCM (see clause 5.2.2.2), except as detailed below.

1) Class extension names defined by ENI **shall** be named with an "ENI" prefix. This serves two purposes. First, it helps provide context to textual descriptions of these model elements. Second, it enables ENI-defined extensions to the MCM model elements, patterns, and approaches to be easily recognized.

2) ENI class names **shall** be in UpperCamelCase.

3) ENI attribute names **shall** be in lowerCamelCase. Attribute names that begin with an underscore are private attributes that reference an end of an association.

4) ENI relationship names **shall** be in UpperCamelCase (i.e. the first letter is capitalized). Relationship names shall not begin with any non-alphabetic character, and no spaces are allowed.

5) Each attribute **shall** be prefixed with "eni".

6) Each relationship **shall** be prefixed with "ENI".

5.2.3.3 Events

5.2.3.3.1 Introduction

An Event is a mandatory abstract class that models significant occurrences that contain data and/or knowledge pertaining to a specific context. An Event carries information that may be used by one or more to evaluate whether a set of MCMEntity objects should execute some behaviour or not (e.g. whether it should make a state transition or not). For MCMUnManagedEntity objects, this is done using a proxy object. For example, a Rack or a Chassis is a type of UnManagedEntity. However, both could have a type of MCMManagedEntity object, such as a sensor, attached to them that could react to an event. Hence, while the Rack or Chassis itself has no behaviour, it could contain a sensor (or similar object) that provides information on behalf of the Rack or Chassis (e.g. the number of free slots).

The information may be the event itself (e.g. sending an acknowledgement) or the Event may carry (or point to) one or more data objects. Events can be applied to the MCMParty, MCMManagedEntity, MCMBusinessObject, and/or MCMDomain classes.

5.2.3.3.2 ENIEvent Class Definition

This is a mandatory abstract class.

An ENIEvent is a significant occurrence that contains data and/or knowledge pertaining to a specific context. An ENIEvent carries information that may be used by one or more MCMEntity objects to evaluate whether an MCMEntity object should execute some behaviour or not (e.g. whether it should make a state transition or not). The information may be the Event itself (e.g. sending an acknowledgement) or the Event may carry one or more data objects.

An Event has a distinct identity - a set of attributes and associated operations that are used to distinguish multiple instances of the same Event, and ensure that the correct Identity data and operations to verify those data are bound to the correct Context.

Events may be asynchronous that can happen at arbitrary times (e.g. signal, time and change) or synchronous (e.g. a call).

Figure 5-14 defines the top level classes, attributes, and operations of the ENIEvent class hierarchy.

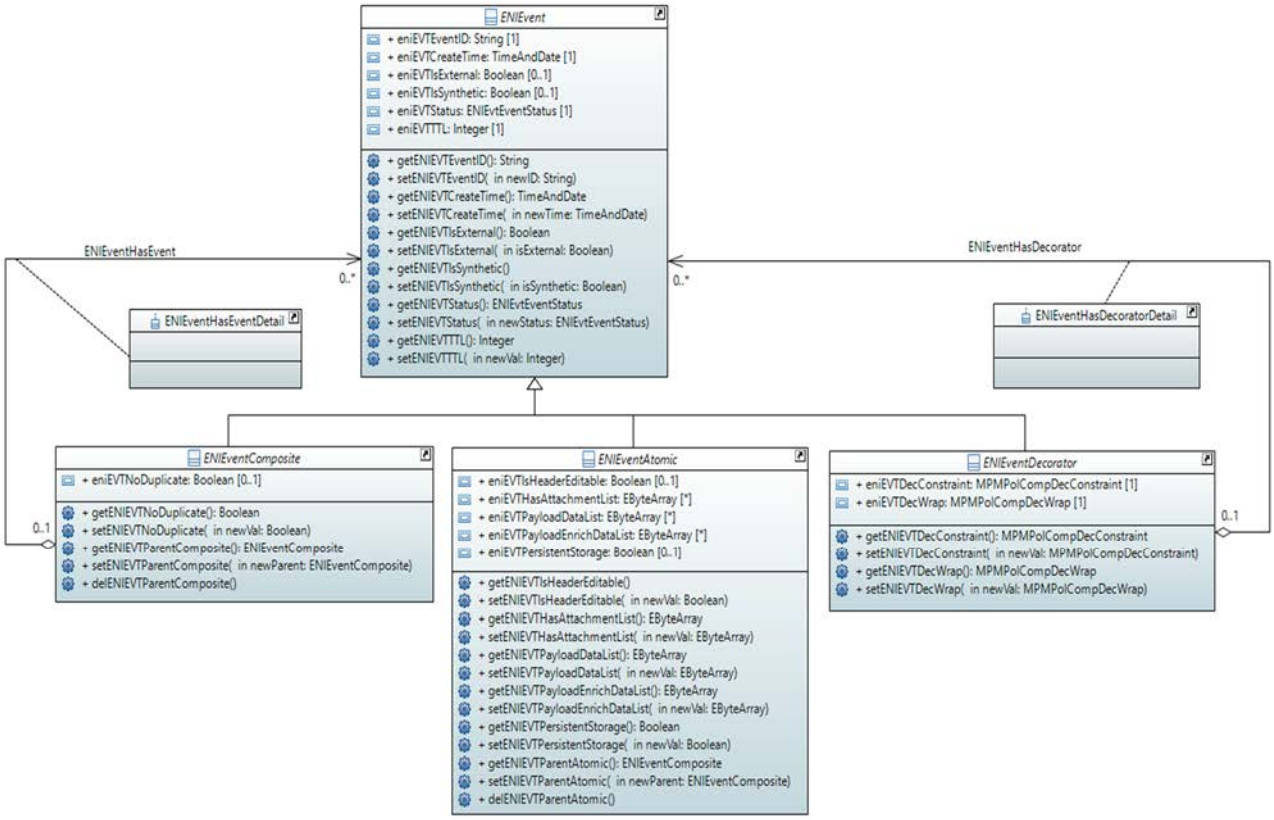

Figure 5-14: Overview of the ENIEvent Class Hierarchy

5.2.3.3.3 Attribute Definition

Table 5-39 defines the attributes of this class. An ENIEvent may be defined as valid for a particular time period; this would be done using attached MCMMetaData. Similarly, a location may be associated with an ENIEvent; this could be defined using an explicit association between MCMLocation and ENIEvent; alternatively, the existing MCMMgdEntityRefersToMCMUnMgdEntity association could be used.

Table 5-39: Attributes of the ENIEvent Class

Attribute Name	Description
eniEVTEventID : String[1..1]	This is a mandatory string attribute, and contains a unique value that enables instances of this ENIEvent object to be disambiguated from other ENIEvent objects (including ENIEvent objects of the same type). The value of this attribute **shall not** be a NULL or EMPTY string.
eniEvtIsExternal : Boolean[0..1]	This is an optional Boolean attribute. If the value of this attribute is TRUE, then this ENIEvent object originated from a source external to the ENI System. Otherwise, this ENIEvent object was generated by the system being monitored.
eniEvtIsSynthetic : Boolean[0..1]	This is an optional Boolean attribute. If the value of this attribute is TRUE, then this ENIEvent object was generated by the ENI System (e.g. for aggregating multiple events). Otherwise, this ENIEvent object is a natural event that was generated by the system being monitored.
eniEvtCreateTime : String[0..1]	This is an optional TimeAndDate attribute that contains the date and time that this ENIEvent object was created.
eniEvtStatus : ENIEventStatus[1..1]	This is an optional enumerated string attribute that contains the status information of this ENIEvent (e.g. processed, ignored). Allowed values are defined in the ENIEventStatus enumeration.
eniEVTTTL : Integer[0..1]	This attribute defines the Time-To-Live (TTL) for this ENIEvent object. The TTL is a value for the period of time that a packet, or data, should exist on a computer or network before being discarded. A default value **may** be defined for this attribute.

5.2.3.3.4 Operation Definition

Table 5-40 defines the operations of this class.

Table 5-27: Operations of the ENIEvent Class

Operation Name	Description
getEVTEventID() : String[1..1]	This operation returns the value of the evtEventID attribute, which is a textual identifier to disambiguate this ENIEvent object instance from all other ENIEvents. There are no input parameters to this operation. If an attribute value is NOT found, then an error **shall** be returned. The value of this attribute **shall not** be an empty or NULL string.
setEVTEventID(in newID : String[1..1])	This operation sets the value of the evtEventID attribute, which is a textual identifier to disambiguate this ENIEvent object instance from all other ENIEvents. This operation takes a single input parameter, called newID, which is a String attribute. The value of this attribute **shall not** be an empty or NULL string.
getEVTCreateTime() : TimeAndDate[1..1]	This operation returns the value of the evtCreateTime attribute, which defines when this ENIEvent object instance was created. There are no input parameters to this operation. If an attribute value is NOT found, then an error **shall** be returned..
setEVTCreateTime(in newTime : TimeAndDate[1..1])	This operation sets the value of the evtCreateTime attribute, which defines when this ENIEvent object instance was created.
getEVTIsExternal() : Boolean[1..1]	This operation returns the value of the evtIsExternal attribute, which defines whether this ENIEvent object originated from a source external to the ENI System. There are no input parameters to this operation. If an attribute value is NOT found, then an error **shall** be returned.
setEVTIsExternal (in isExternal : Boolean[1..1])	This operation sets the value of the evtIsExternal attribute, which defines whether this ENIEvent object originated from a source external to the ENI System.
getEVTIsSynthetic() : Boolean[1..1]	This operation returns the value of the evtIsExternal attribute, which defines whether this ENIEvent object originated from a source external to the ENI System. There are no input parameters to this operation. If an attribute value is NOT found, then an error **shall** be returned.
setEVTIsSynthetic (in isSynthetic : Boolean[1..1])	This operation sets the value of the evtIsExternal attribute, which defines whether this ENIEvent object originated from a source external to the ENI System.
getEVTStatus : ENIEvtEventStatus[1..1]	This operation returns the value of the evtStatus attribute, which defines the status of this ENIEvent instance (e.g. whether it has been received and processed). If an attribute value is NOT found, then an error **shall** be returned.

Operation Name	Description
setEVTStatus (in newStatus : ENIEvtEventStatus[1..1])	This operation sets the value of the evtStatus attribute, which defines the status of this ENIEvent instance (e.g. whether it has been received and processed). This operation takes a single input parameter, called newStatus, which is an ENIEvtEventStatus datatype.
getENIEVTTTL() : Integer[1..1]	This operation returns the value of the eniEVTTTL attribute. If an attribute value is NOT found, then an error **shall** be returned.
setENIEVTTTL(in newVal: Integer[1..1])	This operation sets the value of the eniEVTTTL attribute. This operation takes a single input parameter, called newVal, which is an integer attribute.

5.2.3.3.5 Relationship Definition

This class participates in a single aggregation, called ENIEventHasEvent. See clause 5.2.3.3.6.3. At this time, no relationships are defined for this class.

5.2.3.3.6 ENIEvent Subclasses

5.2.3.3.6.1 Introduction

Figure 5-14 provides an overview of the three current ENIEvent subclass attributes and operations.

NOTE: Figure 5-14 may be augmented with additional subclasses in the future.

5.2.3.3.6.2 ENIEventAtomic

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

This object **shall not** contain another ENIEvent object.

Table 5-41 defines the attributes of this class.

Table 5-41: Attributes of the ENIEventAtomic Class

Attribute Name	Description
eniEVTIsHeaderEditable : Boolean[0..1]	This is an optional Boolean attribute. If the value of this attribute is TRUE, then the header of this ENIEvent is editable. Otherwise, it is not.
eniEVTHasAttachmentList: EByteArray[0..*]	This is an optional list of arrays that serve as a set of attachments to this event.
eniEVTPayloadDataList: EByteArray [0..*]	This is an optional list of arrays that collectively serve as a payload of this event.
eniEVTPayloadEnrichDataList: EByteArray [0..*]	This is an optional list of arrays that serve as a set of enriched data for the payload of this event.
eniEVTPersistentStorage: Boolean[0..1]	This is an optional Boolean attribute. If the value of this attribute is TRUE, then this ENIEvent object instance will be persistently stored.

Table 5-43 defines the operations of this class.

Table 5-43: Operations of the ENIEventAtomic Class

Operation Name	Description
getENIEVTIsHeaderEditable() : Boolean[1..1]	This operation returns the value of the eniEVTIsHeaderEditable attribute. There are no input parameters to this operation.
setENIEVTIsHeaderEditable (in newVal : Boolean[1..1])	This operation sets the value of the eniEVTIsHeaderEditable attribute. This operation takes a single input parameter, called newVal, which is a Boolean attribute.
getENIEVTHasAttachmentList() : EByteArray[1..*]	This operation returns the value of the eniEVTHasAttachmentList attribute. There are no input parameters to this operation. If an attribute value is NOT found, then an error **shall** be returned.
setENIEVTHasAttachmentList(in newVal : EByteArray[1..*])	This operation sets the value of the eniEVTHasAttachmentList attribute. This operation takes a single input parameter, called newVal, which is an EByteArray attribute.

Operation Name	Description
getENIEVTPayloadDataList() : EByteArray[1..*]	This operation returns the value of the eniEvtPayloadList attribute. There are no input parameters to this operation. If an attribute value is NOT found, then an error **shall** be returned.
setENIEVTPayloadDataList(in newVal : EByteArray[1..*])	This operation sets the value of the eniEvtPayloadList attribute. This operation takes a single input parameter, called newVal, which is an EByteArray attribute.
getENIEVTPayloadEnrichDataList() : EByteArray[1..*]	This operation returns the value of the eniEVTPayloadEnrich attribute. There are no input parameters to this operation. If an attribute value is NOT found, then an error **shall** be returned.
setENIEVTPayloadEnrichDataList(in newVal : EByteArray[1..*])	This operation sets the value of the eniEVTPayloadEnrich attribute. This operation takes a single input parameter, called newVal, which is an EByteArray attribute.
getENIEVTPersistentStorage : Boolean[1..1]	This operation returns the value of the eniEVTPersistentStorage attribute. There are no input parameters to this operation. If an attribute value is NOT found, then an error **shall** be returned.
setENIEVTPersistentStorage(in newVal : Boolean[1..1])	This operation sets the value of the eniEVTPersistentStorage attribute. This operation takes a single input parameter, called newVal, which is a Boolean attribute.
getEVTParentAtomic : ENIEventComposite[1..1]	This operation retrieves the ENIEventComposite object that contains this ENIEventAtomic object. This operation takes no input parameters. If this ENIEventAtomic object has no containing ENIEventComposite object, then it **should** return a NULL ENIEventComposite object. An ENIEventAtomic object **shall not** have more than one parent ENIEventComposite object.
setEVTParentAtomic : (in newParent : ENIEventComposite[1..1])	This operation sets the parent of this ENIEventAtomic object. This operation takes a single input parameter, called newParent, which is an ENIEventComposite object.
delEVTParentAtomic()	This operation deletes the inheritance relationship of this ENIEventAtomic object. There are no input parameters to this operation.

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

5.2.3.3.6.3 ENIEventComposite

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

Its primary use is to collect other types of ENIEvent objects. Typically, these other ENIEvent objects are related in some way. For example, an ENIEventComposite object may aggregate other ENIEventComposite and ENIEventAtomic objects for delivery to one or more MCMManagedEntity objects.

Each ENIEventComposite object **may** contain zero or more ENIEventAtomic and/or zero or more ENIEventComposite objects.

Table 5-43 defines the attributes of this class.

Table 5-43: Attributes of the ENIEventComposite Class

Attribute Name	Description
eniEVTNoDuplicate : Boolean[0..1]	This is an optional Boolean attribute. If the value of this attribute is TRUE, then this ENIEventComposite object **shall not** contain duplicate ENIEvent objects.

Table 5-44 defines the operations of this class.

Table 5-44: Operations of the ENIEventComposite Class

Operation Name	Description
getENIEVTNoDuplicate() : Boolean[1..1]	This operation returns the value of the eniEVTNoDuplicate attribute. If the value of this attribute is TRUE, then no duplicate ENIEvents are allowed to be contained in this ENIEventComposite object.
setENIEVTNoDuplicate (in newVal : Boolean [1..1])	This operation sets the value of the eniEVTNoDuplicate attribute. This operation takes a single input parameter, called newVal, which is a Boolean attribute.

Operation Name	Description
getENIEVTParentComposite() : ENIEventComposite[1..1]	This operation retrieves the parent of this ENIEventComposite object. This operation takes no input parameters. If this ENIEventComposite object is not contained by an ENIEventComposite object, then it **should** return a NULL ENIEventComposite object. An ENIEventComposite object **shall not** have more than one parent ENIEventComposite object.
setENIEVTParentComposite (in newParent : ENIEventComposite[1..1])	This operation defines a new ENIEventComposite to contain this particular ENIEvent object. This operation takes a single input parameter, called newParent, which is an ENIEventComposite object. If this ENIEvent object already has a parent ENIEventComposite object, then this ENIEventComposite object will be deleted by first, deleting the accompanying association class, and second, deleting the corresponding aggregation. Then, a new aggregation (an instance of ENIEventHasEvent) is created; following that, a new association class is then created to realize the semantics of the aggregation. This ENIEvent object **shall not** have more than one parent.
delENIEVTParentComposite()	This operation deletes the parent of this ENIEventComposite object. If another parent is found, then an error **shall** be returned.

This class defines a single optional aggregation, called ENIEventHasEvent. This is an optional aggregation, and defines the set of ENIEventAtomic and/or ENIEventComposite objects that are contained in an ENIEventComposite object. The semantics of this aggregation are defined by the ENIEventHasEventDetail association class.

The multiplicity of this aggregation is 0..1 - 0..n. If this aggregation is instantiated (i.e. the "1" part of the 0..1 cardinality), then zero or more ENIEventAtomic and/or ENIEventComposite objects may be contained in this particular ENIEventComposite object. The 0..* cardinality enables an ENIEventAtomic or an ENIEventComposite object to be defined without having to define a containing ENIEventComposite object.

The ENIEventHasEventDetail object is a concrete association class, and defines the semantics of the ENIEventHasEvent aggregation. The attributes and relationships of this class can be used to define which ENIEventAtomic and/or ENIEventComposite object can be contained in this particular ENIEventComposite object. These semantics can be further enhanced by using the Policy Pattern to define policy rules that constrain which ENIEventAtomic and/or ENIEventComposite objects can be contained in this particular ENIEventComposite object. The MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

5.2.3.3.6.4 ENIEventDecorator

This is a mandatory abstract class, and is used to implement the decorator pattern for ENIEvents. This means that any concrete subclass of ENIEventDecorator can wrap any concrete subclass of ENIEventAtomic and/or ENIEventComposite.

The Decorator Pattern is a design pattern that allows behaviour to dynamically be added to an object, without affecting the behaviour of other objects from the same class. More specifically, this pattern enables all or part of one object to wrap another object. In effect, this means that the decorated object may intercept a call to the object it is wrapping, and insert attributes or execute operations before and/or after the wrapped object executes.

Hence, the decorator pattern provides a flexible alternative to subclassing for extending functionality where different behaviours are required (e.g. dependent on context).

In addition, subclassing statically defines the characteristics and behaviour of an object at compile time, whereas the decorator pattern can change the characteristics and behaviour of an object at runtime

A concrete subclass of the ENIEventDecorator class **may** be used to decorate any concrete subclass of the ENIEventAtomic or ENIEventComposite classes.

Table 5-45 defines the attributes of this class.

Table 5-45: Attributes of the ENIEventDecorator Class

Attribute Name	Description
eniEVTDecConstraint : MPMPolCompDecConstraint[1..1]	This is a mandatory non-negative enumerated integer attribute that defines the language used, if any, that this ENIEventDecorator object uses to constrain the object that it is wrapping. Valid values are defined by the MPMPolCompDecConstraint enumeration. A default value of 2 (NONE) **may** be defined.
eniEVTDecWrap: MPMPolCompDecWrap[1..1]	This is an optional attribute that defines if this decorated object should be wrapped before and/or after the wrapped object is executed. Valid values are defined by the MPMPolCompDecWrap enumeration.

Table 5-46 defines the operations of this class.

Table 5-46: Operations of the ENIEventAtomic Class

Operation Name	Description
getENIEVTDecConstraint() : MPMPolCompDecConstraint[1..1]	This operation returns the current value of the eniEVTDecConstraint attribute. This operation takes no input parameters. If this attribute does not have a value, then this operation **should** return an error, because NONE is one of the enumerated values of MPMPolCompDecConstraint.
setENIEVTDecConstraint(in newVal : MPMPolCompDecConstraint[1..1])	This operation sets the value of the eniEVTDecConstraint attribute. This operation takes a single input parameter, called newVal, which defines the new value for the eniEVTDecConstraint attribute.
getENIEVTDecWrap() : MPMPolCompDecWrap[1..1]	This operation returns the current value of the eniEVTDecWrap attribute. This operation takes no input parameters. If this attribute does not have a value, then this operation **should** return an error.
setENIEVTDecWrap(in newVal : MPMPolCompDecWrap[1..1])	This operation sets the value of the eniEVTDecWrap attribute. This operation takes a single input parameter, called newValue, which defines the new value for the eniEVTDecWrap attribute.

This class defines a single optional aggregation, called ENIEventHasDecorator. This is an optional aggregation, and defines the set of ENIEventDecorator objects that wrap, or decorate, an ENIEventAtomic or an ENIEventComposite object. The semantics of this aggregation are defined by the ENIEventHasDecoratorDetail association class.

The attachment of different ENIEventDecorator objects shall be used to change the syntax, semantics, and/or behaviour of a given ENIEventAtomic or an ENIEventComposite object.

The multiplicity of this aggregation is 0..1 - 0..n. If this aggregation is instantiated (i.e. the "1" part of the 0..1 cardinality), then zero or more ENIEventDecorator objects can decorate an ENIEventAtomic or an ENIEventComposite object. The 0..* cardinality enables an ENIEventAtomic or an ENIEventComposite object to be defined without having to define an associated ENIEventDecorator object for it to decorate.

The ENIEventHasDecoratorDetail object is a concrete association class, and defines the semantics of the ENIEventHasDecorator aggregation. The attributes and relationships of this class can be used to define which ENIEventDecorator objects can decorate an ENIEventAtomic or an ENIEventComposite object. These semantics can be further enhanced by using the Policy Pattern to define policy rules that constrain which part objects (i.e. ENIEventDecorator) are attached to which object. The MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

5.2.3.4 Behaviour

5.2.3.4.1 Introduction

The purpose of this class hierarchy is to define a common management approach to represent the behaviour of MCMManagedEntity and MCMParty objects. More specifically, objects in this class hierarchy shall be able to represent how a MCMManagedEntity or an MCMParty functions. In this definition, the term "functions" shall include the following two different meanings:

1) Operation:

   a) For MCMManagedEntity objects, this should include defining how the object operates. This may include defining appropriate Role objects that specifically provide the desired behaviour.

   b) For MCMParty objects, this includes the set of responsibilities that an object has. This should be realized using appropriate Role objects.

2) Reaction to Events:

   a) For MCMManagedEntity objects, this includes how the object reacts to different stimuli in different situations.

   b) For MCMParty objects, this includes defining the set of Role objects that are used to define the reaction of the object in response to different stimuli in different situations.

The above is reflected in Figure 5-15.

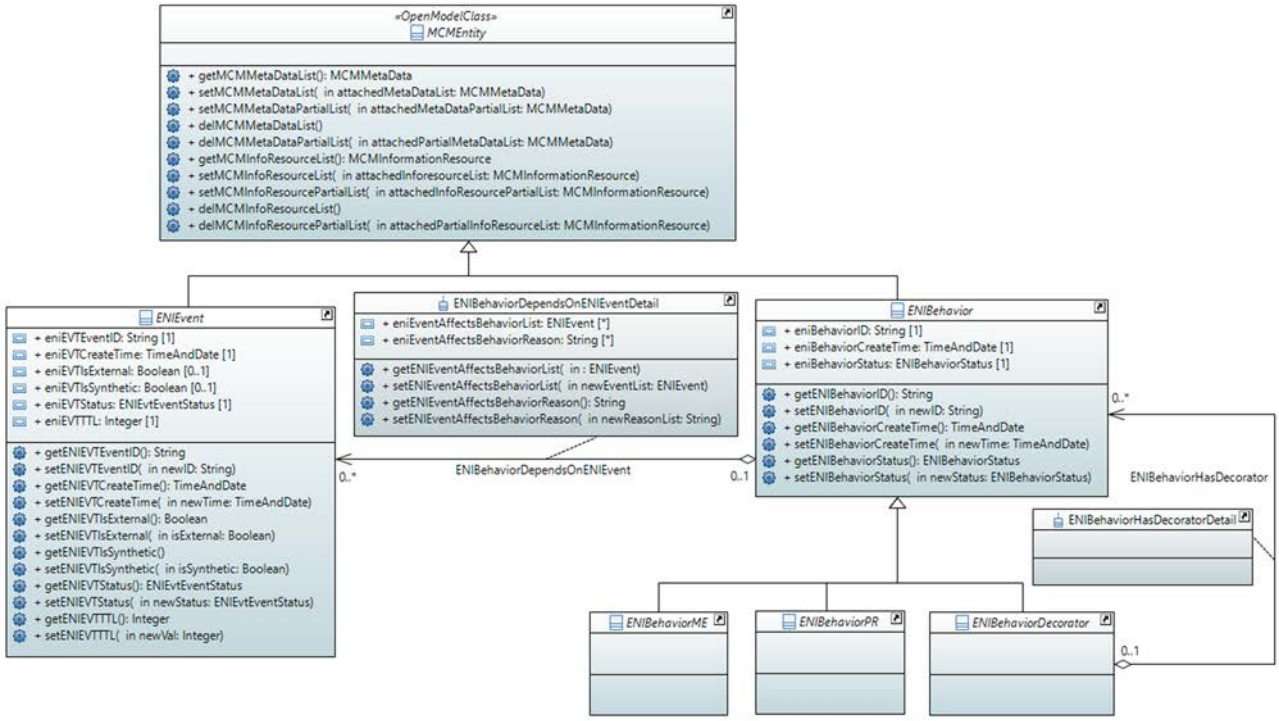

Figure 5-15: Overview of the ENIBehavior Class Hierarchy

This class hierarchy should also used to orchestrate and/or choreograph the behaviour of one or more MCMManagedEntity and/or MCMParty objects.

The classes in this hierarchy do not include information such as the time and date that this MCMManagedEntity was created; this is because this information exists in the Version subclass of MetaData, which is related to this class through the EntityHasMetaData aggregation.

5.2.3.4.2 ENIBehavior Class Definition

ENIBehavior is a mandatory abstract class that extends the Entity class to represent how an MCMManagedEntity or an MCMParty object functions and reacts to Events in a given context.

5.2.3.4.3 Attribute Definition

Table 5-45 defines the attributes of this class. An ENIBehavior object may be defined as valid for a particular time period; this would be done using attached MCMMetaData. Similarly, a location may be associated with an ENIBehavior; this could be defined using an explicit association between MCMLocation and ENIBehavior.

Table 5-47: Attributes of the ENIEvent Class

Attribute Name	Description
eniBehaviorID: String[1..1]	This is a mandatory string attribute, and contains a unique value that enables instances of this ENIBehavior object instance to be disambiguated from all other ENIBehavior object instances (including ENIBehavior objects of the same type). The value of this attribute **shall not** be a NULL or EMPTY string.
eniBehaviorCreateTime: TimeandDate[1..1]	This is a mandatory TimeAndDate attribute that contains the date and time that this ENIBehavior object was created.
eniBehaviorStatus: ENIBehaviorStatus[1..1]	This is a mandatory enumerated string attribute that contains the status information of this ENIBehavior object (e.g. processed, ignored). Allowed values are defined in the ENIBehaviorStatus enumeration.

5.2.3.4.4 Operation Definition

Table 5-48 defines the operations of this class.

Table 5-48: Operations of the ENIEventAtomic Class

Operation Name	Description
getENIBehaviorID() : String[1..1]	This operation returns the current value of the eniBehaviorID attribute. This operation takes no input parameters. If this attribute does not have a value, then an error **shall** be returned. The value of this attribute **shall not** be an empty or NULL string.
setENIBehaviorID(in newID : String[1..1])	This operation sets the value of the eniBehaviorID attribute, which is a textual identifier to disambiguate this ENIBehavior object instance from all other ENIBehavior object instances. This operation takes a single input parameter, called newID, which is a String attribute. The value of this attribute **shall not** be an empty or NULL string.
getENIBehaviorCreateTime() : TimeAndDate[1..1]	This operation returns the eniBehaviorCreateTime attribute, which defines when this ENIBehavior object instance was created. There are no input parameters to this operation. If an attribute value is NOT set, then an error **shall** be returned.
setENIBehaviorCreateTime(in newTime : TimeAndDate[1..1])	This operation sets the value of the eniBehaviorCreateTime attribute, which defines when this ENIBehavior object instance was created. This operation takes a single input parameter, called newTime, which is a String attribute.
getENIBehaviorStatus() : ENIBehaviorStatus[1..1]	This operation returns the eniBehaviorStatus attribute, which defines the status of this ENIBehavior object instance (e.g. whether it has been received and processed). There are no input parameters to this operation. If an attribute value is NOT set, then an error **shall** be returned.
setENIBehaviorStatus(in newStatus: ENIBehaviorStatus[1..1])	This operation sets the value of the eniBehaviorStatus attribute, which defines the status of this ENIBehavior object instance (e.g. whether it has been received and processed). This operation takes a single input parameter, called newStatus, which is an ENIBehaviorStatus datatype.

5.2.3.4.5 Relationship Definition

This class defines a single optional aggregation, called ENIBehaviorDependsOnEvent. This is an optional aggregation, and defines the set of ENIEvent objects that affect the behavior of this particular ENIBehavior object. The semantics of this aggregation are defined by the ENIBehaviorDependsOnEventDetail association class.

The multiplicity of this aggregation is 0..1 - 0..*. If this aggregation is instantiated (i.e. the "1" part of the 0..1 cardinality), then zero or more ENIEvent objects affect this particular ENIBehavior object. The 0..* cardinality enables an ENIEvent object to be defined without having to define an associated ENIBehavior object.

The ENIBehaviorDependsOnEventDetail object is a concrete association class, and defines the semantics of the ENIBehaviorDependsOnEvent aggregation. The attributes and relationships of this class can be used to define which ENIEvent objects can affect this particular ENIBehavior object. These semantics can be further enhanced by using the Policy Pattern to define policy rules that constrain which part objects (i.e. ENIEvent) affect which aggregate (i.e. ENIBehavior) object. The MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

This association class defines the following attributes.

Table 5-49: Attributes of the ENIBehaviorDependsOnENIEventDetail Class

Attribute Name	Description
eniEventAffectsBehaviorList : ENIEvent[0..*]	This attribute defines an array of ENIEvent objects that are permitted to affect the behavior of this particular ENIBehavior object.
eniEventAffectsBehaviorReason : String[0..*]	This attribute defines an array of strings that describe why each ENIEvent in the eniEventAffectsBehaviorList attribute can affect the behavior of this particular ENIBehavior object.

This association class defines the following operations.

Table 5-50: Attributes of the ENIBehaviorDependsOnENIEventDetail Class

Attribute Name	Description
getENIEventAffectsBehaviorList: ENIEvent[0..*]	This operation returns the set of ENIEvent objects that can affect this particular ENIBehavior object. This operation takes no input parameters. If there are no ENIEvent objects that can affect this particular ENIBehavior object, then a NULL ENIEvent object **should** be returned.
getENIEventAffectsBehaviorList: String[0..*]	This operation defines a new set of ENIEvent objects that affect the operation of this particular ENIBehavior object. This operation takes a single input parameter, called newEventList, which defines a set of one or more ENIEvent objects. If one or more ENIEvent objects are already associated with this particular ENIBehavior object, then those ENIEvent objects will be deleted by first, deleting the accompanying association class, and second, deleting the corresponding association. Then, a new association (an instance of ENIBehaviorDependsOnENIEventDetail) is created for each ENIEvent object in the newEventList. Every association created **should** have a new corresponding association class to realize the semantics of that association.
getENIEventAffectsBehavior-Reason() : String[0..*]	This operation returns the set of reasons, which are textual descriptions, of why each ENIEvent object affects this particular ENIBehavior object. This operation takes no input parameters. If there are no ENIEvent objects that can affect this particular ENIBehavior object, then a NULL String object **should** be returned.
setENIEventAffectsBehavior-Reason(in newReasonList : String[1..*])	This operation defines the set of reasons, which are textual descriptions, of why each ENIEvent object affects this particular ENIBehavior object. This operation takes a single input parameter, called newReasonList, which defines a set of one or more Strings, each containing a description of the reason corresponding to a particular ENIEvent.

5.2.3.4.6 ENIBehavior Subclasses

5.2.3.4.6.1 Introduction

Figure 5-15 provides an overview of the three current ENIBehavior subclass attributes and operations.

NOTE: Figure 5-15 may be augmented with additional subclasses in the future.

5.2.3.4.6.2 ENIBehaviorME

This is a mandatory abstract class that defines the behavior of MCMManagedEntity objects. The behavior of an MCMManagedEntity object can be affected in two different ways:

1) Operation:

   - This defines how the object operates. This may include defining appropriate Role objects that specifically provide the desired behavior.

2) Reaction to Events:

   - This includes how the object reacts to different stimuli in different situations.

The operation is best expressed by defining operations on the object in conjunction with using an ENIFiniteStateMachine. The reaction to Events is best expressed by the above in combination with Activity and/or Interaction diagrams.

5.2.3.4.6.3 ENIBehaviorPR

This is a mandatory abstract class that defines the behavior of MCMPartyRole objects. The behavior of an MCMPartyRle object can be affected in two different ways:

1) Operation:

   - This defines the set of responsibilities that an object has. This should be realized using appropriate Role objects. Roles are context-dependent.

2) Reaction to Events:

   - This defines how the object reacts to different stimuli in different situations.

The operation is best expressed by defining operations on the object in conjunction with using an ENIFiniteStateMachine. The reaction to Events is best expressed by the above in combination with Activity and/or Interaction diagrams.

5.2.3.4.6.4 ENIBehaviorDecorator

This is a mandatory abstract class, and is used to implement the decorator pattern for ENIBehavior objects. This means that any concrete subclass of ENIEventDecorator can wrap any concrete subclass of ENIBehaviorME and/or ENIBehaviorPR.

The Decorator Pattern is a design pattern that allows behaviour to dynamically be added to an object, without affecting the behaviour of other objects from the same class. More specifically, this pattern enables all or part of one object to wrap another object. In effect, this means that the decorated object may intercept a call to the object it is wrapping, and insert attributes or execute operations before and/or after the wrapped object executes.

Hence, the decorator pattern provides a flexible alternative to subclassing for extending functionality where different behaviours are required (e.g. dependent on context). In addition, subclassing statically defines the characteristics and behaviour of an object at compile time, whereas the decorator pattern can change the characteristics and behaviour of an object at runtime.

A concrete subclass of the ENIEventDecorator class may be used to decorate any concrete subclass of the ENIEventAtomic or ENIEventComposite classes.

This class currently does not define any attributes or operations.

This class defines a single aggregation, called ENIBehaviorHasDecorator. This is an optional aggregation, and defines the set of ENIBehaviorDecorator objects that wrap, or decorate, an ENIBehavior object. The semantics of this aggregation are defined by the ENIBehaviorHasDecoratorDetail association class.

The attachment of different ENIBehaviorDecorator objects shall be used to change the syntax, semantics, and/or behaviour of a given ENIBehavior object. The multiplicity of this aggregation is 0..1 - 0..n. If this aggregation is instantiated (i.e. the "1" part of the 0..1 cardinality), then zero or more ENIBehaviorDecorator objects can decorate any concrete subclass of the ENIBehavior object. The 0..* cardinality enables concrete subclass of the ENIBehavior class to be defined without having to define an associated ENIBehaviorDecorator object for it to decorate.

The ENIBehaviorHasDecoratorDetail object is a concrete association class, and defines the semantics of the ENIBehaviorHasDecorator aggregation. The attributes and relationships of this class can be used to define which ENIBehaviorHasDecorator objects can wrap any concrete subclass of the ENIBehavior object. These semantics can be further enhanced by using the Policy Pattern to define policy rules that constrain which part objects (i.e. ENIEventDecorator) are attached to which object. The MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

5.2.3.5 Identity

5.2.3.5.1 Introduction

The identity of an object is the set of data and information that allow an object to be disambiguated from all other objects in a system, including objects of the same type.

A digital identity is the set of data and information used by a computer system to represent an actor, such as a person, device, or application.

A *contextual identity* is the digital identity of an object for a particular context. For example, a person may have a combination of username and password to log onto a system as well as a passport to enable that person to fly. Each identity in this example is specific to a particular context in which it is used.

An ENI System **shall** use digital identities for all objects in its information and data models.

An ENI System **should** use *contextual identities* where possible.

Figure 5-16 shows the ENIIdentity class hierarchy.

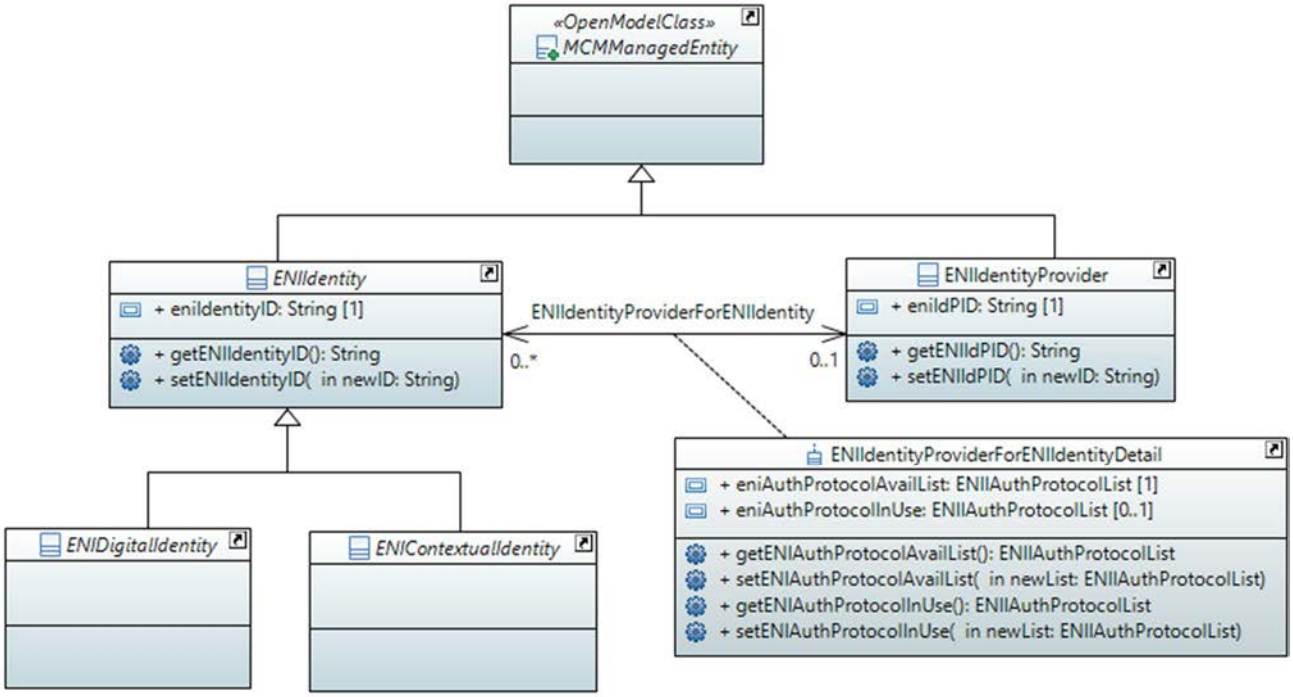

Figure 5-16: Overview of the ENIIdentity Class Hierarchy

5.2.3.5.2 ENIIdentity Class Definition

An ENIIdentity is a mandatory abstract class that defines a set of characteristics and behaviours by which an object **shall** be uniquely recognized and disambiguated from all other objects in the system, including objects of the same type.

From a security point-of-view, an Identity is the set of characteristics by which a Subject or Target Entity is recognizable and that, within the scope of an Identity Provider's responsibility, is sufficient to uniquely disambiguate an instance of that Actor from an instance of any other Actor [9].

5.2.3.5.3 Attribute Definition

Table 5-51 defines the operations of this class.

Table 5-51: Attributes of the ENIIdentity Class

Attribute Name	Description
eniIdentityID: String[1..1]	This is a mandatory string attribute, and contains a unique value that enables instances of this ENIIdentity object to be disambiguated from other ENIIdentity objects (including ENIIdentity objects of the same type). The value of this attribute **shall not** be a NULL or EMPTY string.

5.2.3.5.4 Operation Definition

Table 5-52 defines the operations of this class.

Table 5-52: Operations of the ENIIdentity Class

Operation Name	Description
getENIIdentityID() : String[1..1]	This operation returns the current value of the eniIdentityID attribute. This operation takes no input parameters. If this attribute does not have a value, then an error **shall** be returned. The value of this attribute **shall not** be an empty or NULL string.
setENIIdentityID(in newID : String[1..1])	This operation sets the value of the eniIdentityID attribute, which is a textual identifier to disambiguate this ENIIdentity object instance from all other ENIIdentity object instances. This operation takes a single input parameter, called newID, which is a String attribute. The value of this attribute **shall not** be an empty or NULL string.

5.2.3.5.5 Relationship Definition

This class participates in a single relationship, called ENIIdentityProviderOfENIIdentity. This association is defined in clause 5.2.3.5.9.

5.2.3.5.6 ENIIdentityProvider Class Definition

An ENI Identity Provider is mandatory abstract class. It defines an organization that manages the Authentication Credentials of an Actor.

5.2.3.5.7 Attribute Definition

Table 5-53 defines the operations of this class.

Table 5-53: Attributes of the ENIIdentity Class

Attribute Name	Description
eniIdPID: String[1..1]	This is a mandatory string attribute, and contains a unique value that enables instances of this ENIIdentityProvider object to be disambiguated from other ENIIdentityProvider objects (including ENIIdentityProvider objects of the same type). The value of this attribute **shall not** be a NULL or EMPTY string.

5.2.3.5.8 Operation Definition

Table 5-54 defines the operations of this class.

Table 5-54: Operations of the ENIIdentity Class

Operation Name	Description
getENIIdPID() : String[1..1]	This operation returns the current value of the eniIdPID attribute. This operation takes no input parameters. If this attribute does not have a value, then an error **shall** be returned. The value of this attribute **shall not** be an empty or NULL string.
setENIIdPID (in newID : String[1..1])	This operation sets the value of the eniIdPID attribute, which is a textual identifier to disambiguate this ENIIdentityProvider object instance from all other ENIIdentityProvider object instances. This operation takes a single input parameter, called newID, which is a String attribute. The value of this attribute **shall not** be an empty or NULL string.

5.2.3.5.9 Relationship Definition

This class defines a single optional association, called ENIIdentityProviderForENIIdentity that defines the particular ENIIdentityProvider that authenticates this particular ENIIdentity object. The semantics of this association are defined by the ENIIdentityProviderForENIIdentityDetail association class.

The multiplicity of this association is 0..1 - 0..*. If this association is instantiated (i.e. the "1" part of the 0..1 cardinality), then one ENIIdentityProvider object is defined to authenticate one or more particular ENIIdentity object. The 0..* cardinality enables an ENIIdentity object to be defined without having to define an associated ENIIdentityProvider object.

The ENIIdentityProviderForENIIdentityDetail object is a concrete association class, and defines the semantics of the ENIIdentityProviderForENIIdentity association. The attributes and relationships of this class can be used to define which ENIIdentityProvider object can be used to authenticate this set of ENIIdentity objects. These semantics can be further enhanced by using the Policy Pattern to define policy rules that constrain which part objects (i.e. ENIIdentity) by which specific aggregate (i.e. ENIIdentityProvider) object. The MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

This association class defines the following attributes.

Table 5-55: Attributes of the ENIIdentityProviderForENIIdentityDetail Class

Attribute Name	Description
eniAuthProtocolAvailList: ENIAuthProtocolList[1..1]	This attribute defines an approved set of authentication protocols to use by an ENIIdentityProvider to authenticate an ENIIdentity object.
eniAuthProtocolInUse: ENIAuthProtocolList[1..1]	This attribute defines the current authentication protocol that is used to authenticate this particular ENIIdentity object by this specific ENIIdentityProvider.

This association class defines the following operations.

Table 5-56: Attributes of the ENIIdentityProviderForENIIdentityDetail Class

Attribute Name	Description
getENIAuthProtocolAvailList: ENIAuthProtocolList [0..*]	This operation returns the set of approved authentication protocols that can be used by an ENIIdentityProvider to authenticate an ENIIdentity object. This operation takes no input parameters.
setENIAuthProtocolAvailList(in newList : ENIAuthProtocolList [1..1])	This operation defines a new set of approved authentication protocols that can be used by an ENIIdentityProvider to authenticate an ENIIdentity object. This operation takes a single input parameter, called newList, which defines a set of one or more authentication protocols.
getENIAuthProtocolInUse: ENIAuthProtocolList [1..1]	This operation returns the current authentication protocol that is being used to authenticate this particular ENIIdentity object by this ENIIdentityProvider object. This operation takes no input parameters.
getENIAuthProtocolInUse(in newList : ENIAuthProtocolList [1..1])	This operation defines a new authentication protocol that is being used to authenticate this particular ENIIdentity object by this ENIIdentityProvider object. This operation takes a single input parameter, called newList, which defines an approved authentication protocol to use.

5.2.3.5.10 ENIIdentity Subclasses

5.2.3.5.10.1 Introduction

Figure 5-16 provides an overview of the three current ENIIdentity subclass attributes and operations.

5.2.3.5.10.2 ENIDigitalIdentity

An ENIDigitalIdentity is a mandatory abstract class. It contains the set of data and information used by a computer system to represent an actor, such as a person, device, or application.

An ENI System **shall** use digital identities for all objects in its information and data models.

This class, and its subclasses, are for further study in Release 4 of the present document.

5.2.3.5.10.3 ENIContextualIdentity

An ENIContextualIdentity is a mandatory abstract class. It contains the digital identity of an object for a particular context.

EXAMPLE: A person may have a combination of username and password to log onto a system as well as a passport to enable that person to fly. Each identity in this example is specific to a particular context in which it is used.

An ENI System **should** use contextual identities where possible.

This class, and its subclasses, are for further study in Release 4 of the present document.

5.2.4 ENI Extended Core Model

The ENI Extended Core Model **shall** be based on the MCM classes, attributes, operations, and relationships defined in the present document. In addition, three class hierarchies (i.e. ENIEvent, ENIBehaviour, and ENIIdentity and their respective subclasses) are defined by ETSI ISG ENI that extend the MCM. These three class hierarchies **should** be used to define the ENI Extended Core Model.

Additional classes defined by ETSI ISG ENI are for further study in Release 4 of the present document.

5.3 Models that Inherit from the ENI Extended Core Model

5.3.1 Introduction

The MCM has a number of models that refine it. Each of these models is an extension (refinement) of a concept defined in the MCM. The Policy model is one such model, as it refines the MCMPolicyObject class (which is a type of MCMManagedEntity).

5.3.2 Policy Model

5.3.2.1 Introduction

The present document **shall** use UML (Unified Modeling Language) [6] to describe the salient characteristics and behaviour of ENI Policies. ENI Policies **may** take the following forms: imperative, declarative, and intent policies. ENI Policies **may** contain multiple types of policies (e.g. an intent policy may invoke an imperative policy). The present document defines how an ENI Policy can affect the behaviour of entities that are important to the managed environment. In particular, it defines an information model for describing ENI Policies.

[i.2] describes policy management in detail, and is a good background for understanding the present document.

5.3.2.2 Purpose

Management involves monitoring the activity of a system, making decisions about how the system is acting, and performing control actions to modify the behaviour of the system. The purpose of policy is to ensure that consistent decisions are made governing the behaviour of a system. Consistency shall be provided by using a common mechanism to construct and format recommendations and commands, as well as a standard mechanism to deliver ENI Policies.

The definition of policy is in ETSI GS ENI 005 [3]:

- a set of rules that is used to manage and control the changing; and/or
- maintaining of the state of one or more managed objects.

Hence, policies are one mechanism for *controlling the behaviour* of an Entity. Two important types of Policies are authorization and obligation policies. Authorization policies define what the target of a policy is permitted or not permitted to do. Obligation policies define what the management engine "shall" or "shall not" do and hence, guide the decision-making process. These two types of Policies are based on deontic logic (https://plato.stanford.edu/entries/logic-deontic/).

NOTE: There are other types of policies that correspond to different types of logic (https://plato.stanford.edu/entries/logic-modal/). Examples include alethic logic (i.e. the logic of necessity and possibility), epistemic logic (i.e. the logic of certainty), temporal logic (i.e. logic that qualifies statements based on time). These are beyond the scope of the present document.

The different types of logics may be used to help translate a policy to a mathematical form that can be validated (i.e. proven to be correct). This validated form can then be used to disambiguate the (original) policy and to check for conflicts with other policies.

For further information on the MPM, see [5] and [7]. The rest of clause 5.3.2 and its subclauses summarise the salient information from [5] and [7] that are used to build the ENI Extended Policy Model.

5.3.2.3 Extensions to the PDO Model

The MEF PDO model (MPM) is defined in [5]. The MPM uses five important abstractions that collectively enable it to model multiple types of policies:

1) The first is the concept of a *policy container*. This means that any type of policy shall be structured as an object that is made up of policy components.

2) The second defines two fundamental types of objects, a policy (called MPMPolicyStructure) and a policy component (called MPMPolicyComponentStructure). Hence, any policy consists of one instance of MPMPolicyStructure and one or more instances of MPMPolicyComponentStructure.

3) Third, the content of any policy shall be made up of one or more statements (called MPMPolicyStatement).

4) Fourth, any MPMPolicyStatement may be made up of one or more clauses (called MPMPolicyClause).

5) Fifth, the type of policy shall determine the set of statements that it can contain. This is enforced by a novel software pattern.

In summary, there are two parallel hierarchies: one for types of policies, and one for policy components. The type of policy determines the allowed set of policy components that it can contain. Each policy component defines content that is either represented in a policy clause or a policy statement.

5.3.2.4 MEF Types of Policies

5.3.2.4.1 Introduction

There are three main types of policy that are used in the PDO:

- imperative;
- declarative; and
- intent policies.

The policy abstractions defined above enable other types of policies (e.g. utility functions) to be easily added in the future.

5.3.2.4.2 Imperative Policies

Imperative policies follow the imperative programming paradigm, which focuses on describing how a program operates. Conceptually, the policy author is the "compiler", since the policy author is responsible for controlling the transitioning of one state to another state. Specifically, there is only one target state that is allowed to be chosen.

MPM defines two types of imperative policies:

- commands and Event-Condition-Action (ECA) policies.

These are specified in clauses 5.3.2.6.4.6 and 5.3.2.6.4.7 of the present document.

5.3.2.4.3 Declarative Policies

Declarative policies describe what needs to be done without defining its implementation.

For the purposes of the present document, a declarative policy is defined as a program that executes according to a theory defined in a formal logic. Declarative policies are written in a formal logic language, such as First Order Logic.

This is contrasted with intent policies (see clause 5.3.2.4.4), which are written in a (controlled) natural language and then translated to a different form.

More information about declarative policies is specified in clause 5.3.2.6.4.

5.3.2.4.4 Intent Policies

An intent policy is a type of declarative policy. As such, it describes what needs to be done without defining its implementation. However, an intent policy is written in a controlled natural language, whereas a declarative policy is written in a formal logic language. As such, each statement in an intent policy may require the translation of one or more of its terms to a form that another managed functional entity can understand. More information about intent policies is specified in clause 5.3.2.6.4.5.

5.3.2.5 MEF Policy Model Naming Rules

The MPM uses the following naming rules, which are compatible with those used in the MCM (see clause 5.2.2.2). Naming rules for ENI-defined extensions of the MPM are defined in clause 5.3.2.5:

1) Class names shall be in PascalCase.

2) Class names shall not begin with any non-alphabetic character.

3) Each MPM class shall be prefixed with "MPM".

NOTE 1: This serves three purposes. First, it helps provide context to textual descriptions of these model elements. Second, it provides a namespace for MPM objects. Third, it enables MPM model elements, patterns, and approaches to be compared to those of other SDOs and consortia unambiguously.

4) Attribute names shall be in camelCase.

5) Attribute names shall not begin with any non-alphabetic character except for the underscore.

6) Each attribute shall be prefixed with "mpm".

7) Relationship names shall be in PascalCase.

8) Each relationship shall be prefixed with "MPM".

9) All association classes shall be suffixed with the word "Detail".

NOTE 2: This enables association classes to be easily identified and differentiated from regular classes.

10) All MPM 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 MPM) shall be prefixed with "MPMMEF".

11) All MPM classes that model a concept from another SDO *exactly as it is defined* in that SDO shall be prefixed with "MPM", followed by the name of the SDO, followed by the class name.

NOTE 3: For example, if an SDO named Foo defined a class named Bar, and MPM imported this concept with no changes, it would be named MPMFooBar.

5.3.2.6 MEF Policy Hierarchy

5.3.2.6.1 Overview

Figure 5-17 shows a simplified functional diagram of the MEF Policy Model, emphasizing the abstractions defined in clause 5.3.2.3.

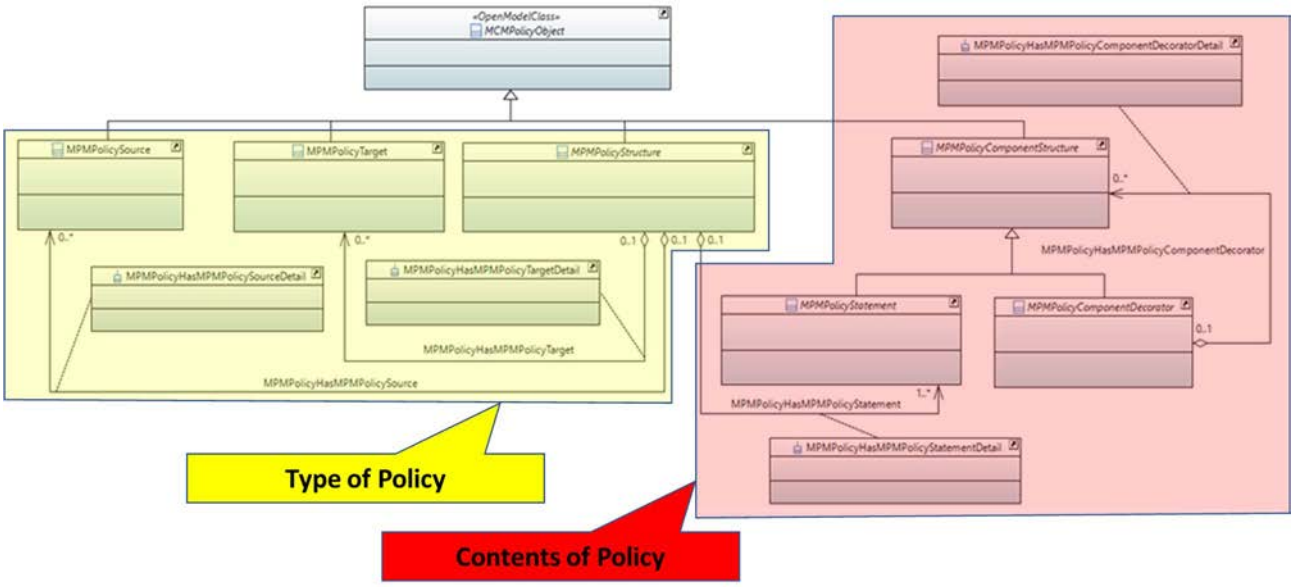

Figure 5-17: Simplified Functional Diagram of the MEF Policy Model

Different types of policies are represented by different subclasses of MPMPolicyStructure. Once a particular subclass of MPMPolicyStructure is chosen, this restricts the types of MPMPolicyStatements that can be used to define its content. The restriction should be implemented using a combination of the attributes, operations, and relationships of the MPMPolicyHasMPMPolicyStatementDetail association class. In addition, OCL may be used for formal specification of these restrictions.

Conceptually, the "left side" of Figure 5-17 represents the type of policy, and the "right side" represents the contents of the policy. When a given policy is defined on the left side, the set of components that can be used to populate its content are then defined on the right side.

The content of any policy is defined as a set of one or more statements. Any statement can be decorated by subclasses of the MPMPolicyComponentDecorator class.

5.3.2.6.2 MPMPolicyStructure Overview

Figure 5-18 shows the subclasses of the MPMPolicyStructure class.

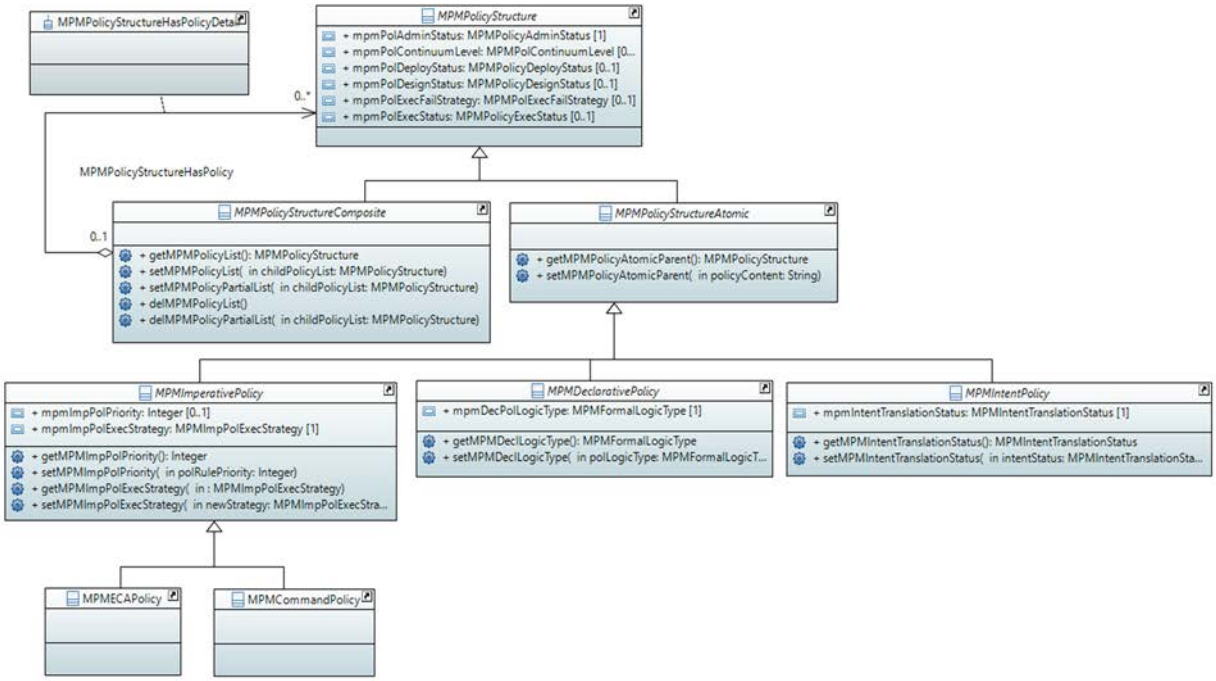

Figure 5-18: Subclasses of the MPMPolicyStructure Class

5.3.2.6.3 MPMPolicyStructure Class Definition

MPMPolicyStructure is a mandatory abstract class that defines the type of an MPMPolicy (i.e. imperative, declarative, or intent). The MPMPolicyStructure class is a type of PolicyContainer, which defines the set of MPMPolicyComponentStructure objects that it **may** contain. The following table defines the attributes of the MPMPolicyStructure class.

Table 5-57 defines the attributes for this class.

Table 5-57: Attributes of the MPMPolicyStructure Class

Attribute Name	Description
mpmPolAdminStatus : MPMPolicyAdminStatus[1..1]	This is a mandatory enumerated non-negative integer attribute that defines the current administrative status of this particular MPMPolicy object.
mpmPolContinuumLevel: MPMPolContinuumLevel[0..1]	This is an optional enumerated non-negative integer attribute that defines the level of abstraction, as represented by the Policy Continuum Level, of this particular MPMPolicy.
mpmPolDeployStatus : MPMPolicyDeployStatus[0..1]	This is an optional enumerated, non-negative integer attribute that indicates whether this MPMPolicy can or cannot be deployed by the policy management system. This attribute enables the policy manager to know which MPMPolicies are currently deployed, and may be useful for the policy execution system for planning the staging of MPMPolicies.
mpmPolDesignStatus : MPMPolicyDesignStatus[0..1]	This is an optional enumerated, non-negative integer whose value defines the current design status of this MPMPolicy object.
mpmPolExecFailStrategy: MPMPolExecFailStrategy[0..1]	This is an optional enumerated, non-negative integer attribute. It is used to define what actions, if any, should be taken by this particular MPMPolicy if it fails to execute correctly. Some systems may not be able to support all options specified in this enumeration. For example, if rollback is NOT supported by the system, then options 2 and 3 may be skipped, and options 4 and 5 may be used in their place. The allowable values of this enumeration are defined by the MPMPolExecFailStrategy enumeration.
mpmPolExecStatus: MPMPolicyExecStatus[0..1]	This is an optional enumerated non-negative enumerated integer whose value defines the current execution status of this MPMPolicy object.

Attribute Name	Description
mpmPolExecOrder: Integer[0..1]	This is an optional non-negative integer whose value defines the desired order of execution of this MPMPolicy Object (compared with all other MPMPolicy Objects). The MPMPolicy whose mpmPolExecOrder attribute has the highest value is executed first. A default value of 0 **may** be defined. MPMPolicies that have the same value for their mpmPolExecOrder attributes **may** be executed in any order.

Table 5-58 defines the operations for this class.

Table 5-58: Operations of the MPMPolicyStructure Class

Operation Name	Description
getMPMPolAdminStatus() : MPMPolicyAdminStatus[1..1]	This operation returns the current administrative status of this particular MPMPolicy object. If the mpmPolAdminStatus attribute does not have a value, then this operation **shall** return an error.
setMPMPolAdminStatus (in inputStatus : MPMPolicyAdminStatus[1..1])	This operation sets the value of the current administrative status of this particular MPMPolicy object.
getMPMPolContinuumLevel() : MPMPolContinuumLevel[1..1]	This operation returns the level of abstraction, as represented by the Policy Continuum Level, of this particular MPMPolicy. If the mpmPolContinuumLevel attribute does not have a value, then this operation **should** return a NULL string.
setMPMPolContinuumLevel(in polContinuumLevel : MPMPolContinuumLevel[1..1])	This operation sets the level of abstraction, as represented by the Policy Continuum Level, of this particular MPMPolicy.
getMPMPolDeployStatus() : MPMPolicyDeployStatus[1..1]	This operation returns the current deployment status of this particular MPMPolicy, which is defined by the MPMPolicyDeployStatus enumeration. If the mpmPolDeployStatus attribute does not have a value, then this operation **should** return a NULL string.
setMPMPolDeployStatus(in polDeployStatus : MPMPolicyDeployStatus[1..1])	This operation sets the current deployment status of this particular MPMPolicy.
getMPMPolDesignStatus() : MPMPolicyDesignStatus[1..1]	This operation returns the current design status of this particular MPMPolicy object, which is defined by the MPMPolicyDesignStatus enumeration. If the mpmPolDesignStatus attribute does not have a value, then this operation **shall** return an error.
setMPMPolDesignStatus(in polDesignStatus : MPMPolicyDesignStatus[1..1])	This operation sets the value of the current design status of this particular MPMPolicy object.
getMPMPolExecFailStrategy() : MPMPolExecFailStrategy[1..1]	This operation returns the current strategy for dealing with execution failures. This defines what actions, if any, should be taken by this particular MPMPolicy if it fails to execute correctly. If the mpmPolExecFailStrategy attribute does not have a value, then this operation **should** return a NULL string.
setMPMPolExecFailStrategy(in polExecFailStrategy : MPMPolExecFailStrategy[1..1])	This operation sets the current strategy for dealing with execution failures. This defines what actions, if any, should be taken by this particular MPMPolicy if it fails to execute correctly.
getMPMPolExecStatus() : MPMPolicyExecStatus[1..1]	This operation returns the current execution status of this MPMPolicy object. If the mpmPolExecStatus attribute does not have a value, then this operation **should** return a NULL string.
setMPMPolExecStatus(in polExecStatus : MPMPolicyExecStatus[1..1])	This operation sets the current execution status of this MPMPolicy object.
getMPMPolExecOrder() : Integer[1..1]	This operation returns the current execution order of this MPMPolicy Object.
setMPMPolExecOrder(in newOrder : Integer[1..1])	This operation sets the current execution order of this MPMPolicy Object.
getMPMPolSourceObjectList() : MPMPolicySource[1..*]	This operation retrieves the set of MPMPolicySource objects that are contained in this particular MPMPolicyStructure object. This is obtained by following the MPMPolicyHasMPMPolicySource aggregation. Each instance of this aggregation defines an MPMPolicySource object, which is then added to the return value of this operation. The return value of this operation is an array of one or more MPMPolicySource objects. If this MPMPolicyStructure object does not instantiate this aggregation, then this operation **should** return a NULL MPMPolicySource object.

Operation Name	Description
setMPMPolSourceObjectList(in polSourceObjectList : MPMPolicySource[1..*])	This operation defines a new set of MPMPolicySource objects that will be contained in this particular MPMPolicyStructure object. If this MPMPolicyStructure object already has a set of one or more MPMPolicySource objects that it contains, then those MPMPolicySource objects will be deleted by deleting both the accompanying association class and the corresponding association. Then, a new association (an instance of MPMPolicyHaMPMPolicySource) is created for each MPMPolicySource object in the polSourceObjectList parameter. Every association created **should** have a new association class created to realize the semantics of that association.
setMPMPolSourceObjectPartialList (in polSourceObjectList : MPMPolicySource[1..*])	This operation defines a new set of MPMPolicySource objects that will be contained in this particular MPMPolicyStructure object. If this MPMPolicyStructure object already has a set of one or more MPMPolicySource objects that it contains, then those MPMPolicySource objects are ignored. Then, a new association (an instance of MPMPolicyHaMPMPolicySource) is created for each MPMPolicySource object in the polSourceObjectList. Every association created **should** have a new association class created to realize the semantics of that association. Any association between this MPMPolicyStructure object and other MPMPolicySource objects that is not specified in the polSourceObjectList **shall not** be affected.
delMPMPolSourceObjectList()	This operation removes all instances of the MPMPolicyHasMPMPolicySource aggregation, and its association classes, that enables this particular MPMPolicyStructure object to contain any MPMPolicySource objects. This operation does NOT affect either the MPMPolicySource object or the MPMPolicyStructure object; it just deletes the association between this MPMPolicyStructure object and this MPMPolicySource object.
delMPMPolSourceObjectPartialList (in polSourceObjectList : MPMPolicySource[1..*])	This operation removes the association, and its association class, for each MPMPolicySource object in the polSourceObjectList that is contained by this particular MPMPolicyStructure object. This operation takes a single input parameter, called polSourceObjectList, that defines the set of MPMPolicySource objects that will be unlinked from this particular MPMPolicyStructure object. This operation does NOT affect either the MPMPolicyStructure object or the MPMPolicySource object; it just deletes the association between this MPMPolicyStructure object and this MPMPolicySource object. Any association between this MPMPolicyStructure object and other MPMPolicySource objects that is not specified in the polSourceObjectList **shall not** be affected.
getMPMPolTargetObjectList() : MPMPolicyTarget[1..*]	This operation retrieves the set of MPMPolicyTarget objects that are contained in this particular MPMPolicyStructure object. This is obtained by following the MPMPolicyHasMPMPolicyTarget aggregation. Each instance of this aggregation defines an MPMPolicyTarget object, which is then added to the return value of this operation. The return value of this operation is an array of one or more MPMPolicyTarget objects. If this MPMPolicyStructure object does not instantiate this aggregation, then this operation **should** return a NULL MPMPolicyTarget object.
setMPMPolTargetObjectList(in polTargetObjectList : MPMPolicyTarget[1..*])	This operation defines a new set of MPMPolicyTarget objects that will be contained by this particular MPMPolicyStructure object. This operation takes a single input parameter, called polTargetObjectList, which defines a set of one or more MPMPolicyTarget objects. If this MPMPolicyStructure object already has a set of one or more MPMPolicyTarget objects that it refers to, then those MPMPolicyTarget objects will be deleted by first, deleting the accompanying association class, and second, deleting the corresponding association. Then, a new association (an instance of MPMPolicyHasMPMPolicyTarget) is created for each MPMPolicyTarget object in the polTargetObjectList parameter. Every association created **should** have a new association class created to realize the semantics of that association.

Operation Name	Description
setMPMPolTargetObjectPartialList (in polTargetObjectList : MPMPolicyTarget[1..*])	This operation defines a new set of MPMPolicyTarget objects that will be contained by this particular MPMPolicyStructure object. This operation takes a single input parameter, called polTargetObjectList, which defines a set of one or more MPMPolicyTarget objects. If this MPMPolicyStructure object already has a set of one or more MPMPolicyTarget objects that it contains, then those MPMPolicyTarget objects are ignored. Then, a new association (an instance of MPMPolicyHaMPMPolicyTarget) is created for each MPMPolicyTarget object in the polTargetObjectList. Every association created **should** have a new association class created to realize the semantics of that association. Any association between this MPMPolicyStructure object and other MPMPolicyTarget objects that is not specified in the polTargetObjectList **shall not** be affected.
delMPMPolTargetObjectList()	This operation removes all instances of the MPMPolicyHasMPMPolicyTarget aggregation, and its association classes, that enables this particular MPMPolicyStructure object to refer to any MPMPolicyTarget objects. This operation does NOT affect either the MPMPolicyTarget object or the MPMPolicyStructure object; it just deletes the association between this MPMPolicyStructure object and this MPMPolicyTarget object.
delMPMPolTargetObjectPartialList (in polTargetObjectList : MPMPolicyTarget[1..*])	This operation removes the association, and its association class, for each MPMPolicyTarget object in the polSourceObjectList that is associated with this particular MPMPolicyStructure object. This operation takes a single input parameter, called polTargetObjectList, that defines the set of MPMPolicyTarget objects that will be unlinked from this particular MPMPolicyStructure object. This operation does NOT affect either the MPMPolicyStructure object or the MPMPolicyTarget object; it just deletes the association between this MPMPolicyStructure object and this MPMPolicyTarget object. Any association between this MPMPolicyStructure object and other MPMPolicyTarget objects that is not specified in the polTargetObjectList **shall not** be affected.
getMPMPolStatementList() : MPMPolicyStatement[1..*]	This operation retrieves the set of MPMPolicyStatement objects that are contained in this particular MPMPolicyStructure object. This is obtained by following the MPMPolicyHasMPMPolicyStatement aggregation. Each instance of this aggregation defines an MPMPolicyStatement object, which is then added to the return value of this operation. The return value of this operation is an array of one or more MPMPolicyStatement objects. If this MPMPolicyStructure object does not instantiate this aggregation, then this operation **should** return a NULL MPMPolicyStatement object.
setMPMPolStatementList (in polStatementObjectList : MPMPolicyStatement[1..*])	This operation defines a new set of MPMPolicyStatement objects that are contained by this particular MPMPolicyStructure object. This operation takes a single input parameter, called polStatementObjectList, which defines a set of one or more MPMPolicyStatement objects. If this MPMPolicyStructure object already has a set of one or more MPMPolicyStatement objects that it refers to, then those MPMPolicyStatement objects will be deleted by first, deleting the accompanying association class, and second, deleting the corresponding association. Then, a new association (an instance of MPMPolicyHaMPMPolicyStatement) is created for each MPMPolicyStatement object in the polStatementObjectList parameter. Every association created **should** have a new association class created to realize the semantics of that association.
setMPMPolStatementPartialList(in polStatementObjectList: MPMPolicyStatement[1..*])	This operation defines a new set of MPMPolicyStatement objects that refer to this particular MPMPolicyStructure object. This operation takes a single input parameter, called polStatementObjectList, which defines a set of one or more MPMPolicyStatement objects. If this MPMPolicyStructure object already has a set of one or more MPMPolicyStatement objects that it refers to, then those MPMPolicyStatement objects are ignored. Then, a new association (an instance of MPMPolicyHaMPMPolicyStatement) is created for each MPMPolicyStatement object in the polStatementObjectList. Every association created **should** have a new association class created to realize the semantics of that association.

Operation Name	Description
delMPMPolStatementObjectList()	This operation removes all instances of the MPMPolicyHaMPMPolicyStatement aggregation, and its association classes, that enables this particular MPMPolicyStructure object to refer to any MPMPolicyStatement objects. This operation does NOT affect either the MPMPolicyStatement object or the MPMPolicyStructure object; it just deletes the association between this MPMPolicyStructure object and this MPMPolicyStatement object.
delMPMPolStatementObjectPartialList (in polStatementObjectList: MPMPolicyStatement[1..*])	This operation removes the association, and its association class, for each MPMPolicyStatement object in the polStatementObjectList that is associated with this particular MPMPolicyStructure object. This operation takes a single input parameter, called polStatementObjectList, that defines the set of MPMPolicyStatement objects that will be unlinked from this particular MPMPolicyStructure object. This operation does NOT affect either the MPMPolicyStructure object or the MPMPolicyStatement object; it just deletes the association between this MPMPolicyStructure object and this MPMPolicyStatement object. Any association between this MPMPolicyStructure object and other MPMPolicyStatement objects that is not specified in the polStatementObjectList **shall not** be affected.

The MPMPolicyStructure class participate in four relationships.

MPMPolicyHasMPMPolicyStatement is a mandatory aggregation that defines the content of the policy as a set of one or more statements. The attachment of different MPMPolicyStatement objects changes the content, and hence the behaviour, of a given ENIPolicy. The multiplicity of this aggregation 0..1 - 1..n. If this aggregation is instantiated (i.e. the "1" part of the 1..* cardinality), then one or more MPMPolicyStatement objects can be aggregated by this particular MPMPolicyStructure object. The cardinality on the part side (MPMPolicyStatement) is 1..*; this cardinality was chosen to make explicit that any MPMPolicy shall contain at least one MPMPolicyStatement to be considered a valid policy rule. The semantics of this aggregation are defined by the MPMHasPolicyStatementDetail association class. This enables the management system to control which MPMPolicyStatement objects can be attached to which particular set of MPMPolicyStructure objects. These semantics can be further enhanced by using the Policy Pattern to define policy rules that constrain which part objects (i.e. MPMPolicyStatement) are attached to which object. The MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

MPMPolicyHasMPMPolicySource is an optional aggregation that defines a set of MPMPolicySource objects that authored, or have responsibility for, this policy. MPMPolicyHasMPMPolicyTarget defines the set of objects that will be affected by this policy. MPMPolicySource objects are used for authorization policies, as well as to enforce deontic and alethic logic. The multiplicity of this aggregation is 0..1 - 0..n. If this aggregation is instantiated (i.e. the "1" part of the 0..1 cardinality), then zero or more MCMPolicySource objects can wrap this particular MCMPolicyStructure object. The 0..* cardinality enables an MCMPolicyStructure object to be defined without having to define an associated MCMPolicySource object for it to aggregate. The semantics of this aggregation are defined by the MPMHasPolicySourceDetail association class. This enables the management system to control which set of concrete subclasses of MCMPolicyStructure can aggregate which types of MPMPolicySource objects. These semantics can be further enhanced by using the Policy Pattern to define policy rules that constrain which part objects (i.e. MPMPolicyStatement) are attached to which object. The MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

MPMPolicyHasMPMPolicyTarget is a mandatory aggregation that defines the set of MPMPolicyTarget objects that are attached to this particular MPMPolicyStructure object. MPMPolicyTarget objects are MCMManagedEntity objects whose state and/or behaviour will be affected by the execution of a set of MPMPolicy objects. The multiplicity of this aggregation is 0..1 - 0..n. If this aggregation is instantiated (i.e. the "1" part of the 0..1 cardinality), then zero or more MPMPolicyTarget objects can be aggregated by this particular MPMPolicyStructure object. Note that the cardinality on the part side (MPMPolicyTarget) is 0..*; this enables an MPMPolicyStructure object to be defined without having to define an associated MPMPolicyTarget object for it to aggregate. The semantics of this aggregation are defined by the MPMHasPolicyTargetDetail class. This enables the management system to control which set of concrete subclasses of MCMPolicyStructure can aggregate which types of MPMPolicyTarget objects. These semantics can be further enhanced by using the Policy Pattern to define policy rules that constrain which part objects (i.e. MPMPolicyStatement) are attached to which object. The MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

The MPMPolicyStructureHasPolicy aggregation is an optional aggregation and defines the set of MPMPolicyStructure Objects that are attached to this particular MPMPolicyStructureComposite Object. The semantics of this aggregation are defined by the MPMPolicyStructureHasPolicyDetail association class. The multiplicity of this aggregation is 0..1 - 0..*. If this aggregation is instantiated (i.e. the "1" part of the 0..1 cardinality), then zero or more (0..*) MPMPolicyStructure Objects can be contained in this particular MPMPolicyStructureComposite Object.

The 0..* cardinality enables an MCMPolicyStructureComposite Object to be defined without having to first define an associated MPMPolicyStructure Object for it to aggregate. The semantics of this aggregation are defined by the MPMPolicyStructureHasPolicyDetail association class. This enables the management system to control which set of concrete subclasses of MCMPolicyStructureComposite can aggregate which types of MPMPolicyStructure Objects. The MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

5.3.2.6.4 MPMPolicyStructure Subclasses

5.3.2.6.4.1 MPMPolicyStructureAtomic

This is an abstract class that specializes MPMPolicyStructure. This class represents stand-alone MPMPolicy Objects. It is shown in Figure 5-18.

An MPMPolicyStructureAtomic Object **shall not** contain another MPMPolicyStructure Object.

At the current time, this class defines no attributes. This is because this class is meant to ontologically distinguish between stand-alone and composite MPMPolicyStructure objects.

Table 5-59 defines the operations for this class.

Table 5-59: Operations of the MPMPolicyStructureAtomic Class

Operation Name	Description
getMPMPolicyAtomicParent() : MPMStructure[1..1]	This operation returns the parent of this MPMPolicyStructureAtomic object. If this MPMPolicyStructureAtomic object is a stand-alone MMPolicy, then a NULL MPMPolicyStructure object **should** be returned. If this MPMPolicyStructureAtomic object is part of a policy hierarchy, then a MPMPolicyStructureComposite object **shall** be returned.
setMPMPolicyAtomicParent(in: newParent : MPMPolicyStructure [1..1])	This operation sets the parent of this MPMPolicyStructureAtomic object. If this MPMPolicyStructureAtomic object is a stand-alone MMPolicy, then its parent **shall** be a NULL MPMPolicyStructure object. If this MPMPolicyStructureAtomic object is part of a policy hierarchy, then its parent **shall** be a MPMPolicyStructureComposite object.

At the current time, this class has no relationships.

5.3.2.6.4.2 MPMPolicyStructureComposite

This is an abstract class that specializes MPMPolicyStructure. This class represents a set of related MCMPolicyStructure Objects that are organized into a tree structure. It is shown in Figure 5-18.

Each MPMPolicyStructureComposite Object **may** contain zero or more MPMPolicyStructureAtomic objects.

Each MPMPolicyStructureComposite Object **may** contain zero or more MPMPolicyStructureComposite Objects.

At the current time, this class defines no attributes. This is because this class is meant to ontologically distinguish between stand-alone and composite MPMPolicyStructure objects.

Table 5-60 defines the operations of this class.

Table 5-60: Operations of the MPMPolicyStructureComposite Class

Operation Name	Description
getMPMPolicyList() : MPMStructure[1..*]	This operation returns the set of all MPMPolicyStructure objects (i.e. the list is made up of MPMPolicyStructureAtomic and/or MPMPolicyStructureComposite objects) that are contained in this specific MPMPolicyStructureComposite Object. If this Object does not contain any MPMPolicyStructure Objects, then a NULL MPMPolicyStructure Object **should** be returned.
setMPMPolicyList (in childObjectList : MPMPolicyStructure [1..*])	This operation defines a set of MPMPolicyStructure Objects that will be contained by this particular MPMPolicyStructureComposite Object. This operation first deletes any existing contained MPMPolicyStructure Objects (and their aggregations and association classes), and then instantiates a new set of MPMPolicyStructure objects for each element in the input parameter. Each MPMPolicyStructure object is contained within this particular MPMPolicyStructure Composite object by creating an instance of the MCMPolicyStructureHasPolicy aggregation. Each created aggregation **should** have an association class (i.e. an instance of the MCMPolicyStructureHasPolicy association class).
setMPMPolicyPartialList (in childObjectList : MPMPolicyStructure[1..*])	This operation defines a set of one or more MPMPolicyStructure Objects that should be contained within this particular MPMPolicyStructureComposite Object **without** affecting any other existing contained MPMPolicyStructure Objects or the Objects that are contained in them. This operation creates a set of aggregations between this particular MPMPolicyStructureComposite Object and each of the MPMPolicyStructure objects identified in the input parameter. Each created aggregation **should** have an association class (i.e. an instance of the MPMPolicyStructureHasPolicy association class).
delMPMPolicyList()	This operation deletes all contained MPMPolicyStructure Objects of this particular MPMPolicyStructureComposite Object. This has the effect of removing both the association class and the aggregation between this MPMPolicyStructureComposite Object and each MPMPolicyStructure Object that is contained in this MPMPolicyStructureComposite Object.
delMPMPolicyPartialList (in childObjectList : MPMPolicyStructure[1..*])	This operation deletes a set of MPMPolicyStructure Objects from this particular MPMPolicyStructureComposite Object. This has the effect of removing both the association class and the aggregation between each MPMPolicyStructure object specified in the input parameter and this MPMPolicyStructureComposite object. All other aggregations between this MPMPolicyStructureComposite and other MPMPolicyStructure Objects that are not identified in the input parameter **shall not** be affected.

There is one relationship defined for the MPMPolicyStructureComposite class, called MPMPolicyStructureHasPolicy. It was defined in clause 5.3.2.6.3.

5.3.2.6.4.3 MPMImperativePolicy

This is a mandatory abstract class, which is a type of PolicyContainer that is used to represent imperative policy rules. An imperative policy explicitly defines how the state of the target MCMManagedEntity objects will be affected. This version of the present document supports two types of imperative policy rules:

1) ECA policy rules; and

2) commands.

Figure 5-18 shows the attributes and operations of the MPMImperativePolicy class.

Table 5-61 defines the operations of this class.

Table 5-61: Attributes of the MPMPolicyStructureComposite Class

Attribute Name	Description
mpmImpPolPriority : Integer[0..1]	This is an optional non-negative integer attribute that defines the priority of this particular MPMImperativePolicy object. A larger value indicates a higher priority. Priority can be used to resolve conflicts among policy actions by selecting the policy action with the highest priority. Priority can also be used to define the execution order of a set of policy rules. A default value of 0 **may** be assigned.
mpmImpPolExecStrategy : MPMImpPolExecStrategy [1**]**	This is a mandatory non-negative integer attribute that defines the execution strategy of this particular MPMImperativePolicy object. The execution strategy consists of the order that actions will execute, and whether encountering an error terminates the process of executing actions or not. If no actions are contained in this MPMImperativePolicy class, then an error **shall** be returned.

Table 5-62 shows the operations of the MPMImperativePolicy class.

Table 5-62: Operations of the MPMImperativePolicy Class

Operation Name	Description
getMPMImpPolPriority() : Integer[1..1]	This operation returns the current value of the mpmImpPolRulePriority attribute. If the mpmImpPolRulePriority attribute does not have a value, then this operation **shall** return an error.
setMPMImpPolPriority (in polRulePriority : Integer[1..1])	This operation sets the value of the mpmImpPolRulePriority attribute. The value of the mpmImpPolRulePriority attribute **shall** be a non-negative integer.
getMPMImpPolExecStrategy() : MPMImpPolExecStrategy [1..1]	This operation returns the current value of the mpmImpPolExecStrategy attribute. If the mpmImpPolExecStrategy attribute does not have a value, then this operation **shall** return an error.
setMPMImpPolExecStrategy(in newStrategy : MPMImpPolExecStrategy[1..1])	This operation sets the value of the mpmImpPolExecStrategy attribute.

At the current time, this class participates in no relationships.

5.3.2.6.4.4 MPMDeclarativePolicy

This is a mandatory concrete class, which is a type of PolicyContainer that is used to represent declarative policy rules. A declarative policy uses statements to express the goals of the policy, but not how to accomplish those goals.

In the present document, Declarative Policy are defined as policies that execute as theories of a formal logic.

A Declarative Policy **shall** be written using propositional, predicate, or a higher form of a formal logic.

Figure 5-18 shows the attributes and operations of the MPMDeclarativePolicy class.

Table 5-63 defines its attributes.

Table 5-63: Attributes of the MPMImperativePolicy Class

Attribute Name	Description
mpmDecPolLogicType : MPMFormalLogicType[1..1]	This is a mandatory non-negative enumerated integer. It defines the type of formal logic used by this MPMDeclarativePolicy object.

Table 5-64 shows the operations of the MPMImperativePolicy class.

Table 5-64: Operations of the MPMImperativePolicy Class

Operation Name	Description
getMPMDeclLogicType() : MPMFormalLogicType[1..1]	This operation returns the current value of the mpmDecPolLogicType attribute. If the mpmDecPolLogicType attribute does not have a value, then this operation **shall** return an error.
setMPMDeclLogicType(in polLogicType : MPMFormalLogicType[1..1])	This operation sets the value of the mpmDecPolLogicType attribute.

At the current time, this class participates in no relationships.

5.3.2.6.4.5 MPMIntentPolicy

This is a mandatory concrete class, which is a type of PolicyContainer that is used to represent intent policy rules.

An intent policy is a type of policy that uses statements from a restricted natural language to express the goals of the policy, but not how to accomplish those goals. An Intent Policy is written in a Controlled Language (i.e. a language that restricts the grammar and vocabulary used). In particular, formal logic syntax **shall not** used. This version of the present document will use a restricted version of English. Controlled languages simplify machine translation of the source content, and enable the source content to be translated to other types of languages. An example of a Controlled Language is the Attempto Controlled English language; most Domain Specific Languages (DSLs) are also Controlled Languages.

An Intent Policy **shall** be written in a Controlled Language.

An Intent Policy **may** be written in a DSL.

In general, a newly written intent is likely to not be directly executable. This is because of ambiguities in using a Controlled Language, as well as the use of more abstract comments. For example, a Customer might be referred to by name; this would need to be translated to a form that is machine processable (e.g. an IP address).

Figure 5-18 shows the attributes and operations of the MPMIntentPolicy class.

Table 5-65 defines its attributes.

Table 5-65: Attributes of the MPMIntentPolicy Class

Attribute Name	Description
mpmIntentTranslationStatus : MPMIntentTranslationStatus[1..1]	This is a mandatory non-negative enumerated integer, and defines the status of the translation of the content of this MPMIntentPolicy. If the value of the mpmIntentTranslationStatus attribute is not 2 (i.e. SUCCESS), then this MPMIntentPolicy **shall not** be executed.

Table 5-66 defines its attributes.

Table 5-66: Operations of the MPMIntentPolicy Class

Operation Name	Description
getMPMIntentTranslationStatus() : MPMIntentTranslationStatus[1..1]	This operation returns the current value of the mpmIntentTranslationStatus attribute. If the mpmIntentTranslationStatus attribute does not have a value, then this operation **shall** return an error.
setMPMIntentTranslationStatus(in intentTranslationStatus : MPMIntentTranslationStatus[1..1])	This operation sets the value of the mpmIntentTranslationStatus attribute.

At the current time, this class participates in no relationships.

5.3.2.6.4.6 MPMECAPolicy

This is a mandatory concrete class that specializes MPMImperativePolicy. An MPMECAPolicy is a PolicyContainer that aggregates a set of events, conditions, and actions into an imperative policy rule known as an Event-Condition-Action (ECA) policy rule. This has the following semantics:

```
IF the event portion of the policy rule evaluates to TRUE
   IF the condition portion of the policy rule evaluates to TRUE
      THEN actions in the action portion of the policy rule may be executed
   ENDIF
ENDIF
```

In the above definition:

1) An event is a Boolean clause that represents something that happens or is happening that triggers a decision-making process to start.

2) A condition is a Boolean clause that is an evaluation in a decision-making process.

3) An action is a Boolean clause that defines an atomic computation that is executed as a result of a decision-making process.

4) Actions **may** be executed according to the semantics of the MPMECAPolicyRule instance.

Figure 5-18 shows the attributes and operations of the MPMECAPolicy class.

The event, condition, and action portions of an MPMECAPolicy will be referred to as Event, Condition, and Action Statements (to differentiate them from Event, Condition, and Action objects). The Event, Condition, and Action Statements are all Boolean clauses (i.e. a statement that produces a value of either true or false). An MPMECAPolicy refines the notion of an MPMImperativePolicy by mandating that at least one Event or Condition Statement is present, and at least one Action Statement is present.

An MPMECAPolicy **shall** contain at least one Event Statement or at least one Condition Statement.

An MPMECAPolicy **shall** contain at least one Action Statements.

Any Boolean statement can be combined with another Boolean statement to form compound Boolean statements using any of the logical connectives (i.e. AND, OR, and NOT). This realizes the concept of a portion of an MPMECAPolicy evaluating to true. For example, if an event Boolean clause is true, that satisfies the first IF statement in the above pseudocode. As another example, the event portion of an MPMECAPolicy may consist of two or more Boolean statements; this enables the evaluation of the event portion to be determined by the Boolean value of each statement according to the logical connectives that are present in the event portion. Boolean statements are realized by the MPMBooleanStatement class. Note that other types of MPMPolicyStatements may be combined with one or more MPMBooleanStatements for any of the Event, Condition, and Action Statements. In addition, any Event, Condition, or Action Statement can be decorated with a concrete subclass of the MPMPolicyComponentDecorator.

The following requirements summarize the structural semantics of an MPMECAPolicy:

1) An MPMECAPolicy **shall** have an action portion that is made up of one or more MPMPolicyStatements.

2) An MPMECAPolicy **may** contain one or more Event Statements.

3) If an MPMECAPolicy does not contain an Event Statement, the Condition Statement **shall** both trigger the start of the decision-making process and evaluate the decision.

4) An MPMECAPolicy **may** contain one or more Condition Statements.

5) If an MPMECAPolicy does not contain a Condition Statement, the Event Statement **shall** both trigger the start of the decision-making process and evaluate the decision.

6) Either the Event Statement or the Condition Statement, but not both, **may** be NULL in an MPMECAPolicy.

The following requirements summarize the behavioural semantics of an MPMECAPolicy:

1) An MPMECAPolicy **shall** contain one or more MPMBooleanStatements.

2) Each of the Event, Condition, and Action Statements **shall** contain one or more MPMBooleanStatements.

3) Any MPMBooleanStatement **may** contain other types of MPMPolicyStatements, as long as their addition does not prevent the MPMBooleanStatement from evaluating to either true or false.

4) Any MPMBooleanStatement **may** be decorated by one or more concrete subclasses of the MPMPolicyComponentDecorator class, as long as their addition does not prevent the MPMBooleanStatement from evaluating to either true or false.

There are currently no attributes, operations, or relationships defined for this class. Its purpose is to provide a concrete realization of a particular type of MPMImperativePolicy with the above semantics.

5.3.2.6.4.7 MPMCommandPolicy

This is a mandatory concrete class, whose superclass is MPMImperativePolicy. An MPMCommandPolicy is a PolicyContainer that contains one or more Action Statements. Stylistically, it corresponds to the imperative mood in English.

The difference between an MPMCommandPolicy and an MPMECAPolicy is that the former only has a set of Action Statements, whereas the latter has either an Event and/or a Condition Statement in addition to set of Action Statements.

The following requirements summarize the structural semantics of an MPMECAPolicy:

1) An MPMCommandPolicy **shall** contain one or more Action Statements.

2) An instance of this class **shall not** contain Event or Condition Statements.

The following requirements summarize the behavioural semantics of an MPMECAPolicy.

1) Each Action Statement **shall** contain one or more MPMBooleanStatements.

2) Any MPMBooleanStatement **may** contain other types of MPMPolicyStatements, as long as their addition does not prevent the MPMBooleanStatement from evaluating to either true or false.

3) Any MPMBooleanStatement **may** be decorated by one or more concrete subclasses of the MPMPolicyComponentDecorator class, as long as their addition does not prevent the MPMBooleanStatement from evaluating to either true or false.

Figure 5-18 shows the attributes and operations of the MPMCommand Policy class. There are currently no attributes or operations defined for this class. Its purpose is to provide a concrete realization of a particular type of MPMImperativePolicy that does not need Event and Condition Statements.

5.3.2.6.5 MPMPolicyComponentStructure Class Hierarchy

The top portion of this class hierarchy is shown in Figure 5-19. This class hierarchy is defined to facilitate adding new types of policy components later. The main "worker" class is MPMPolicyStatement; concrete subclasses of this class are aggregated by all types of policy rules (i.e. concrete subclasses of MPMPolicyStructure). An MPMPolicyStatement object **may** optionally be made up of MPMPolicyClause objects, which are used to define custom MPMPolicyStatements.

An MPMPolicyComponentDecorator is used to define optional objects (or parts of an object) to decorate, or wrap, concrete subclasses of MPMPolicyStatement and/or MPMPolicyClause objects.

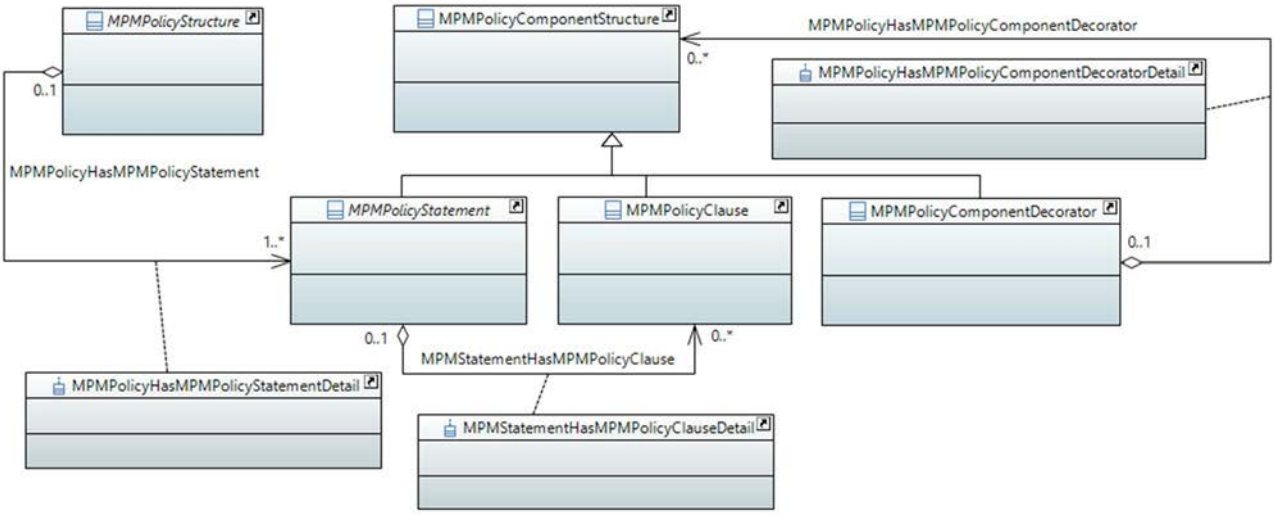

Figure 5-19: MPMPolicyComponentStructure Hierarchy

5.3.2.6.6 MPMPolicyComponentStructure Class Definition

This is a mandatory abstract class. It is the superclass for all types of components that may be contained in a particular type of an MPMPolicy. In this model, the type of Policy (e.g. imperative, declarative, intent) is a type of PolicyContainer. The type of PolicyContainer defines the type of MPMPolicyStructureComponent objects that it can contain.

The version of the present document does not define any attributes or operations for this class. Its main purpose is from an ontological perspective, as it is used as the superclass for all types of components that can be contained by all types of policies that are defined by the MPM. This class participates in one relationship, called MPMPolicyHasMPMPolicyComponentDecorator, which is defined in clause 5.3.2.6.7.15.

5.3.2.6.7 MPMPolicyComponentStructure Subclasses

5.3.2.6.7.1 MPMPolicyStatement Class Hierarchy

This clause describes the main subclasses of the MPMPolicyStatement class hierarchy. An MPMPolicyStatement provides the important abstraction that *all types of MPMPolicies contain a set of one or more MPMPolicyStatements.* This fundamental abstraction enables the content of multiple types of MPMPolicyRules to be represented using a common mechanism.

This class and its concrete subclasses can have their functionality enhanced and/or customized by using the MPMPolicyClause class and its subclasses (this is described in clause 5.3.2.6.7.8) and subclasses of MPMPolicyComponentDecorator (this is described in clause 5.3.2.6.7.15).

The MPMPolicyStatement class hierarchy is shown in Figure 5-20.

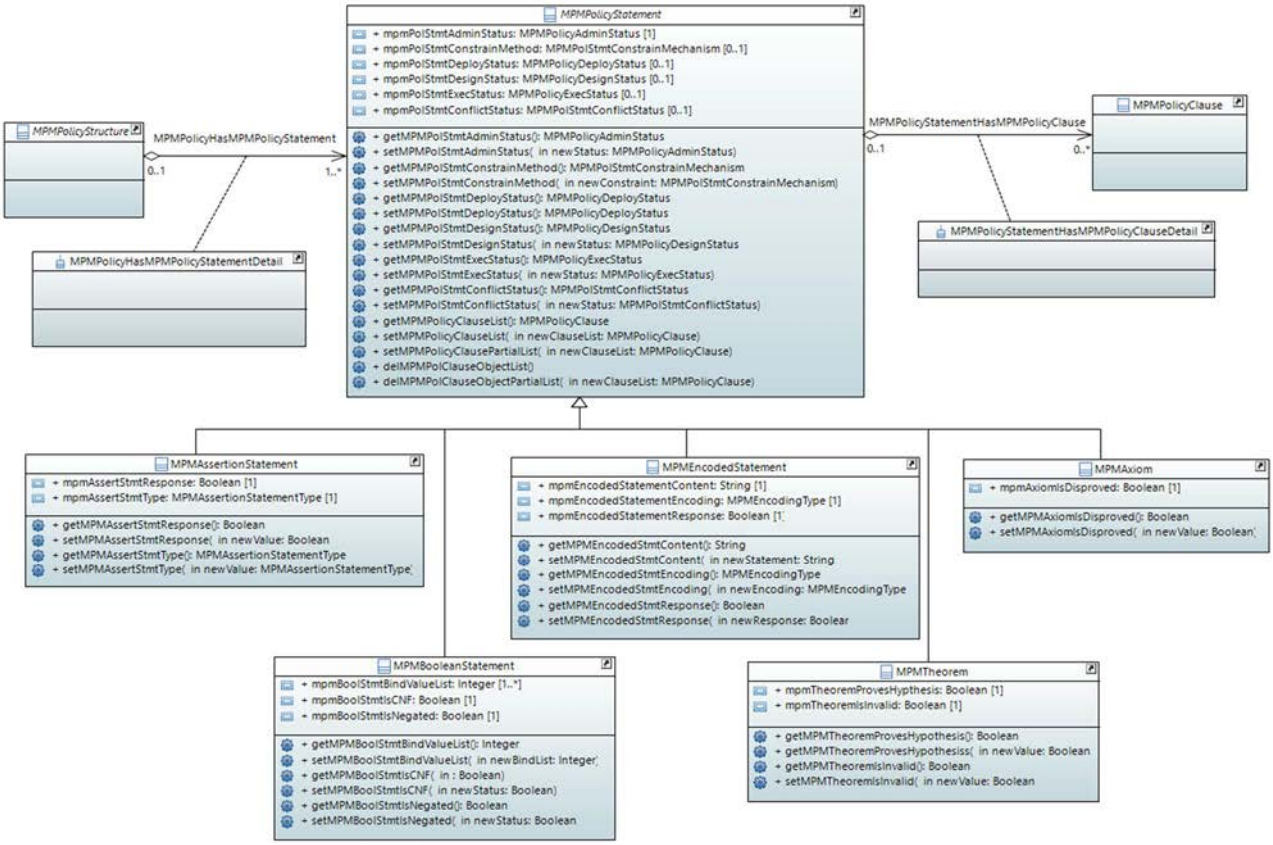

Figure 5-20: MPMPolicyStatement Class Hierarchy

5.3.2.6.7.2 MPMPolicyStatement Class Definition

This is a mandatory abstract class. It separates the representation of an MPMPolicy from its implementation.

An MPMPolicy, regardless of its structure and semantics, **shall** be abstracted into a set of statements, which are instances of this class. This is defined by one or more instances of the MPMPolicyHasMPMPolicyStatement aggregation.

Each MPMPolicyStatement **may** optionally be abstracted into a set of clauses, which are instances of MPMPolicyClause (see clause 5.3.2.6.7.9). This is defined by one or more instances of the MPMPolicyStatementHasMPMPolicyClause aggregation.

Each MPMPolicyClause is made up of a set of policy elements. The type of MPMPolicyStructure determines the type of policy statements that it can contain; this in turn determines the types of policy clauses and policy elements that are used by any given type of policy statement.

There are two ways to enforce the semantics of restricting the type of MPMPolicyStatements that can be contained in a particular type of MPMPolicyStructure:

1) Use the MPMPolicyHasMPMPolicyStatementDetail association class.

2) Define appropriate OCL statements to enforce the restriction.

The first operation avoids the use of OCL, but is harder to implement. It uses the model elements of the MPMPolicyHasMPMPolicyStatementDetail association class to define explicit semantics to restrict the type of MPMPolicyStatement, and their decorations, that can be contained by this particular type of MPMPolicyStructure. The second is easier, since OCL is a formal language that enables these semantics to be easily defined. However, some implementations do not support OCL, so the particular choice of which operation to use is left to the implementer.

Table 5-67 defines the attributes for this class.

Table 5-67: Attributes of the MPMPolicyStatement Class

Attribute Name	Description
mpmPolStmtAdminStatus : MPMPolicyAdminStatus[1..1]	This is a mandatory enumerated non-negative integer attribute that defines the current administrative status of this particular MPMPolicyStatement object.
mpmPolStmtConstrainMethod : MPMPolMethodConstrain-Mechanism[0..1]	This is a non-negative enumerated integer, and defines the mechanism used to constrain which concrete subclasses of MPMPolicyStatement can be used with this particular concrete subclass of MPMPolicyStructure.
mpmPolStmtDeployStatus : MPMPolStatementDeploy-Status[0..1]	This is an optional enumerated, non-negative integer attribute. It is used to indicate whether this MPMPolicyStatement can or cannot be deployed by the policy management system. This attribute enables the policy manager to know which MPMPolicies are currently deployed, and may be useful for the policy execution system for planning the staging of MPMPolicies.
mpmPolStmt-DesignStatus : MPMPolicy-DesignStatus[0..1]	This is an optional enumerated, non-negative integer whose value defines the current design status of this MPMPolicyStatement object.
mpmPolStmtExecStatus : MPMPolicyExecStatus[1..1]	This is a mandatory enumerated non-negative enumerated integer whose value defines the current execution status of this MPMPolicyStatement object.
mpmPolStmtConflictStatus : MPMPolStmtConflict-Status[1..1]	This is an optional enumerated, non-negative integer whose value defines whether this particular MPMPolicyStatement has, or ever had, a conflict with another MPMPolicyStatement. If the value of this attribute is not "RESOLVED" or "NONE", then this MPMPolicyStatement object **shall not** be used.

Table 5-68 defines the operations for this class.

Table 5-68: Operations of the MPMPolicyStatement Class

Operation Name	Description
getMPMPolStmtAdminStatus() : MPMPolicyAdminStatus[1..1]	This operation returns the current value of the mpmPolStmtAdminStatus attribute. If the mpmPolStmtAdminStatus attribute does not have a value, then this operation **shall** return an error.
setMPMPolStmtAdminStatus(in newStatus : MPMPolicyAdminStatus[1..1])	This operation sets the value of the mpmPolStmtAdminStatus attribute.
getMPMPolStmtConstrain-Method() : MPMPolStmtConstrain-Mechanism[1..1]	This operation returns the current value of the mpmPolConstrainMethod attribute. If the mpmPolConstrainMethod attribute does not have a value, then this operation **shall** return an error.
setMPMPolStmtConstrain-Method (in newStatus : MPMPol-StmtConstrainMechanism[1..1])	This operation sets the value of the mpmPolConstrainMethod attribute.
getMPMPolStmtDeployStatus() : MPMPolicyDeployStatus[1..1]	This operation returns the current value of the mpmPolStmtDeployStatus attribute. If the mpmPolStmtDeployStatus attribute does not have a value, then this operation **shall** return an error.
setMPMPolStmtDeployStatus(in newStatus : MPMPolicyDeployStatus[1..1])	This operation sets the value of the mpmPolStmtDeployStatus attribute.
getMPMPolStmtDesignStatus() : MPMPolicyDesignStatus[1..1]	This operation returns the current value of the mpmPolStmtDesignStatus attribute. If the mpmPolStmtDesignStatus attribute does not have a value, then this operation **shall** return an error.
setMPMPolStmtDesignStatus(in newStatus : MPMPolicyDesignStatus[1..1])	This operation sets the value of the mpmPolStmtDesignStatus attribute.
getMPMPolStmtExecStatus() : MPMPolicyExecStatus[1..1]	This operation returns the current value of the mpmPolStmtExecStatus attribute. If the mpmPolStmtExecStatus attribute does not have a value, then this operation **shall** return an error.
setMPMPolStmtExecStatus(in newStatus : MPMPolicyExecStatus[1..1])	This operation sets the value of the mpmPolStmtExecStatus attribute.

Operation Name	Description
getMPMPolStmtConflictStatus() : MPMPolStmtConflictStatus[1..1])	This operation returns the current value of the mpmPolStmtConflictStatus attribute. If the mpmPolStmtConflictStatus attribute does not have a value, then this operation **shall** return an error.
setMPMPolStmtConflictStatus(in newStatus : MPMPolStmtConflictStatus[1..1])	This operation sets the value of the mpmPolStmtConflictStatus attribute.
getMPMPolicyClauseList() : MPMPolicyClause[1..*]	This operation retrieves the set of MPMPolicyClause objects that are contained in this particular MPMPolicyStatement object. This is obtained by following the MPMStatementHasMPMPolicyClause aggregation. Each instance of this aggregation defines an MPMPolicyClause object, which is then added to the return value of this operation. If this MPMPolicyStatement object does not instantiate this aggregation, then this operation **should** return a NULL MPMPolicyClause object.
setMPMPolicyClauseList(in newClauseList : MPMPolicyClause[1..*])	This operation defines a new set of MPMPolicyClause objects that will be contained in this particular MPMPolicyStatement object. If this MPMPolicyStatement object already has a set of one or more MPMPolicyClause objects that it contains, then those MPMPolicyClause objects will be deleted by deleting both the accompanying association class and the corresponding association. Then, a new association is created for each MPMPolicyClause object in the newClauseList parameter. Every association created **should** have a new association class created to realize the semantics of that association.
setMPMPolicyClausePartialList(in newClauseList : MPMPolicyClause[1..*])	This operation defines a new set of MPMPolicyClause objects that will be contained in this particular MPMPolicyStatement object. If this MPMPolicyStatement object already has a set of one or more MPMPolicyClause objects that it contains, then those MPMPolicyClause objects are ignored. Then, a new association is created for each MPMPolicyClause object in the newClauseList. Every association created **should** have a new association class created to realize the semantics of that association. Any association between this MPMPolicyStatement object and other MPMPolicyClause objects that is not specified in the newClauseList **shall not** be affected.
delMPMPolClauseObjectList()	This operation removes all instances of the MPMStatementHasMPMPolicyClause aggregation, and its association classes, that enables this particular MPMPolicyStatement object to contain any MPMPolicyClause objects. This operation does **not** affect either the MPMPolicyClause object or the MPMPolicyStatement object; it just deletes the association between this MPMPolicyStatement object and this MPMPolicyClause object.
delMPMPolClauseObjectPartial-List(in newClauseList: MPMPolicyClause[1..*])	This operation removes the association, and its association class, for each MPMPolicyClause object in the newClauseList that is contained by this particular MPMPolicyStatement object. This operation does NOT affect either the MPMPolicyStatement object or the MPMPolicyClause object; it just deletes the association between this MPMPolicyStatement object and this MPMPolicyClause object. Any association between this MPMPolicyStatement object and other MPMPolicySource objects that is not specified in the newClauseList **shall not** be affected.

The MPMPolicyStatement class participates in two aggregations.

MPMPolicyHasMPMPolicyStatement is a mandatory aggregation that was defined in clause 5.3.2.6.3. This defines the set of MPMPolicyStatement objects that form the content of a given MPMPolicyStructure object.

The MPMStatementHasMPMPolicyClause Aggregation is an optional aggregation that defines the set of MPMPolicyClause objects that make up this particular MPMPolicyStatement. This aggregation enables the content of an MPMPolicyStatement to be changed without affecting the rest of the MPMPolicy.

The multiplicity of this aggregation is 0..1 - 0..n. If this aggregation is instantiated (i.e. the "1" part of the 0..1 cardinality), then zero or more MPMPolicyClause objects define the content of this particular MPMPolicyStatement object. The 0..* cardinality enables an MPMPolicyStatement object to be defined without having to define an associated MPMPolicyClause object for it to aggregate. The semantics of this aggregation are defined by the MPMStatementHasMPMPolicyClauseDetail association class. This enables the management system to control which set of concrete subclasses of MCMPolicyStatement can aggregate which types of MPMPolicyClause objects.

The MPMStatementHasMPMPolicyClauseDetail is a concrete association class, and defines the semantics of the MPMStatementHasMPMPolicyClause aggregation. The attributes and relationships of this class can be used to define which MPMPolicyClause objects can be aggregated by which particular set of MCMPolicyStatement objects. These semantics can be further enhanced by using the Policy Pattern to define policy rules that constrain which part objects (i.e. MPMPolicyClause) are attached to which MCMPolicyStatement object. The MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

5.3.2.6.7.3 MPMAssertionStatement Class Definition

An MPMAssertionStatement is a collection of one or more MPMAssertionClauses (see clause 5.3.2.6.7.10). The canonical form of an MPMAssertionStatement is a 3-tuple, containing three MPMAssertionClauses:

<pre-condition, post-condition, invariant>

In this definition:

1) Pre-conditions are predicates that **shall** be true in order for a operation or function to execute.

2) Post-conditions are predicates that **shall** be true after a operation or function has executed.

3) Attributes are predicates that **shall** be true during the life of operation or function execution (i.e. they are invariant through the life of the operation).

A valid MPMAssertionStatement **shall** specify at least one pre-condition, post-condition, or invariant attribute.

This 3-tuple is especially useful when reasoning about whether a computer program is correct. An enumeration (MPMAssertionStatementType) is defined that specifies what types of MPMAssertionClauses are used by this particular MPMAssertionStatement. In general, an MPMAssertionStatement is typically used in imperative policies; MPMAxiom and MPMTheorem policies are used in declarative policies, and MPMEncodedStatement and MPMBooleanStatement can be used by all policy types.

Figure 5-21 shows the MPMAssertionStatement class.

Figure 5-20: MPMPolicyComponentStructure Hierarchy

Table 5-69 defines the attributes for this class.

Table 5-69: Attributes of the MPMPolicyStatement Class

Attribute Name	Description
mpmAssertStmtResponse : Boolean[1..1]	This is a mandatory Boolean attribute that provides a Boolean response for this MPMAssertionStatement. This enables this MPMAssertionStatement to be combined with other subclasses of an MPMPolicyStatement that provide a Boolean value that defines the status as to their correctness and/or evaluation state. This enables this object to be used to construct more complex MPMPolicyStatements.
mpmAssertStmtType : MPMAssertionStatementType[1..1]	This is a mandatory enumerated non-negative integer attribute that defines the composition of this particular MPMAssertionStatement object.

Table 5-70 defines the operations for this class.

Table 5-70: Operations of the MPMPolicyStatement Class

Operation Name	Description
getMPMAssertStmtResponse() : Boolean[1..1]	This operation returns the current value of the mpmAssertStmtResponse attribute. If the mpmAssertStmtResponse attribute does not have a value, then this operation **shall** return an error.
setMPMAssertStmtResponse (in newValue : Boolean[1..1])	This operation sets the value of the mpmAssertStmtResponse attribute.
getMPMAssertStmtType() : MPMAssertionStatementType[1..1]	This operation returns the current value of the mpmAssertStmtType attribute. If the mpmAssertStmtType attribute does not have a value, then this operation **shall** return an error.
setMPMAssertStmtType(in newValue : MPMAssertionStatementType[1..1])	This operation sets the value of the mpmAssertStmtType attribute.

Note that there are no operations that retrieve the number of MPMAssertionClause objects from an MPMAssertionStatement. This is because of two reasons. First, the MPMAssertionStatement object inherits the MPMStatementHasMPMPolicyClause aggregation from its superclass (see clause 5.3.2.6.7.2); this defines the number of MPMPolicyClause objects that are aggregated in this particular MPMAssertionStatement object. Second, the MPMAssertionStatement can aggregate more than one type of MPMPolicyClause object.

5.3.2.6.7.4 MPMBooleanStatement Class Definition

An MPMBooleanStatement specializes an MPMPolicyStatement. It defines a statement that evaluates to either true or false. An MPMBooleanStatement may be made up of one or more Boolean clauses, which is a subclass of the MPMPolicyClause class (see clause 5.3.2.6.7.9). This is modeled using the MPMStatementHasMPMPolicyClause aggregation (see clause 5.3.2.6.7.2).

Boolean expressions correspond to propositional formulas in logic. Hence, an MPMBooleanStatement may be used by imperative, declarative, and intent policies.

Figure 5-21 shows the MPMBooleanStatement class.

Table 5-71 defines the attributes for this class.

Table 5-71: Attributes of the MPMBooleanStatement Class

Attribute Name	Description
mpmBoolStmtBindValue : Integer[1..*]	This is a mandatory array of positive integers that defines the order in which constituent terms bind to this MPMBooleanStatement. For example, the Boolean expression "((A AND B) OR (C AND NOT (D OR E)))" has the following binding order: terms A and B have a bind value of 1; term C has a binding value of 2, and terms D and E have a binding value of 3. All values in this attribute **shall** be greater than 0.
mpmBoolStmtIsCNF : Boolean[1..1]	This is a mandatory Boolean attribute. If the value of this attribute is TRUE, then this MPMBooleanStatement is in Conjunctive Normal Form. Otherwise, it is in Disjunctive Normal Form.
mpmBoolStmtIsNegated: Boolean[1..1]	This is a mandatory Boolean attribute. If the value of this attribute is TRUE, then this (entire) MPMBooleanStatement is negated.

Table 5-72 defines the operations for this class.

Table 5-72: Operations of the MPMBooleanStatement Class

Operation Name	Description
getMPMBoolStmt-BindValueList() : Integer[1..*]	This operation returns the current value of the mpmBoolStmtBindValue attribute, which is an array of positive integers. If the mpmBoolStmtBindValue attribute does not have a value, then this operation **shall** return an error.
setMPMBoolStmt-BindValueList (in newBindList : Integer[1..1])	This operation sets the value of the mpmBoolStmtBindValue attribute. All values in this attribute **shall** be greater than 0.
getMPMBoolStmtIsCNF() : Boolean[1..1]	This operation returns the current value of the mpmBoolStmtIsCNF attribute. If the mpmBoolStmtIsCNF attribute does not have a value, then this operation **shall** return an error.
setMPMBoolStmtIsCNF (in newStatus : Boolean[1..1])	This operation sets the value of the mpmBoolStmtIsCNF attribute.
getMPMBoolStmtIsNegated() : Boolean[1..1]	This operation returns the current value of the mpmBoolStmtIsNegated attribute. If the mpmBoolStmtIsNegated attribute does not have a value, then this operation **shall** return an error.
setMPMBoolStmtIsNegated (in newStatus : Boolean[1..1])	This operation sets the value of the mpmBoolStmtIsNegated attribute.

Note that there are no operations that retrieve the number of MPMBooleanClause objects from an MPMBooleanStatement. This is because of two reasons. First, the MPMBooleanStatement object inherits the MPMStatementHasMPMPolicyClause aggregation from its superclass; this defines the set of MPMPolicyClause objects that are aggregated by this particular MPMBooleanStatement. Second, the MPMBooleanStatement can aggregate more than one type of MPMPolicyClause object.

5.3.2.6.7.5 MPMEncodedStatement Class Definition

An MPMEncodedStatement represents a policy statement as an encoded object. This class defines a generalized extension mechanism for representing MPMPolicyStatements that have not been modeled with other MPMPolicyComponentStructure objects. For example, this class is useful for encoding YANG and RDF statements.

This class encodes the contents of the policy clause directly into the attributes of the MPMEncodedStatement. Hence, MPMEncodedStatement objects are reusable at the object level, whereas other types of MPMPolicyStatement objects are reusable at the individual policy expression level.

The benefit of an MPMEncodedStatement is that it enables direct encoding of the text of the MPMPolicyStatement, without having the "overhead" of using other objects. However, note that while this operation is efficient, it does not reuse other MPMPolicyComponentStructure objects. Furthermore, its potential for reuse is reduced, as only MPMPolicies that can use the exact encoding of this clause can reuse this object.

Figure 5-21 shows the MPMEncodedStatement class.

Table 5-73 defines the attributes for this class.

Table 5-73: Attributes of the MPMEncodedStatement Class

Attribute Name	Description
mpmEncodedStatement Content : String[1..1]	This is a mandatory string attribute that defines the content of this particular MPMEncodedStatement object. It works with another class attribute, called mpmEncodedStatementEncoding, which defines how to interpret the value of this attribute (e.g. as a string or reference). These two attributes form a tuple, and together enable a machine to understand the syntax and value of this object instance.
mpmEncodedStatement Encoding : MPMEncodingType[1..1]	This is a mandatory enumerated non-negative integer attribute, and defines how to interpret the value of the mpmEncodedStatementContent class attribute. These two attributes form a tuple, and together enable a machine to understand the syntax and value of the encoded clause for the object instance of this class.
mpmEncodedStatement Response : Boolean[1..1]	This is a mandatory Boolean attribute that emulates a Boolean response of this statement, so that it may be combined with other subclasses of the MPMPolicyStatement that provide a Boolean value that defines their correctness and/or evaluation state. This enables this object to be used to construct more complex Boolean clauses.

Table 5-74 defines the operations for this class.

Table 5-74: Operations of the MPMEncodedStatement Class

Operation Name	Description
getMPMEncodedStmtContent() : String[1..1]	This operation returns the current value of the mpmEncodedStatementContent. This operation takes no input parameters. If this attribute does not have a value, then this operation **shall** return an error.
setMPMEncodedStmtContent (in newStatement : String[1..1])	This operation sets the value of the mpmEncodedStatementContent attribute. The value of the mpmEncodedStatementContent attribute **shall not** be empty or NULL.
getMPMEncodedStmtEncoding() : MPMEncodingType[1..1]	This operation returns the current value of the mpmEncodedStatementEncoding attribute. If this attribute does not have a value, then this operation **shall** return an error.
setMPMEncodedStmtEncoding(in newEncoding : MPMEncodingType[1..1])	This operation sets the value of the mpmEncodedStatementContent attribute.
getMPMEncodedStmtResponse() : Boolean[1..1]	This operation returns the current value of the mpmEncodedStatementResponse attribute. If this attribute does not have a value, then this operation **shall** return an error.
setMPMEncodedStmtResponse(in : newResponse : Boolean[1..1])	This operation sets the value of the mpmEncodedStatementResponse attribute.

Note that there are no operations that retrieve the number of MPMPolicyClause objects from an MPMEncodedStatement. This is because the MPMEncodedStatement object inherits the MPMStatementHasMPMPolicyClause aggregation from its superclass.

5.3.2.6.7.6 MPMTheorem Class Definition

An MPMTheorem refines an MPMPolicyStatement. It represents a logic statement that:

a) is not self-evident;

b) can be proven to be true.

The proof of a theorem is defined by the set of MPMPolicyClauses that it is associated with. Specifically, two or more MPMPremiseClause (see clause 5.3.2.6.7.13) objects **shall** have all been proven to be true, which makes the associated MPMConclusionClause (see clause 5.3.2.6.7.14) true. This is found by following the MPMStatementHasMPMPolicyClause aggregation (see clause 5.3.2.6.7.2).

An MPMTheorem object **shall** have previously been proven to be true in order for it to be used.

Figure 5-21 shows the MPMTheorem class.

Table 5-75 defines the attributes for this class.

Table 5-75: Attributes of the MPMTheorem Class

Attribute Name	Description
mpmTheorem-ProvesHypthesis : Boolean[1..1]	This is a mandatory Boolean attribute. If the value of this attribute is TRUE, then this MPMTheorem proves a previously unknown hypothesis. Otherwise, it is the result of previously known axioms and/or other theorems.
mpmTheorem-IsInvalid : Boolean[1..1]	This is a mandatory Boolean attribute. If the value of this attribute is TRUE, then this MPMTheorem was rendered incorrect due to one of its dependent axioms or theorems, that was previously true, being proved false. This requires revisiting all MPMPolicyStatements that depended on it. If the value of this attribute is FALSE, then the system **shall** set the mpmPolStmtExecStatus to ERROR for this MPMTheorem.

Table 5-76 defines the operations for this class.

Table 5-76: Operations of the MPMTheorem Class

Operation Name	Description
getMPMTheoremProves-Hypothesis() : Boolean[1..1]	This operation returns the current value of the mpmTheoremProvesHypothesis attribute. If this attribute does not have a value, then this operation **shall** return an error.
setMPMTheoremProves-Hypothesis(in newValue : Boolean[1..1])	This operation sets the value of the mpmTheoremProvesHypothesis attribute.
getMPMTheoremIsInvalid() : Boolean[1..1]	This operation returns the current value of the mpmTheoremIsInvalid attribute. If this attribute does not have a value, then this operation **shall** return an error.
setMPMTheoremProves-Hypothesis(newValue : Boolean[1..1])	This operation sets the value of the mpmTheoremIsInvalid attribute. The value of the mpmTheoremIsInvalid attribute **shall** be either true or false. If the value of this attribute is FALSE, then the system **shall** set the mpmPolStmtExecStatus to ERROR for this MPMTheorem.

Note that there are no operations that retrieve the number of MPMPolicyClause objects from an MPMTheorem. This is because the MPMTheorem object inherits the MPMStatementHasMPMPolicyClause aggregation from its superclass.

5.3.2.6.7.7 MPMAxiom Class Definition

An MPMAxiom is a type of MPMStatement that is taken to always be TRUE. Hence, it serves as a premise for other types of reasoning. Axioms are linked to MPMPolicyStatements using the MPMStatementHasMPMPolicyClause aggregation (see clause 5.3.2.6.7.2).

An MPMAxiom object **shall** be defined as true in order for it to be used.

An MPMAxiom **shall not** have to be proven to be true.

If an MPMAxiom is proven to be true, then:

1) the MPMAxiom **shall** be transformed into an MPMTheorem by copying its content into a new MPMTheorem object; and

2) the original MPMAxiom object **shall** be deleted.

Figure 5-21 shows the MPMAxiom class.

Table 5-77 defines the attributes for this class.

Table 5-77: Attributes of the MPMAxiom Class

Attribute Name	Description
mpmAxiomIsDisproved : Boolean[1..1]	This is a mandatory Boolean attribute. If the value of this attribute is TRUE, then this MPMAxiom has been proven FALSE. This requires revisiting all MPMPolicyStatements that depended on it. If the value of this attribute is FALSE, then the system **shall** set the mpmPolStmtExecStatus to ERROR for this MPMAxiom.

Table 5-78 defines the operations for this class.

Table 5-78: Operations of the MPMAxiom Class

Operation Name	Description
getMPMAxiomIsDisproved() : Boolean[1..1]	This operation returns the current value of the mpmAxiomIsDisproved attribute. If this attribute does not have a value, then this operation **shall** return an error.
setMPMAxiomIsDisproved (newValue : Boolean[1..1])	This operation sets the value of the mpmAxiomIsDisproved attribute. The value of the mpmAxiomIsDisproved attribute **shall** be either true or false. If the value of this attribute is FALSE, then the system **shall** set the mpmPolStmtExecStatus to ERROR for this MPMAxiom.

Note that there are no operations that retrieve the number of MPMPolicyClause objects from an MPMAxiom. This is because the MPMAxiom object inherits the MPMStatementHasMPMPolicyClause aggregation (see clause 5.3.2.6.7.2).

5.3.2.6.7.8 MPMPolicyClause Class Hierarchy

An MPMPolicyClause is a part of a statement. It defines all or part of the content of an MPMPolicyStatement. The decorator pattern is used to enable an extensible set of objects to "wrap" the MPMPolicyClause; this enables the contents of a MPMPolicyClause to be adjusted dynamically at runtime without affecting other objects. Put another way, this enables the contents of an MPMPolicyClause to be dynamically changed without affecting the MPMPolicyStatements that use it.

MPMPolicyClauses are objects in their own right, which facilitates their reuse. Different MPMPolicyClauses can be used with different MPMPolicyStatements to realize specific types of MPMPolicies.

An MPMPolicyClause, along with its subclasses, is shown in Figure 5-22.

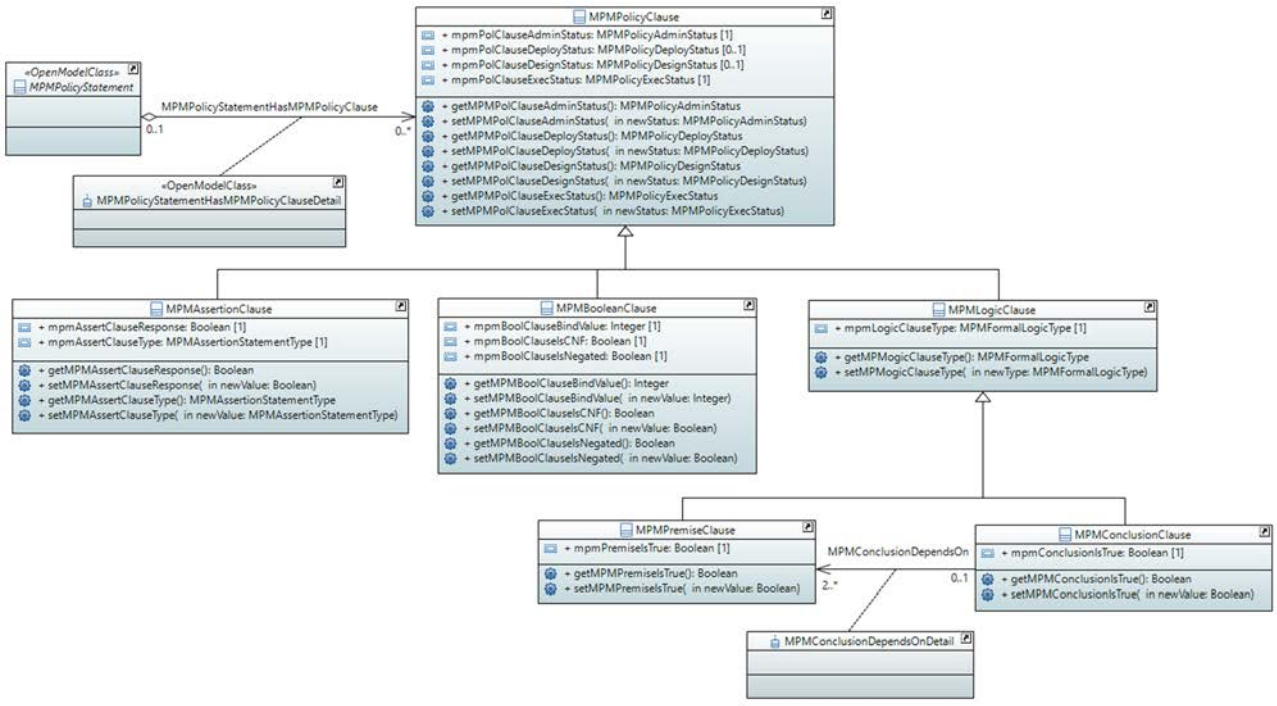

Figure 5-21: MPMPolicyClause Hierarchy

5.3.2.6.7.9 MPMPolicyClause Class Definition

An MPMPolicyClause is a mandatory abstract class whose subclasses define different types of clauses that are used to create the content for different types of MPMPolicyStatements. An MPMPolicyClause serves as a convenient aggregation point for assembling other objects that make up an MPMPolicyStatement.

The attributes of an MPMPolicyClause are shown in Table 5-79.

Table 5-79: Attributes of the MPMPolicyClause Class

Attribute Name	Description
mpmPolClause-AdminStatus : MPMPolicy-AdminStatus[1..1]	This is a mandatory enumerated non-negative integer attribute that defines the current administrative status of this particular MPMPolicyClause object.
mpmPolClause-DeployStatus : MPMPolicy-DeployStatus[1..1]	This is an optional enumerated, non-negative integer attribute that is used to indicate whether this MPMPolicyClause can or cannot be deployed by the policy management system. This attribute enables the policy manager to know which MPMClauses are currently deployed for a given MPMPolicyStatement, and may be useful for the policy execution system for planning the staging of MPMPolicies.
mpmPolClause-DesignStatus : MPMPolicy-DesignStatus[1..1]	This is an optional enumerated, non-negative integer whose value defines the current design status of this MPMPolicyClause object.
mpmPolClause-ExecStatus : MPMPolicy-ExecStatus[1..1]	This is a mandatory enumerated non-negative enumerated integer whose value defines the current execution status of this MPMPolicyClause object.

Table 5-80 defines the operations for this class.

Table 5-80: Operations of the MPMPolicyClause Class

Operation Name	Description
getMPMPolClauseAdmin-Status() : MPMPolicy-AdminStatus [1..1]	This operation returns the current value of the mpmPolClauseAdminStatus attribute. If the mpmPolClauseAdminStatus attribute does not have a value, then this operation **shall** return an error.
setMPMPolClauseAdmin-Status(in newStatus : MPMPolicyAdminStatus[1..1])	This operation sets the value of the mpmPolClauseAdminStatus attribute.
getMPMPolClauseDeploy-Status() : MPMPolicy-DeployStatus [1..1]	This operation returns the current value of the mpmPolClauseDeployStatus attribute. If the mpmPolClauseDeployStatus attribute does not have a value, then this operation **shall** return an error.
setMPMPolClauseDeploy-Status(in newStatus : MPMPolicyDeployStatus[1..1])	This operation sets the value of the mpmPolClauseDeployStatus attribute.
getMPMPolClauseDesign-Status() : MPMPolicy-DesignStatus [1..1]	This operation returns the current value of the mpmPolClauseDesignStatus attribute. If the mpmPolClauseDesignStatus attribute does not have a value, then this operation **shall** return an error.
setMPMPolClauseDesign-Status(in newStatus : MPMPolicyDesignStatus[1..1])	This operation sets the value of the mpmPolClauseDesignStatus attribute.
getMPMPolClauseExec-Status() : MPMPolicy-ExecStatus [1..1]	This operation returns the current value of the mpmPolClauseExecStatus attribute. If the mpmPolClauseExecStatus attribute does not have a value, then this operation **shall** return an error.
setMPMPolClauseExec-Status(in newStatus : MPMPolicyExecStatus[1..1])	This operation sets the value of the mpmPolClauseExecStatus attribute.

5.3.2.6.7.10 MPMAssertionClause Class Definition

An MPMAssertionClause is a predicate (i.e. a Boolean-valued function) that should evaluate to true at that point in the program's execution. An MPMAssertionClause may be used by different types of MPMPolicyStatements.

An MPMPolicyStatement **may** contain zero or more MPMAssertionClauses.

An MPMAssertionClause **may** be used with zero or more other MPMPolicyClauses.

Figure 5-22 shows the MPMPolicyClause class. The attributes of an MPMPolicyClause are shown in Table 5-81.

Table 5-81: Attributes of the MPMAssertionClause Class

Attribute Name	Description
mpmAssertClause-Response : Boolean[1..1]	This is a mandatory Boolean attribute that provides a Boolean response for this clause. This enables this MPMAssertionClause to be combined with other subclasses of an MPMPolicyClause and/or an MPMPolicyStatement that provide a Boolean value that defines the status as to their correctness and/or evaluation state. This enables this object to be used to construct more complex MPMPolicyClauses and MPMPolicyStatements.
mpmAssertClauseType : MPMAssertionStatementType[1..1]	This is a mandatory enumerated non-negative integer attribute that defines the composition of this particular MPMAssertionClause object.

Table 5-82 defines the operations for this class.

Table 5-82: Operations of the MPMAssertionClause Class

Operation Name	Description
getMPMAssertClause-Response() : Boolean[1..1]	This operation returns the current value of the mpmAssertClauseResponse attribute. If the mpmAssertClauseResponse attribute does not have a value, then this operation **shall** return an error.
setMPMAssertClause-Response(in newValue : Boolean[1..1])	This operation sets the value of the mpmAssertClauseResponse attribute.
getMPMAssertClauseType() : MPMAssertionStatementType[1..1]	This operation returns the current value of the mpmAssertClauseType attribute. If the mpmAssertClauseType attribute does not have a value, then this operation **shall** return an error.
setMPMAssertClauseType(in newValue : MPMAssertionStatementType[1..1])	This operation sets the value of the mpmAssertClauseType attribute.

5.3.2.6.7.11 MPMBooleanClause Class Definition

A Boolean clause has the canonical form of a {variable, operator, value} 3-tuple, which evaluates to either true or false. A Boolean clause may be made up of a combination of the Boolean constants true or false, along with Boolean-typed variables, Boolean-valued operators, and Boolean-valued functions.

Boolean clauses may be joined together using logical connectives (e.g. AND OR), forming more complex Boolean clauses. Individual terms in a Boolean clause, as well as an entire Boolean clause, **may** be negated.

An MPMPolicyStatement **may** contain zero or more MPMBooleanClauses.

Figure 5-22 shows the MPMBooleanClause class.

Table 5-83 defines the attributes for this class.

Table 5-83: Attributes of the MPMBooleanClause Class

Attribute Name	Description
mpmBoolClauseBindValue : Integer[1..1]	This is a mandatory array of positive integers that defines the order in which constituent terms bind to this MPMBooleanClause. For example, the Boolean expression "((A AND B) OR (C AND NOT (D OR E)))" has the following binding order: terms A and B have a bind value of 1; term C has a binding value of 2, and terms D and E have a binding value of 3. All values in this attribute **shall** be greater than 0.
mpmBoolClauseIsCNF : Boolean[1..1]	This is a mandatory Boolean attribute. If the value of this attribute is TRUE, then this MPMBooleanClause is in Conjunctive Normal Form. Otherwise, it is in Disjunctive Normal Form.
mpmBoolClauseIsNegated : Boolean[1..1]	This is a mandatory Boolean attribute. If the value of this attribute is TRUE, then this (entire) MPMBooleanClause is negated.

Table 5-84 defines the attributes for this class.

Table 5-84: Operations of the MPMBooleanClause Class

Operation Name	Description
getMPMBoolClauseBindValue() : Integer[1..1]	This operation returns the current value of the mpmBoolStmtBindValue attribute, which is an array of positive integers. If the mpmBoolStmtBindValue attribute does not have a value, then this operation **shall** return an error.
setMPMBoolClauseBind-Value(in newValue : Integer[1..1])	This operation sets the value of the mpmBoolClauseBindValue attribute.. The value of the mpmBoolClauseBindValue attribute **shall** be a positive (non-zero) integer.
getMPMBoolClauseIsCNF() : Boolean[1..1]	This operation returns the current value of the mpmBoolClauseIsCNF attribute. If the mpmBoolClauseIsCNF attribute does not have a value, then this operation **shall** return an error.
setMPMBoolClauseBind-Value(in newValue : Boolean[1..1])	This operation sets the value of the mpmBoolClauseIsCNF attribute. The value of the mpmBoolClauseIsCNF attribute **shall** be a Boolean value.
getMPMBoolClauseIsNegated() : Boolean[1..1]	This operation returns the current value of the mpmBoolClauseIsNegated attribute. If the mpmBoolClauseIsNegated attribute does not have a value, then this operation **shall** return an error.
setMPMBoolClauseIsNegated (in newValue : Boolean[1..1])	This operation sets the value of the mpmBoolClauseIsNegated attribute. This operation takes a single input parameter, called newValue, which defines the new value for the mpmBoolClauseIsNegated attribute. The value of the mpmBoolClauseIsNegated attribute **shall** be a Boolean value.

5.3.2.6.7.12 MPMLogicClause Class Definition

An MPMLogicClause is an abstract class that is the superclass for different types of clauses that are used in declarative logic policies. This type of clause is limited to being used with the following MPMPolicyStatements: MPMAssertionStatements, MPMTheorems, and MPMAxioms.

An MPMAssertionStatement **may** contain zero or more MPMLogicClauses.

An MPMTheorem **may** contain zero or more MPMLogicClauses.

An MPMAxiom **may** contain zero or more MPMLogicClauses.

Figure 5-22 shows the MPMLogicClause class and its subclasses. Table 5-85 defines the attributes for this class.

Table 5-85: Attributes of the MPMLogicClause Class

Attribute Name	Description
mpmLogicClauseType : MPMFormalLogicType[1..1]	This is a mandatory enumerated non-zero integer attribute that defines the formal logic system that this particular MPMLogicClause uses.

Table 5-86 defines the operations for this class.

Table 5-86: Operations of the MPMLogicClause Class

Operation Name	Description
getMPMogicClauseType() : MPMFormalLogicType[1..1]	This operation returns the current value of the mpmLogicClauseType attribute. If this attribute does not have a value, then this operation **shall** return an error.
setMPMogicClauseType(in newType : MPMFormalLogicType[1..1]	This operation sets the value of the mpmLogicClauseType attribute.

5.3.2.6.7.13 MPMPremiseClause Class Definition

An MPMPremiseClause is a declarative logic clause that is used to justify a conclusion (represented by an MPMConclusionClause; see clause 5.3.2.6.7.14).

Figure 5-22 shows the MPMPremiseClause class and its subclasses. Table 5-87 defines the attributes for this class.

Table 5-87: Attributes of the MPMPremiseClause Class

Attribute Name	Description
mpmPremiseIsTrue : Boolean[1..1]	This is a mandatory Boolean attribute. If the value of this attribute is TRUE, then this MPMPremiseClause has been proven TRUE.

Table 5-88 defines the operations for this class.

Table 5-88: Operations of the MPMPremiseClause Class

Operation Name	Description
getMPMPremiseIsTrue() : Boolean[1..1]	This operation returns the current value of the mpmPremiseIsTrue attribute. If this attribute does not have a value, then this operation **shall** return an error.
setMPMPremiseIsTrue (in newValue : Boolean[1..1])	This operation sets the value of the mpmPremiseIsTrue attribute.

This class participates in a single association, called MPMConclusionDependsOn. This is described in clause 5.3.2.6.7.14.

5.3.2.6.7.14 MPMConclusionClause Class Definition

An MPMConclusionClause is a declarative logic clause that is entailed (i.e. logically proves to be true) from its set of associated MPMPremiseClauses.

An MPMConclusionClause object **shall** be associated with two or more MPMPremiseClause objects.

An MPMConclusionClause **shall** evaluate to TRUE when all of its associated MPMPremiseClause objects have evaluated to TRUE.

Figure 5-22 shows the MPMConclusionClause class and its subclasses. Table 5-89 defines the operations for this class.

Table 5-89: Attributes of the MPMConclusionClause Class

Attribute Name	Description
mpmConclusionIs-True : Boolean[1..1]	This is a mandatory Boolean attribute. If the value of this attribute is TRUE, then this MPMConclusionClause has been proven TRUE.

Table 5-90 defines the operations for this class.

Table 5-90: Operations of the MPMConclusionClause Class

Operation Name	Description
getMPMConclusionIsTrue() : Boolean[1..1]	This operation returns the current value of the mpmConclusionIsTrue attribute. If this attribute does not have a value, then this operation **shall** return an error.
setMPMConclusionIsTrue(in newValue : Boolean[1..1])	This operation sets the value of the mpmConclusionIsTrue attribute.

This class defines a single optional association, called MPMConclusionDepends on, as shown in Figure 5-22. The MPMConclusionDependsOn association defines the set of MPMPremiseClause objects that are attached to this particular MPMConclusionClause object. The semantics of this association are defined by the MPMConclusionDependsOnDetail association class.

The multiplicity of this association is 0..1 - 2..n. This means that it is an optional association (i.e. the "0" part of the 0..1 cardinality). If this association is instantiated (i.e. the "1" part of the 0..1 cardinality), then two or more MPMPremiseClause objects are associated with this particular MPMConclusionClause object. Specifically, this means that the MPMConclusionClause *depends* on the two or more MPMPremiseClause objects. The 2..* cardinality prescribes a minimum number (2) of MPMPremiseClause objects to be associated with this particular MPMConclusionClause object. The semantics of this association are defined by the MPMConclusionDependsOnDetail association class. This enables the management system to control which set of concrete subclasses of MPMPremiseClause objects can be associated with which types of MPMConclusionClause objects.

The MPMConclusionDependsOnDetail is a concrete association class, and defines the semantics of the MPMConclusionDependsOn association. The attributes and relationships of this class can be used to define which MPMPremiseClause objects can be associated with which particular set of MPMConclusionClause objects. These semantics can be further enhanced by using the Policy Pattern to define policy rules that constrain which part objects (i.e. MPMPremiseClause) are attached to which MPMConclusionClause object. The MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

5.3.2.6.7.15 MPMPolicyComponentDecorator Class Hierarchy

The Decorator Pattern [i.7] is a design pattern that allows behaviour to dynamically be added to an object, without affecting the behaviour of other objects from the same class. More specifically, this pattern enables all or part of one object to wrap another object. In effect, this means that the decorated object may intercept a call to the object it is wrapping, and insert attributes or execute operations before and/or after the wrapped object executes.

Hence, the decorator pattern provides a flexible alternative to subclassing for extending functionality where different behaviours are required (e.g. behaviour that is dependent on context). In addition, subclassing statically defines the characteristics and behaviour of an object at compile time, whereas the decorator pattern can change the characteristics and behaviour of an object at runtime.

Figure 5-23 shows the MPMPolicyComponentDecorator class and its subclasses and relationships with respect to all subclasses of the MPMPolicyComponent class (its superclass).

5.3.2.6.7.16 MPMPolicyComponentDecorator Class Definition

This is a mandatory class that is used to implement the decorator pattern. This means that any concrete subclass of MPMPolicyComponentDecorator can wrap any concrete subclass of MPMPolicyStatement and/or MPMPolicyClause.

The attachment of different MPMPolicyComponentDecorator objects **shall** be used to change the syntax, semantics, and/or behaviour of a given MPMPolicyComponentStructure object.

Figure 5-23 shows the attributes and operations for the MPMPolicyComponentDecorator class.

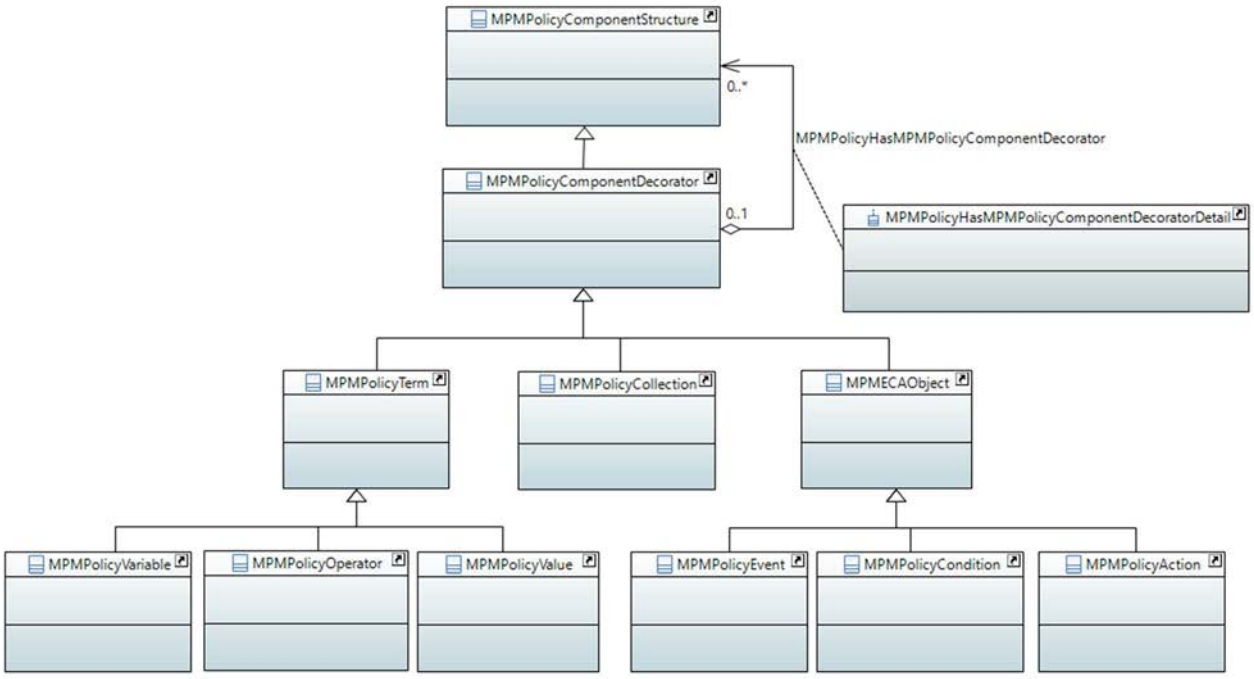

Figure 5-22: MPMPolicyComponentDecorator Attributes and Operations

Table 5-91 defines the attributes for this class.

Table 5-91: Attributes of the MPMPolicyComponentDecorator Class

Attribute Name	Description
mpmPolCompDecConstraint : MPMPolCompDecConstraint[1..1]	This is a mandatory non-negative enumerated integer attribute that defines the language used, if any, that this MPMPolicyComponentDecorator subclass uses to constrain the object that it is wrapping. A default value of 2 (NONE) **may** be defined.
mpmPolCompDecWrap : MPMPolCompDecWrap[1..1]	This is an optional attribute that defines if this decorated object should be wrapped before and/or after the wrapped object is executed.

Table 5-92 defines the attributes for this class.

Table 5-92: Attributes of the MPMPolicyComponentDecorator Class

Operation Name	Description
getMPMPolCompDecConstraint() : MPMPolCompDecConstraint[1..1]	This operation returns the current value of the mpmPolCompDecConstraint attribute. If this attribute does not have a value, then this operation **shall** return an error.
setMPMPolCompDecConstraint(in newValue : MPMPolCompDecConstraint[1..1])	This operation sets the value of the mpmPolCompDecConstraint attribute.
getMPMPolCompDecWrap() : MPMPolCompDecWrap[1..1]	This operation returns the current value of the mpmPolCompDecValue attribute. If this attribute does not have a value, then this operation **shall** return an error.
setMPMPolCompDecWrap(in newValue : MPMPolCompDecWrap[1..1])	This operation sets the value of the mpmPolCompDecValue attribute.

This class participates in a single optional aggregation, called MPMPolicyHasMPMPolicyComponentDecorator. This aggregation defines the set of MPMPolicyComponentDecorator objects that wrap, or decorate, this particular MPMPolicyComponentStructure object. An MPMPolicyComponentStructure object may be decorated by zero or more MPMPolicyComponentDecorator objects. The semantics of this aggregation are defined by the MPMPolicyHasMPMPolicyComponentDecoratorDetail association class.

The multiplicity of this aggregation is 0..1 - 0..n. If this aggregation is instantiated (i.e. the "1" part of the 0..1 cardinality), then zero or more MPMPolicyComponentDecorator objects can decorate this particular MPMPolicyComponentStructure object. The 0..* cardinality enables an MPMPolicyComponentStructure object to be defined without having to define an associated MPMPolicyComponentDecorator object for it to decorate.

The MPMPolicyHasMPMPolicyComponentDecoratorDetail object is a concrete association class, and defines the semantics of the MPMPolicyHasMPMPolicyComponentDecorator aggregation. The attributes and relationships of this class can be used to define which MPMPolicyComponentDecorator objects can decorate this particular set of MPMPolicyComponentStructure objects. These semantics can be further enhanced by using the Policy Pattern to define policy rules that constrain which part objects (i.e. MPMPolicyComponentDecorator) are attached to which object. The MCMPolicyStructure is an abstract class that is the superclass of imperative, declarative, and intent policy rules. See Figure 3 of ETSI GS ENI 005 [3] for an exemplary illustration of the Policy Pattern.

5.3.2.6.7.17 MPMPolicyTerm

This is a mandatory abstract class that subclasses MPMPolicyComponentDecorator. MPMPolicyTerm is the parent of MPMPolicy objects that can be used to define a standard way to test or set the value of a variable. It does this by defining a 3-tuple, whose canonical form is {variable, operator, value}. Each element of the 3-tuple is defined by a concrete subclass of the appropriate type (i.e. MPMPolicyVariable, MPMPolicyOperator, and MPMPolicyValue classes, respectively). Since it is a type of MPMPolicyComponentDecorator, it can be attached to (or detached from) an MPMPolicyStatement or MPMPolicyClause object to change their semantics dynamically at runtime.

For event and condition clauses and statements, this is typically written as:

<variable> <operator> <value>.

For action clauses and statements, this is typically written as:

<operator> <variable> <value>.

Generic test and set expressions do not have to only use objects that are subclasses of MPMPolicyTerm. The utility of the subclasses of MPMPolicyTerm is in the ability of its subclasses to define a generic framework for implementing get and set expressions directly from the model. This enables a dynamic programming environment to construct get and set expressions at runtime.

Figure 5-24 shows the attributes and operations of MPMPolicyTerm and its subclasses.

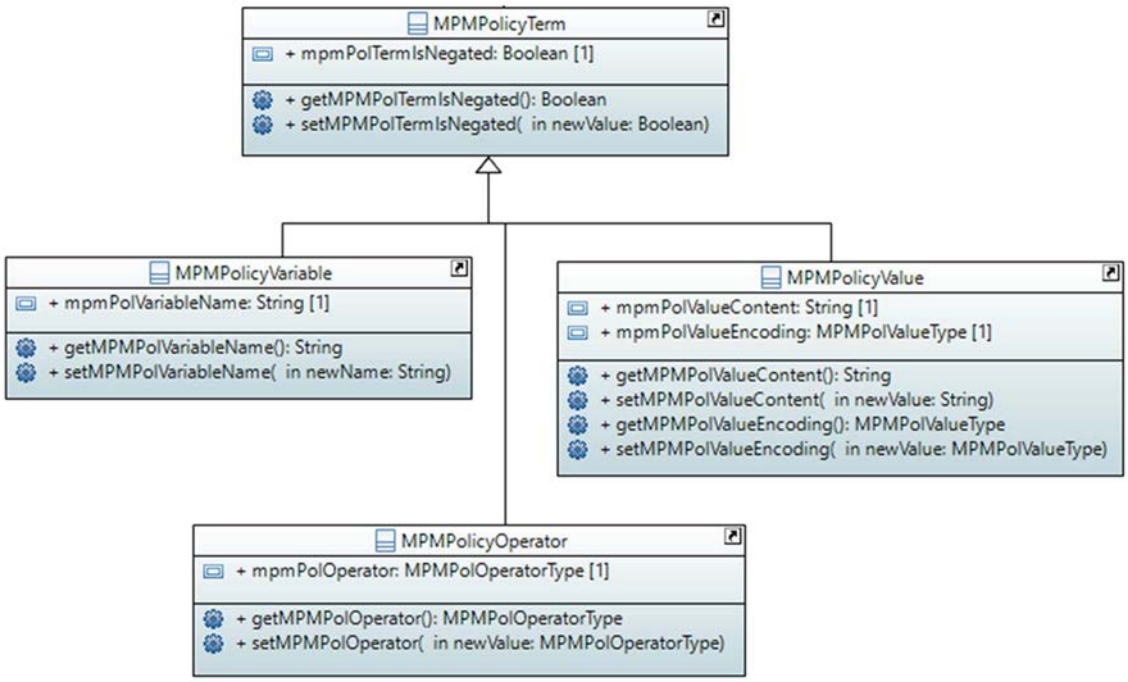

Figure 5-23: MPMPolicyTerm Class Hierarchy Attributes and Operations

Table 5-93 defines the attributes for this class.

Table 5-93: Attributes of the MPMPolicyTerm Class

Attribute Name	Description
mpmPolTermIsNegated : Boolean[1..1]	This is a mandatory Boolean attribute. If the value of this attribute is TRUE, then this (entire) MPMTerm is negated.

Table 5-94 defines the operations for this class.

Table 5-94: Operations of the MPMPolicyTerm Class

Operation Name	Description
getMPMPolTermIsNegated() : Boolean[1..1]	This operation returns the current value of the mpmPolTermIsNegated attribute. If the mpmPolTermIsNegated attribute does not have a value, then this operation **shall** return an error.
setMPMPolTermIsNegated(in newValue : Boolean[1..1])	This operation sets the value of the mpmPolTermIsNegated attribute.

5.3.2.6.7.18 MPMPolicyVariable Class Definition

This is a mandatory concrete class that defines information that forms a part of an MPMPolicyClause or MPMPolicyStatement. It specifies a concept or attribute that represents a variable that should be compared to a value using a particular type of operator.

The value of an MPMPolicyVariable class **may** be able to be changed dynamically at runtime using the decorator pattern.

The value of an MPMPolicyVariable object is typically compared to the value of an MPMPolicyValue object using the type of operator defined in a MPMPolicyOperator object. However, other objects **may** be used instead of the MPMPolicyOperator and MPMPolicyValue objects, and other operators **may** be defined in addition to those defined in the MPMPolicyOperator class.

MPMPolicyVariables are used to abstract the representation of an MPMPolicyClause (or MPMPolicyStatement) from its implementation. This means that some MPMPolicyVariable objects need to be restricted in the values and/or the data type that they may be assigned. For example, port numbers cannot be negative, and they cannot be floating-point numbers. These and other constraints may be defined in three different ways:

1) Use MPMPolicyComponentDecorator attributes to constrain just that individual object.

2) Use the MPMPolicyClauseHasDecoratorDetail association class attributes to constrain the relationship between the concrete subclass of the MPMPolicyClause (or MPMPolicyStatement) and the concrete subclass of the MPMPolicyVariable class.

3) Use OCL statements to define the restriction.

Figure 5-24 shows the MPMPolicyVariable class and its subclasses.

Table 5-95 defines the attributes for this class.

Table 5-95: Attributes of the MPMPolicyVariable Class

Attribute Name	Description
mpmPolVariableName : String[1..1]	This is a mandatory string attribute that contains the name of this MPMPolicyVariable.

Table 5-96 defines the operations for this class.

Table 5-96: Operations of the MPMPolicyVariable Class

Operation Name	Description
getMPMPolVariableName() : String[1..1]	This operation returns the current value of the mpmPolVariableName attribute. If the mpmPolVariableName attribute does not have a value, then this operation **shall** return an error.
setMPMPolVariableName(in newName : String[1..1])	This operation sets the value of the mpmPolVariableName attribute. The value of the mpmPolVariableName attribute **shall not** be empty or NULL.

5.3.2.6.7.19 MPMPolicyOperator Class Definition

This is a mandatory concrete class for modeling different types of operators that are used in an MPMPolicyClause or MPMPolicyStatement.

The restriction of the type of operator used in an MPMPolicyClause or MPMPolicyStatement constrains the semantics that can be expressed in that MPMPolicyClause or MPMPolicyStatement. It is typically, but does not have to be, used with MPMPolicyVariable and MPMPolicyValue objects to form an MPMPolicyClause or MPMPolicyStatement.

The value of an MPMPolicyVariable object is usually compared to the value of an MPMPolicyValue object using the type of operator defined in a MPMPolicyOperator object. However, other objects **may** be used instead of the MPMPolicyOperator and MPMPolicyValue objects.

In addition, and other operators **may** be defined in addition to those defined in the MPMPolicyOperator class.

Since this is a subclass of the MPMPolicyComponentDecorator class, its value may be able to be changed dynamically at runtime using the decorator pattern.

The value of an MPMPolicyOperator class **may** be able to be changed dynamically at runtime using the decorator pattern.

Figure 5-24 shows the MPMPolicyOperator class and its subclasses.

Table 5-97 defines the attributes for this class.

Table 5-97: Attributes of the MPMPolicyOperator Class

Attribute Name	Description
mpmPolOperator : MPMPolOperatorType[1..1]	This is a mandatory enumerated non-negative integer attribute.

Table 5-98 defines the operations for this class.

Table 5-98: Operations of the MPMPolicyOperator Class

Operation Name	Description
getMPMPolOperator() : MPMPolOperatorType[1..1]	This operation returns the current value of the mpmPolOperator attribute. If the mpmPolOperator attribute does not have a value, then this operation **shall** return an error.
setMPMPolOperator(in newValue : MPMPolOperatorType[1..1])	This operation sets the value of the mpmPolOperator attribute.

5.3.2.6.7.20 MPMPolicyValue Class Definition

The MPMPolicyValue class is a mandatory concrete class for modeling different types of values and constants that occur in an MPMPolicyClause or an MPMPolicyStatement.

MPMPolicyValue objects are used to abstract the representation of an MPMPolicyClause or an MPMPolicyStatement from its implementation. Therefore, the design of the MPMPolicyValue object depends on two important factors. First, just as with MPMPolicyVariable objects, some types of MPMPolicyValue objects are restricted in the values and/or the data type that they may be assigned. Second, there is a high likelihood that specific applications will need to use their own variables that have specific meaning to a particular application.

In general, there are three ways to apply constraints to an object instance of an MPMPolicyValue object:

1) Use MPMPolicyClauseComponentDecorator attributes to constrain just that individual object.

2) Use the MPMPolicyClauseHasDecoratorDetail association class attributes to constrain the relationship between the concrete subclass of the MPMPolicyClause (or MPMPolicyStatement) and the concrete subclass of the MPMPolicyVariable class.

3) Use OCL statements to define the restriction.

The value of an MPPolicyValue object is typically compared to the value of an MPMPolicyVariable object using the type of operator defined in an MPMPolicyOperator object. However, other objects may be used instead of an MPMPolicyVariable object, and other operators may be defined in addition to those defined in the MPMPolicyOperator class.

Since this is a subclass of the MPMPolicyComponentDecorator class, its value **may** be able to be changed dynamically at runtime using the decorator pattern.

The value of an MPMPolicyValue class **may** be able to be changed dynamically at runtime using the decorator pattern.

Figure 5-24 shows the MPMPolicyValue class and its subclasses.

Table 5-99 defines the attributes for this class.

Table 5-99: Attributes of the MPMPolicyValue Class

Attribute Name	Description
mpmPolValueContent: String[1..1]	This is a mandatory string attribute that defines the value of this MPMPolicyValue object.
mpmPolValueEncoding: MPMPolValueType[1..1]	This is a mandatory enumerated non-negative integer attribute that defines the datatype of the mpmPolValueContent class attribute.

Table 5-100 defines the operations for this class.

Table 5-100: Operations of the MPMPolicyValue Class

Operation Name	Description
getMPMPolValueContent() : String[1..1]	This operation returns the current value of the mpmPolValueContent attribute. If the mpmPolValueContent attribute does not have a value, then this operation **shall** return an error.
setMPMPolValueContent(in newValue : String[1..1])	This operation sets the value of the mpmPolValueContent attribute. The value of the mpmPolValueContent attribute **shall not** be empty.
getMPMPolValueEncoding() : MPMPolValueType [1..1]	This operation returns the current value of the mpmPolValueContent attribute. If the mpmPolValueContent attribute does not have a value, then this operation **shall** return an error.
setMPMPolValueContent(in newValue : MPMPolValue-Type [1..1])	This operation sets the value of the mpmPolValueContent attribute.

5.3.2.6.7.21 MPMECAObject Hierarchy

The MPMECAObject abstract class is used to define three concrete subclasses, one each to represent the concepts of reusable events, conditions, and actions. They are called MPMPolicyEvent, MPMPolicyCondition, and MPMPolicyAction, respectively.

Figure 5-25 shows the MPMECAObject class and its subclasses.

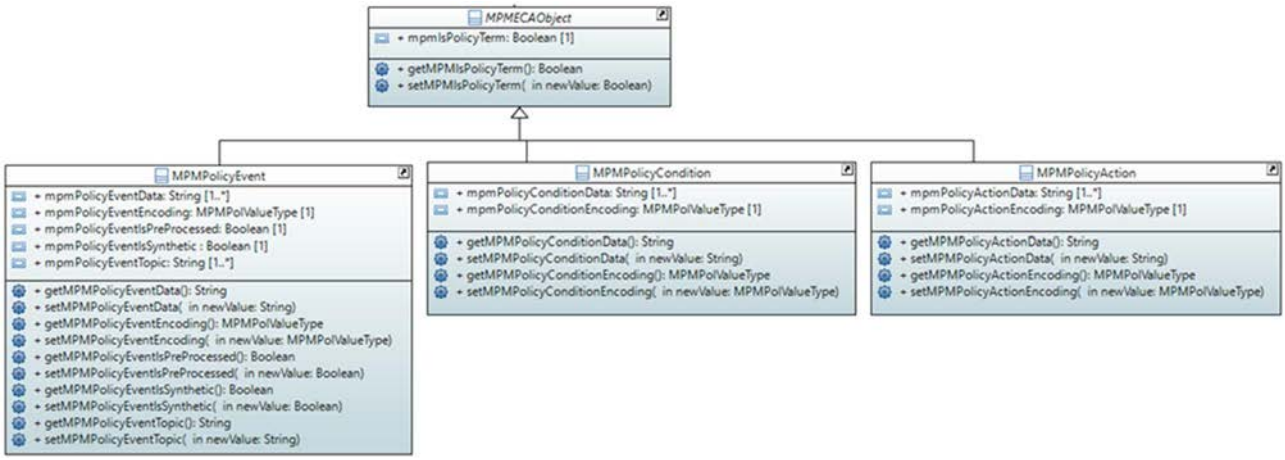

Figure 5-24: MPMECAObject Hierarchy

MPMECAObjects provide two different ways to construct MPMPolicyClauses. The first is for the MPMECAObject to be used as either an MPMPolicyVariable or an MPM PolicyValue, and the second is for the MPMECAObject to contain the entire clause text for an MPMPolicyVariable or an MPMPolicyValue. For example, suppose it is desired to define a policy condition clause with the text "queueDepth > 10". Two approaches could satisfy this as follows:

1) Approach #1 (canonical form):

   - MPMPolicyCondition.mpmPolicyConditionData contains the text 'queueDepth'.
   - MPMPolicyOperator.mpmPolOpType is set to '1' (greater than).
   - MPMPolicyValue.mpmPolValContent is set to '10'.

2) Approach #2 (MPMECAComponent represents the entire clause):

   - MPMPolicyCondition.mpmPolicyConditionData contains the text 'queueDepth > 10'.

In both of the above approaches, MPMPolicyCondition.mpmPolicyConditionEncoding is set to '1' (string).

The main advantage of MPMECAObjects is that they provide a machine-processable mechanism for defining MPMPolicyClauses at runtime.

5.3.2.6.7.22 MPMECAObject Class Definition

This is a mandatory abstract class that defines three concrete subclasses, one each to represent the concepts of reusable events, conditions, and actions. They are called MPMPolicyEvent, MPMPolicyCondition, and MPMPolicyAction, respectively.

Figure 5-25 shows the MPMECAObject class and its subclasses.

Table 5-101 defines the attributes for this class.

Table 5-101: Attributes of the MPMECAObject Class

Attribute Name	Description
mpmIsPolicyTerm : Boolean[1..1]	This is a mandatory Boolean attribute. If the value of this attribute is TRUE, then this MPMECAObject is used as the value of an MPMPolicyTerm to construct an MPMPolicyClause (this is approach #1 in clause 5.3.2.6.7.21). If the value of this attribute is FALSE, then this MPMECAObject contains the text of the entire corresponding MPMPolicyClause (this is approach #2 in the above example).

Table 5-102 defines the operations for this class.

Table 5-102: Operations of the MPMECAObject Class

Operation Name	Description
getMPMIsPolicyTerm() : Boolean[1..1]	This operation returns the current value of the mpmIsPolicyTerm attribute. If the mpmIsPolicyTerm attribute does not have a value, then this operation **shall** return an error.
getMPMIsPolicyTerm (in newValue : Boolean[1..1])	This operation sets the value of the mpmIsPolicyTerm attribute.

5.3.2.6.7.23 MPMPolicyEvent Class Definition

This is a mandatory concrete class that represents the concept of an Event that is applicable to a policy management system. Such an Event is defined as anything of importance to the management system (e.g. a change in the system being managed and/or its environment) occurring at a specific point in time. An MPMPolicyEvent is a subclass of the MPMPolicyComponentDecorator class, which enables the MPMPolicyEvent object to wrap any concrete subclass of MPMPolicyComponentStructure, such as the concrete subclasses of MPMPolicyStatement and MPMPolicyClause.

Instances of this class are not themselves events. Rather, instances of this class appear in MPMPolicyClause objects to describe what types of events the MPMPolicy is triggered by and/or uses. Clause 5.2.3.3 of the present document defines an ENIEvent.

An MPMPolicyEvent object **may** refer to an ENIEvent object.

Information from events that trigger MPMPolicies need to be made available for use in condition and action clauses, as well as in appropriate decorator objects. Application-specific subclasses (such as one for using YANG notifications as policy events) need to define how the information from the environment or event is used to trigger the evaluation of the MPMPolicyCondition subclass.

If an MPMPolicyEvent class is extended by subclassing, then that subclass **should** define how the set of events represented by the MPMPolicyEvent subclass triggers MPMPolicy evaluation.

Figure 5-25 shows the MPMPolicyEvent class and its subclasses. Table 5-103 defines the attributes for this class.

Table 5-103: Operations of the MPMPolicyEvent Class

Attribute Name	Description
mpmPolicyEventData : String[1..*]	This is a mandatory attribute that defines an array of strings. Each string in the array represents an attribute name and value of an Event object. The format of each string is defined as a {name:value} tuple. The 'name' part is the name of the MPMPolicyEvent attribute, and the 'value' part is the value of that attribute. For example, if the value of this attribute is: {('startTime':'08:00'), ('endTime':'17:00'), ('date':'2016-05-11'), ('timeZone':'-08:00')} then this attribute contains four properties, called startTime, endTime, date, and timeZone, whose values are 0800, 1700, May 11 2016, and Pacific Standard Time, respectively. This attribute works with another class attribute, called mpmPolicyEventEncoding, which defines how to interpret this attribute. These two attributes form a tuple, and together enable a machine to understand the syntax and value of the data carried by the object instance of this class. This attribute **may** encode attributes and values of an ENIEvent.
mpmPolicyEventEncoding : PolValueType[1..1]	This is a mandatory non-zero enumerated integer attribute, and defines how to interpret the mpmPolicyEventData class attribute. These two attributes form a tuple, and together enable a machine to understand the syntax and value of the data carried by the object instance of this class.
mpmPolicyEventIsPre-Processed : Boolean[1..1]	This is an optional Boolean attribute. If the value of this attribute is TRUE, then this MPMPolicyEvent has been pre-processed by an external entity, such as an Event Service Bus, before it was received by the Policy Management System.
mpmPolicyEventIsSynthetic : Boolean[1..1]	This is an optional Boolean attribute. If the value of this attribute is TRUE, then this MPMPolicyEvent has been produced by the Policy Management System. If the value of this attribute is FALSE, then this MPMPolicyEvent has been produced by an entity in the system being managed.
mpmPolicyEventTopic : String[1..*]	This is a mandatory array of string attributes, and contains the subject(s) that describe the nature of this PolicyEvent.

Table 5-104 defines the operations for this class.

Table 5-104: Operations of the MPMPolicyEvent Class

Operation Name	Description
getMPMPolicyEventData() : String[1..*]	This operation returns the current value of the mpmPolicyEventData attribute, which is an array of one or more strings. If the mpmPolicyEventData attribute does not have a value, then this operation **should** return a NULL string.
setMPMPolicyEventData(in newValue : String[1..*])	This operation sets the value of the mpmPolicyEventData attribute. The value of the mpmPolicyEventData attribute **shall not** be an empty string.
getMPMPolicyEventEncoding () : MPMPolValueType[1..1]	This operation returns the current value of the mpmPolicyEventEncoding attribute. If this attribute does not have a value, then this operation **shall** return an error.
setMPMPolicyEventEncoding (in newValue : MPMPolValueType[1..1])	This operation sets the value of the mpmPolicyEventEncoding attribute.
getMPMPolicyEventIsPrePro-cessed() : Boolean[1..1]	This operation returns the current value of the mpmPolicyEventIsPreProcessed attribute, which is a Boolean attribute. If the mpmPolicyEventIsPreProcessed attribute does not have a value, then this operation **shall** return an error.
setMPMPolicyEventIsPrePro-cessed(in newValue : Boolean[1..1])	This operation sets the value of the mpmPolicyEventIsPreProcessed attribute.
getMPMPolicyEventIsSynthe-tic() : Boolean[1..1]	This operation returns the current value of the mpmPolicyEventIsSynthetic attribute, which is a Boolean attribute. If the mpmPolicyEventIsSynthetic attribute does not have a value, then this operation **shall** return an error.
setMPMPolicyEventIsSynthe-tic(in newValue : Boolean[1..1])	This operation sets the value of the mpmPolicyEventIsSynthetic attribute.

Operation Name	Description
getMPMPolicyEventTopic() : String[1..*]	This operation returns the current value of the mpmPolicyEventTopic attribute, which is an array of one or more strings. If the mpmPolicyEventData attribute does not have a value, then this operation **shall** return an error.
setMPMPolicyEventTopic(in newValue : String[1..*])	This operation sets the value of the mpmPolicyEventTopic attribute. The value of the mpmPolicyEventTopic attribute **shall not** be an empty string.

5.3.2.6.7.24 MPMPolicyCondition Class Definition

This is a mandatory concrete class that represents the concept of a Condition that will determine whether or not the set of Actions in this MPMPolicy should be executed or not.

MPMPolicyCondition objects can be used as part of an MPMPolicyStatement or an MPMPolicyClause object. An MPMPolicyCondition is a subclass of the MPMPolicyComponentDecorator class, which enables the MPMPolicyCondition object to wrap any concrete subclass of MPMPolicyComponentStructure, such as the concrete subclasses of MPMPolicyStatement and MPMPolicyClause.

Application-specific subclasses of this class (such as one for processing YANG) need to define how the information from the environment is used by this subclass.

If the MPMPolicyCondition class is extended by subclassing, then it **should** define how it uses information from the managed environment to evaluate conditions in MPMPolicies.

Figure 5-25 shows the MPMPolicyCondition class and its subclasses.

Table 5-105 defines the attributes for this class.

Table 5-105: Attributes of the MPMPolicyCondition Class

Attribute Name	Description
mpmPolicyConditionData : String[1..*]	This is a mandatory attribute that defines an array of strings. Each string in the array represents an attribute name and value of an object that serves as a condition. The format of each string is defined as a {name:value} tuple. The 'name' part is the name of the MPMPolicyCondition attribute, and the 'value' part is the value of that attribute. For example, if the value of this attribute is: {('sourcePort':'8080'), ('destPort':'8080')} then this attribute contains two properties, called sourcePort and destPort, whose values are both "8080". This attribute works with another class attribute, called mpmPolicyConditionEncoding, which defines how to interpret this attribute. These two attributes form a tuple, and together enable a machine to understand the syntax and value of the data carried by the object instance of this class.
mpmPolicyConditionEncoding : PolValueType[1..1]	This is a mandatory non-zero enumerated integer attribute, and defines how to interpret the mpmPolicyConditionData class attribute. These two attributes form a tuple, and together enable a machine to understand the syntax and value of the data carried by the object instance of this class. Allowed values are defined in the MPMPolValueType enumeration.

Table 5-106 defines the operations for this class.

Table 5-106: Operations of the MPMPolicyCondition Class

Operation Name	Description
getMPMPolicyConditionData() : String[1..*]	This operation returns the current value of the mpmPolicyConditionData attribute. If the mpmPolicyConditionData attribute do7es not have a value, then this operation **shall** return an error.
setMPMPolicyConditionData(in newValue : String[1..*])	This operation sets the value of the mpmPolicyConditionData attribute. The value of the mpmPolicyConditionData attribute **shall not** be an empty string.
getMPMPolicyConditionEncoding() : MPMPolValueType[1..1]	This operation returns the current value of the mpmPolicyConditionEncoding attribute. If this attribute does not have a value, then this operation **shall** return an error.
setMPMPolicyConditionEncoding(in newValue : MPMPolValueType[1..1])	This operation sets the value of the mpmPolicyConditionEncoding attribute.

5.3.2.6.7.25 MPMPolicyAction Class Definition

This is a mandatory concrete class that represents the concept of an Action, which is a part of an MPMECAPolicy. The Action may be executed when both the event and the condition clauses of its owning MPMECAPolicy evaluate to true.

An MPMPolicyAction is a subclass of the MPMPolicyComponentDecorator class, which enables the MPMPolicyEvent object to wrap any concrete subclass of MPMPolicyComponentStructure, such as the concrete subclasses of MPMPolicyStatement and MPMPolicyClause.

Application-specific subclasses of this class (such as one for processing YANG) need to define how the information from the environment is used by this subclass.

If the MPMPolicyAction class is extended by subclassing, then it **should** define how it uses information from the managed environment to execute actions in MPMPolicies.

The execution of an action is determined by its MPMECAPolicy container, and any applicable MPMPolicyMetadata objects that are attached to that MPMECAPolicy container.

MPMPolicyAction objects **may** be used in three different ways:

1) As part of an MPMPolicyClause (e.g. var = MPMPolicyAction.mpmPolicyActionData).

2) As a standalone MPMPolicyClause (e.g. the mpmPolicyActionData attribute contains text that defines the entire action clause, and the mpmPolicyActionEncoding attribute defines the datatype of the mpmPolicyActionData attribute).

3) To invoke one or more MPMPolicyActions in a different MPMECAPolicy.

In the third case, note that this is NOT invoking a different MPMECAPolicy, but rather, invoking an MPMPolicyAction that is contained in a different MPMECAPolicy.

The problem with an MPMECAPolicy calling MPMECAPolicy is best illustrated with the following example:

MPMECAPolicy A is currently executing
MPMPolicyAction A1 executes successfully
MPMPolicyAction A2 calls MPMECAPolicy B
MPMPolicyAction A3 is either waiting to execute, or is executing

When MPMECAPolicy B is called, it presumably should execute under the scope of control of MPMECAPolicy A (since Policy A has not finished executing). However, calling another MPMECAPolicy means that now, the event clause of Policy B should be activated. It is very difficult to ensure that the next thing the Policy Engine does is determine if the event clause of B is satisfied or not.

Furthermore, what happens to MPMPolicyAction A3? Is MPMECAPolicy B supposed to finish execution before MPMPolicyAction A3? This requires additional logic (priorities do not work here!), which requires communication between the policy engine and both MPMECAPolicy A and MPMECAPolicy B. Even if these problems are solved, what happens if MPMPolicyAction A3 fails, and the mpmPolExecFailStrategy has a value of 2 (i.e. if an action fails, then does a rollback need to be performed)? Does MPMECAPolicy B also get rolled back?

An MPMPolicyAction **shall not** call another MPMPolicy.

An MPMPolicyAction **may** invoke one or more MPMPolicyActions in a different MPMECAPolicy

Figure 5-25 shows the MPMPolicyAction class and its subclasses.

Table 5-107 defines the attributes for this class.

Table 5-107: Attributes of the MPMPolicyAction Class

Attribute Name	Description
mpmPolicyActionData : String[1..*]	This is a mandatory attribute that defines an array of strings. Each string in the array is a 2-tuple, consisting of a single character defining how this attribute is used, and the value of an attribute name of an MPMPolicyAction object. Since this attribute could represent a term in an MPMPolicyClause (e.g. var = MPMPolicyAction.mpmPolicyActionData), a complete MPMPolicyClause (e.g. the mpmPolicyActionData attribute contains text that defines the entire action clause), or the name of a MPMPolicyAction to invoke, each element in the string array is prepended with one of the following strings: o 'v:' (or 'variable:'), to denote a term in an MPMPolicyClause o 'c:' (or 'clause:'), to denote an entire MPMPolicyClause o 'a:' (or 'action:'), to invoke a MPMPolicyAction in a different MPMPolicy For example, if the value of this attribute is: {('t':'set destPort to 80'), ('a':'call PortHandlingAction')} then this attribute contains two actions. The first is the action portion of an MPMPolicyClause, and sets the variable destPort to a value of 80. The second calls the MPMPolicyAction named 'PortHandlingAction'. This attribute works with another class attribute, called mpmPolicyActionEncoding, which defines how to interpret this attribute. These two attributes form a tuple, and together enable a machine to understand the syntax and value of the data carried by the object instance of this class.
mpmPolicyAction-Encoding : MPMPolValueType[1..1]	This is a mandatory non-zero enumerated integer attribute, and defines how to interpret the mpmPolicyActionData class attribute. These two attributes form a tuple, and together enable a machine to understand the syntax and value of the data carried by the object instance of this class. Allowed values are defined in the MPMPolValueType enumeration.

Table 5-108 defines the attributes for this class.

Table 5-108: Operations of the MPMPolicyAction Class

Operation Name	Description
getMPMPolicyActionData() : String[1..*]	This operation returns the current value of the mpmPolicyActionData attribute. If the mpmPolicyActionData attribute does not have a value, then this operation **shall** return an error.
setMPMPolicyActionData(in newValue : String[1..*])	This operation sets the value of the mpmPolicyActionData attribute. The value of the mpmPolicyActionData attribute **shall not** be an empty string.
getMPMPolicyActionEncoding() : MPMPolValueType[1..1]	This operation returns the current value of the mpmPolicyActionEncoding attribute. If this attribute does not have a value, then this operation **shall** return an error.
setMPMPolicyActionEncoding(in newValue : MPMPolValueType[1..1])	This operation sets the value of the mpmPolicyActionEncoding attribute.

5.3.2.6.7.26 MPMPolicyCollection

An MPMPolicyCollection is an optional concrete class that enables a collection (e.g. set, bag, or other, more complex, collections of elements) of arbitrary objects to be defined and used as part of an MPMPolicyClause.

One of the problems with ECA policy rules is when an enumeration occurs in the event and/or condition clauses.

EXAMPLE: If a set of events is received, the policy system may need to wait for patterns of events to emerge (e.g. any number of Events of type A, followed by either one event of type B or two events of type Event C). Similarly, for conditions, testing the value of a set of attributes may need to be performed. Both of these represent behaviour similar to a set of if-then-else statements or a switch statement in imperative programming languages.

It is typically not desirable for the policy system to represent each choice as its own policy clause (i.e. a 3-tuple), as this creates object explosion and poor performance. Furthermore, in these cases, it is often required to have a set of complex logic to be executed, where the logic varies according to the particular event or condition that was selected. It is therefore much too complex to represent this using separate objects, especially when the logic is application- and/or vendor-specific. However, recall that one of the goals of the present document was to facilitate the machine-driven construction of policies. Therefore, a solution to this problem is needed.

Therefore, the present document defines the concept of a collection of entities, called an MPMPolicyCollection. Conceptually, the items to be collected (e.g. events or conditions) are aggregated in one or more MPMPolicyCollection objects of the appropriate type.

Another example is for an MPMPolicyCollection object to aggregate logic blocks (including MPMDeclarativePolicies) to execute.

The computation(s) represented by an MPMPolicyCollection may be part of a larger MPMPolicyClause, since MPMPolicyCollection is a subclass of MPMPolicyComponentDecorator, and can be used to decorate an MPMPolicyClause.

Figure 5-26 shows the attributes and operations of the MPMPolicyCollection class.

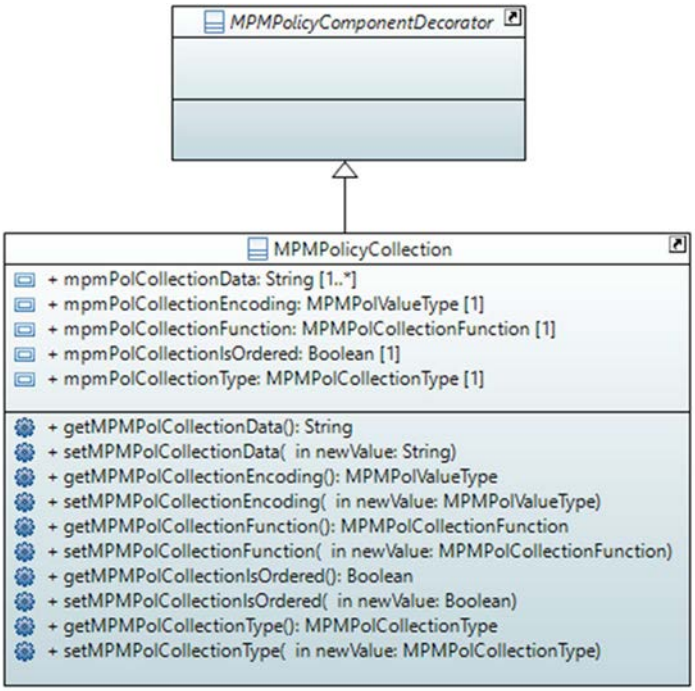

Figure 5-25: MPMPolicyCollection Attributes and Operations

Table 5-109 defines the attributes for this class.

Table 5-109: Attributes of the MPMCollection Class

Attribute Name	Description
mpmPolCollectionData : String[1..*]	This is a mandatory attribute that defines an array of strings. Each string in the array defines a domain-specific identifier of an object that is collected by this object instance. This attribute works with another class attribute, called mpmPolicyCollectionEncoding, which defines how to interpret this attribute. These two attributes form a tuple, and together enable a machine to understand the syntax and value of the data carried by the object instance of this class.
mpmPolCollection-Encoding : MPMPolValueType[1..1]	This is a mandatory non-zero enumerated integer attribute, and defines how to interpret the mpmPolCollectionData class attribute. These two attributes form a tuple, and together enable a machine to understand the syntax and value of the data carried by the object instance of this class. Allowed values are defined in the MPMPolValueType enumeration.

Attribute Name	Description
mpmPolCollection-Function : MPMPolCollection-Function[1..1]	This is a mandatory non-zero enumerated integer attribute, and defines how this collection is used (e.g. is it a collection of objects for an event, or for logic processing, or other functions).
mpmPolCollectionIs-Ordered : Boolean[1..1]	This is a mandatory Boolean attribute. If the value of this attribute is TRUE, then all elements in this instance of this MPMPolicyCollection object are ordered.
mpmPolCollectionType : MPMPolCollection-Type[1..1]	This is a mandatory non-zero enumerated integer attribute, and defines the type of collection that this object instance is.

Table 5-110 defines the operations for this class.

Table 5-110: Operations of the MPMCollection Class

Operation Name	Description
getMPMPolCollectionData() : String[1..*]	This operation returns the current value of the mpmPolCollectionData attribute. If the mpmPolCollectionData attribute does not have a value, then this operation **shall** return an error.
setMPMPolCollectionData(in newValue : String[1..*])	This operation sets the value of the mpmPolCollectionData attribute. The value of the mpmPolCollectionData attribute **shall not** be an empty string.
getMPMPolicyCollection-Encoding() : MPMPolValueType[1..1]	This operation returns the current value of the mpmPolicyCollectionEncoding attribute. If this attribute does not have a value, then this operation **shall** return an error.
setMPMPolicyCollection-Encoding(in newValue : MPMPolValueType[1..1])	This operation sets the value of the mpmPolicyCollectionEncoding attribute.
getMPMPolCollectionFunction() : MPMPolCollectionFunction[1..1]	This operation returns the current value of the mpmPolicyCollectionFunction attribute. If this attribute does not have a value, then this operation **shall** return an error.
setMPMPolCollectionFunction(in newValue : MPMPolCollection-Function [1..1])	This operation sets the value of the mpmPolicyCollectionFunction attribute.
getMPMPolCollectionIs-Ordered() : Boolean[1..1]	This operation returns the current value of the mpmPolicyCollectionIsOrdered attribute. If this attribute does not have a value, then this operation **shall** return an error.
setMPMPolCollectionIsOrdered (in newValue : Boolean[1..1])	This operation sets the value of the mpmPolicyCollectionIsOrdered attribute.
getMPMPolCollectionType() : MPMPolCollectionType [1..1]	This operation returns the current value of the mpmPolicyCollectionType attribute. If this attribute does not have a value, then this operation **shall** return an error.
setMPMPolCollectionType(in newValue : MPMPolCollectionType[1..1])	This operation sets the value of the mpmPolicyCollectionType attribute.

5.3.2.6.7.27 MPMPolicySource

This is an optional class that defines a set of MCMManagedEntity objects that authored, or are otherwise responsible for, this MPMPolicy. An MPMPolicySource does NOT evaluate or execute MPMPolicies. Its primary use is for auditability and the implementation of deontic and/or alethic logic.

An MPMPolicySource object **should** be mapped to a role or set of roles (e.g. using the role-object pattern). This enables role-based or policy-based access control to be used to restrict which MCMManagedEntity objects can author a given policy.

An MPMPolicySource object **should** be mapped to a subclass of MCMPolicyRole.

Figure 5-27 shows the MPMPolicySource and MPMPolicyTarget classes.

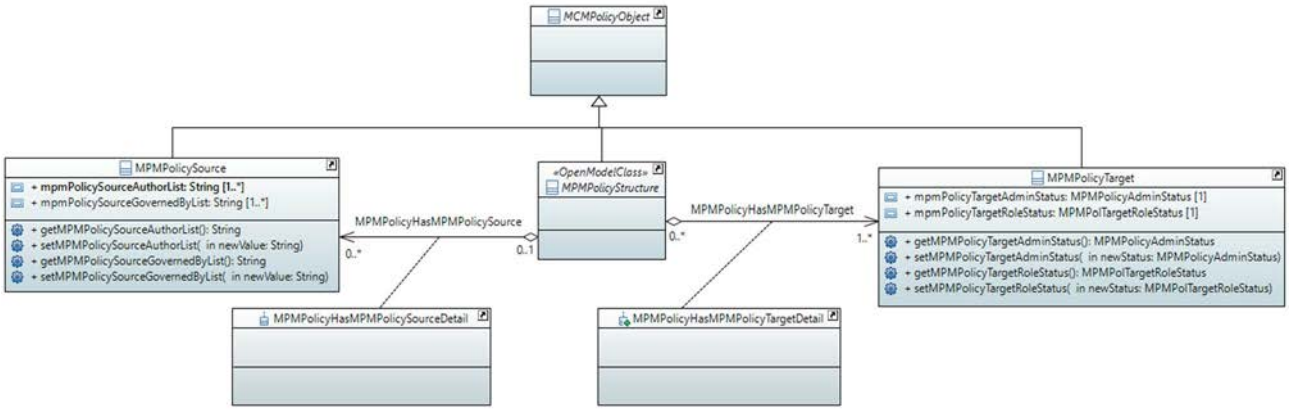

Figure 5-26: MPMPolicySource and MPMPolicyTarget Attributes and Operations

The purpose of the MPMPolicySource object is to provide a mechanism for business logic to be inserted into the model to manipulate the objects that serve as the authors or are responsible for this MPMPolicy.

Table 5-111 defines the attributes for this class.

Table 5-111: Attributes of the MPMPolicySource Class

Attribute Name	Description
mpmPolicySourceAuthor : String[1..*]	This is an optional attribute that defines an array of strings. Each string in the array is a 2-tuple, consisting of a single character defining the type of object that contains the Author, and the name of an MCMParty or MCMPartyRole object class. The mapping of the first character is defined as follow: 'o': MCMOrganization 'p': MCMPerson 'r': MCMPartyRole For example, if this attribute contains the following values: {('o':'CustomerSupport'), ('r':'CSRole')} then the first 2-tuple identifies an MCMOrganization named 'CustomerSupport', and the second identifies an MCMPartyRole named 'CSRole'. This attribute **shall** only use the characters '0', 'p', and 'r' to define the type of object. This attribute **shall not** contain a prefix (i.e. a character before the quote) of more than 1 character. This attribute **shall not** contain a NULL or empty string.
mpmPolicySourceGovern edBy : String[1..*]	This is an optional attribute that defines an array of strings. Each string in the array is a 2-tuple, consisting of a single character defining the type of object that governs this MPMPolicy, and the name of an MCMParty or MCMPartyRole object class. The mapping of the first character is defined as follow: 'o': MCMOrganization 'p': MCMPerson 'r': MCMPartyRole For example, if this attribute contains the following values: {('o':'CustomerSupport'), ('r':'CSRole')} then the first 2-tuple identifies an MCMOrganization named 'CustomerSupport', and the second identifies an MCMPartyRole named 'CSRole'. This attribute **shall** only use the characters '0', 'p', and 'r' to define the type of object. This attribute **shall not** contain a prefix (i.e. a character before the quote) of more than 1 character. This attribute **shall not** contain a NULL or empty string.

Table 5-112 defines the operations for this class.

Table 5-112: Operations of the MPMPolicySource Class

Operation Name	Description
getMPMPolicySourceAuthor() : String[1..*]	This operation returns the current value of the mpmPolicySourceAuthor attribute. This attribute **shall** be a two-tuple. This attribute **shall** only use the characters 'o', 'p', and 'r' to define the type of object. If the mpmPolicySourceAuthor attribute does not have a value, then this operation **shall** return an error.
setMPMPolicySourceAuthor(in newValue : String[1..*])	This operation sets the value of the mpmPolicySourceAuthor attribute. This attribute **shall** be a two-tuple. This attribute **shall** only use the characters 'o', 'p', and 'r' to define the type of object. The value of the mpmPolicySourceAuthor attribute **shall not** be a NULL or empty string.
getMPMPolicySourceGovernedBy() : String[1..*]	This operation returns the current value of the mpmPolicySourceGovernedBy attribute. This attribute **shall** be a two-tuple. If this attribute does not have a value, then this operation **shall** return an error. If the mpmPolicySourceAuthor attribute does not have a value, then this operation **shall** return an error.
setMPMPolicySourceGovernedBy(in newValue : String[1..*])	This operation sets the value of the mpmPolicySourceGovernedBy attribute. This attribute **shall** be a two-tuple. This attribute **shall** only use the characters 'o', 'p', and 'r' to define the type of object. The value of the mpmPolicySourceGovernedBy attribute **shall not** be an empty string.

5.3.2.6.7.28 MPMPolicyTarget

This is a mandatory class that defines a set of MCMManagedEntity objects that an MPMPolicy is applied to.

An MCMManagedEntity object **shall** satisfy two conditions in order to be defined as an MPMPolicyTarget. First, the set of MCMManagedEntities that are to be affected by the MPMPolicy **shall** all agree to play the role of an MPMPolicyTarget. In general, an MCMManagedEntity may or may not be in a state that enables MPMPolicy objects to be applied to it to change its state; hence, a negotiation process may need to occur to enable the MPMPolicyTarget to signal when it is willing to have MPMPolicy objects applied to it. Second, an MPMPolicyTarget **shall** be able to process (directly or with the aid of a proxy) the action(s) of a set of MPMPolicy objects on each of the MCMPolicyTarget objects.

If a proposed MPMPolicyTarget object is in a state that enables changes to be made to it, and if it can process those changes, it **shall** have its mpmPolicyTargetEnabled Boolean class attribute set to a value of TRUE.

An MPMPolicyTarget object **should** be mapped to a role or set of roles using the role-object pattern. This enables role-based or policy-based access control to be used to restrict which MCMManagedEntity objects can be affected by a given MPMPolicy.

An MPMPolicyTarget object **should** be mapped to a subclass of MCMPolicyRole.

Figure 5-26 shows the MPMPolicySource and MPMPolicyTarget classes.

Table 5-113 defines the attributes for this class.

Table 5-113: Attributes of the MPMPolicyTarget Class

Attribute Name	Description
mpmPolicyTargetAdminStatus : MPMPolicyAdminStatus[1..1]	This is a mandatory enumerated non-negative integer attribute that defines the current administrative status of this particular MPMPolicyTarget object. The allowable values of this enumeration are defined by the MPMPolicyAdminStatus enumeration.
mpmPolicyTargetRoleStatus : MPMPolTargetRoleStatus[1..1]	This is a mandatory enumerated non-negative integer attribute that defines the current readiness of this particular MPMPolicyTarget object to take on the PolicyTargetRole.

Table 5-114 defines the operations for this class.

Table 5-114: Operations of the MPMPolicyTarget Class

Operation Name	Description
getMPMPolicyTargetAdminStatus() : MPMPolicyAdminStatus[1..1]	This operation returns the current administrative status of this particular MPMPolicyTarget object, which is defined by the MPMPolicyAdminStatus enumeration. If the mpmPolicyTargetAdminStatus attribute does not have a value, then this operation **shall** return an error.
setMPMPolicyTargetAdminStatus(in newStatus : MPMPolicyAdminStatus[1..1])	This operation sets the value of the current administrative status of this particular MPMPolicyTarget object.
getMPMPolicyTargetRoleStatus() : MPMPolicyRoleStatus[1..1]	This operation returns the current readiness of this particular MPMPolicyTarget object to play the role of a PolicyTarget. If the mpmPolicyTargetRoleStatus attribute does not have a value, then this operation **shall** return an error.
setMPMPolicyTargetRoleStatus(in newStatus : MPMPolicyRoleStatus[1..1])	This operation sets the value of the mpmPolicyTargetRoleStatus attribute of this particular MPMPolicyTarget object.

5.3.2.7 ENI Extensions to the MPM

5.3.2.7.1 Introduction

The MPM provides an extensible framework that is organized into two main class hierarchies (MPMPolicyStructure and MPMPolicyComponentStructure) with two supporting classes (MPMPolicySubject and MPMPolicyTarget). This clause defines extensions to the Entity class hierarchy in order to represent key concepts that are needed for an ENI System to achieve its functional requirements as described in clause 5 of [3].

Figure 5-27 shows an overview of the ENI Extensions to the MPM. Two classes, ENIIntentPolicyStatement and ENIIntentPolicyClause, are added, along with subclasses of each (not shown in this figure).

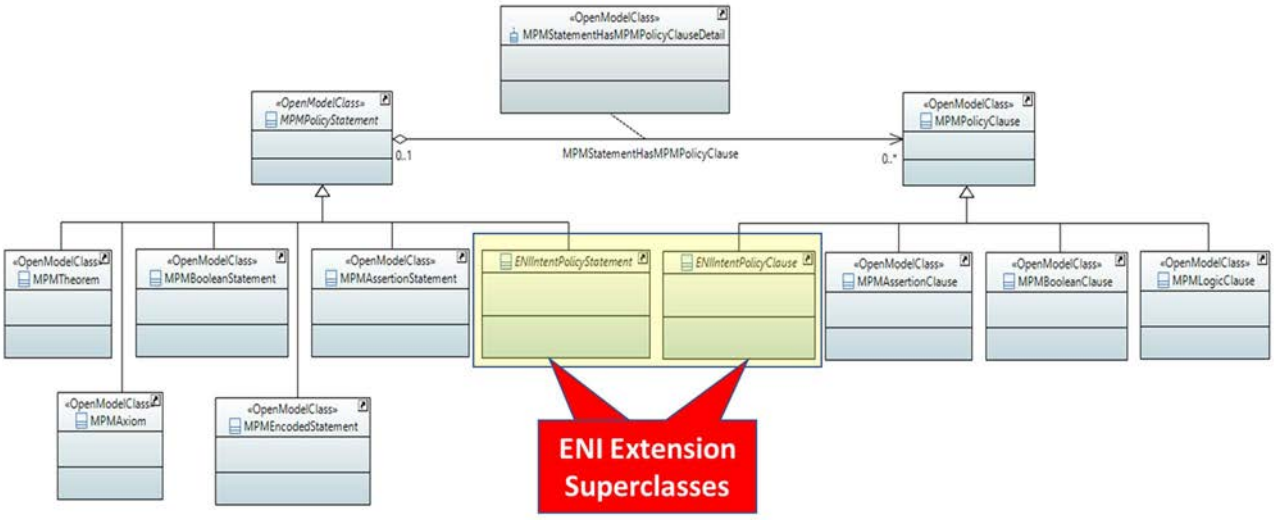

Figure 5-27: Overview of ENI Extensions to the MPM

5.3.2.7.2 Naming Rules

The naming rules of ENI extensions to the MPM model are the same as the naming rules of the ENI Extensions to the MCM (see clause 5.2.3.2).

5.3.2.7.3 ENI Policy Statement Extensions

5.3.2.7.3.1 Introduction

Figure 5-28 defines the current set of ETSI ISG ENI extensions to the MPMPolicyStatement class. This enables new types of ENIPolicyStatement classes to be used with the MPMIntentPolicy, MPMPolicyStatement, and MPMPolicyClause subclasses to define more specific Intent Policies for use by an ENI System.

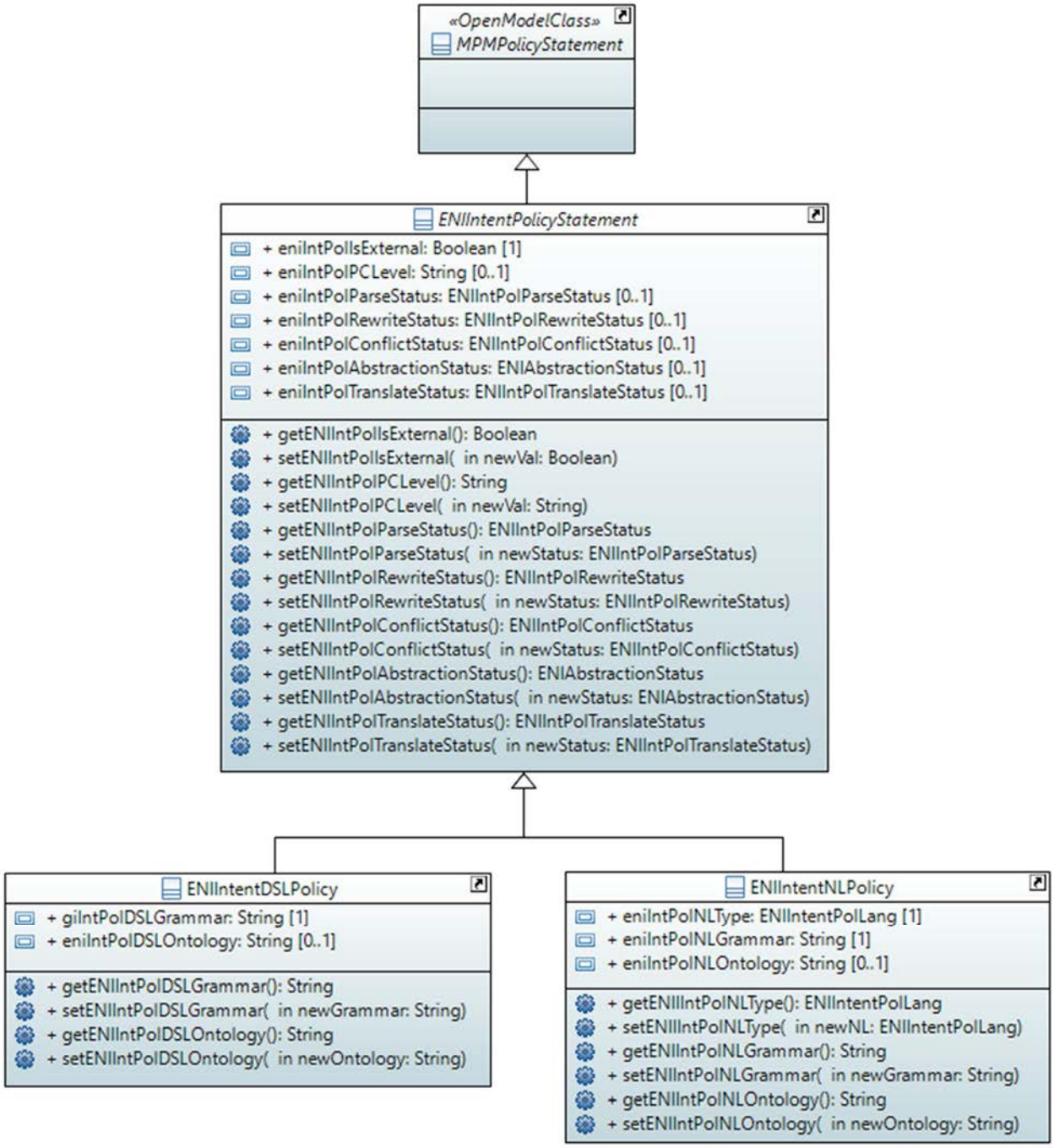

Figure 5-28: ENI Extensions to the MPMPolicyStatement Class

5.2.3.7.3.2 ENIIntentPolicyStatement Class Definition

This is an optional concrete class that defines different types of ENI Intent Policies.

Table 5-115 defines the attributes for this class.

Table 5-115: Attributes of the ENIIntentPolicyStatement Class

Attribute Name	Description
eniIntPolIsExternal: Boolean[1..1]	This is a mandatory Boolean attribute, and whether this ENIIntentPolicyStatement is for external or internal use.
eniIntPolPCLevel: String[0..1]	This is an optional String attribute, and defines the Policy Continuum level of this ENIIntentPolicyStatement.
eniIntPolParseStatus: ENIIntPolParseStatus[0..1]	This is an optional attribute that defines the parsing status of this ENIIntentPolicyStatement. Values are defined in the ENIIntPolParseStatus enumeration.
eniIntPolRewriteStatus: ENIIntPolRewriteStatus[0..1]	This is an optional attribute that defines the rewriting status of this ENIIntentPolicyStatement. Rewriting is needed if a change in abstraction is required. Values are defined in the ENIIntPolRewriteStatus enumeration.
eniIntPolConflictStatus: ENIIntPolConflictStatus[0..1]	This is an optional attribute that defines the conflict status of this ENIIntentPolicyStatement. This is done first, for conflict analysis with other ENIIntentPolicies and second, for other ENIPolicies. Values are defined in the ENIIntPolConflictStatus enumeration.
eniIntPolAbstractionStatus: ENIIntPolAbstractionStatus[0..1]	This is an optional attribute that defines the abstraction status of this ENIIntentPolicyStatement. Changing abstraction (e.g. to a different Policy Continuum level) in general requires a repeat of the above steps. Values are defined in the ENIIntPolAbstractionStatus enumeration.
eniIntPolTranslateStatus: ENIIntPolTranslateStatus[0..1]	This is an optional attribute that defines the translation status of this ENIIntentPolicyStatement. Values are defined in the ENIIntPolTranslateStatus enumeration.

Table 5-116 defines the operations for this class.

Table 5-116: Operations of the ENIIntentPolicyStatement Class

Operation Name	Description
getENIIntPolIsExternal() : Boolean[1..1]	This operation returns the current value of the eniIntPolIsExternal attribute. This operation takes no input parameters. If the mpmIntentTranslationStatus attribute does not have a value, then this operation **shall** return an error.
setENIIntPolIsExternal(in newVal : Boolean[1..1])	This operation sets the value of the eniIntPolIsExternal attribute. This operation takes a single input parameter, called newVal, which defines the new value for the eniIntPolIsExternal attribute.
getENIIntPolPCLevel() : String[1..1]	This operation returns the current value of the eniIntPolPCLevel attribute. This operation takes no input parameters. If the eniIntPolPCLevel attribute does not have a value, then this operation **may** return an error.
setENIIntPolPCLevel(in newVal : String[1..1])	This operation sets the value of the eniIntPolPCLevel attribute. This operation takes a single input parameter, called newVal, which defines the new value for the eniIntPolPCLevel attribute.
getENIIntPolParseStatus() : ENIIntPolParseStatus[1..1]	This operation returns the current value of the eniIntPolParseStatus attribute. This operation takes no input parameters. If the eniIntPolParseStatus attribute does not have a value, then this operation **shall** return an error.
setENIIntPolParseStatus(in newStatus : ENIIntPolParseStatus[1..1])	This operation sets the value of the eniIntPolParseStatus attribute. This operation takes a single input parameter, called newStatus, which defines the new value for the eniIntPolParseStatus attribute.
getENIIntPolRewriteStatus() : ENIIntPolRewriteStatus[1..1]	This operation returns the current value of the eniIntPolRewriteStatus attribute. This operation takes no input parameters. If the eniIntPolRewriteStatus attribute does not have a value, then this operation **shall** return an error.
setENIIntPolRewriteStatus(in newStatus : ENIIntPolRewriteStatus[1..1])	This operation sets the value of the eniIntPolRewriteStatus attribute. This operation takes a single input parameter, called newStatus, which defines the new value for the eniIntPolRewriteStatus attribute.
getENIIntPolConflictStatus() : ENIIntPolConflictStatus[1..1]	This operation returns the current value of the eniIntPolConflictStatus attribute. This operation takes no input parameters. If the eniIntPolConflictStatus attribute does not have a value, then this operation **shall** return an error.
setENIIntPolConflictStatus(in newStatus : ENIIntPolConflictStatus[1..1])	This operation sets the value of the eniIntPolConflictStatus attribute. This operation takes a single input parameter, called newStatus, which defines the new value for the eniIntPolConflictStatus attribute.

Operation Name	Description
getENIIntPolAbstractionStatus() : ENIIntPolAbstractionStatus[1..1]	This operation returns the current value of the eniIntPolAbstractionStatus attribute. This operation takes no input parameters. If the eniIntPolAbstractionStatus attribute does not have a value, then this operation **shall** return an error.
setENIIntPolAbstractionStatus(in newStatus : ENIIntPolAbstractionStatus[1..1])	This operation sets the value of the eniIntPolAbstractionStatus attribute. This operation takes a single input parameter, called newStatus, which defines the new value for the eniIntPolAbstractionStatus attribute.
getENIIntPolTranslateStatus() : ENIIntPolTranslateStatus[1..1]	This operation returns the current value of the eniIntPolTranslate attribute. This operation takes no input parameters. If the eniIntPolTranslate attribute does not have a value, then this operation **shall** return an error.
setENIIntPolTranslateStatus(in newStatus : ENIIntPolTranslateStatus[1..1])	This operation sets the value of the eniIntPolTranslate attribute. This operation takes a single input parameter, called newStatus, which defines the new value for the eniIntPolTranslate attribute.

This class does not define any relationships.

5.3.2.7.3.3 ENIIntentDSLPolicy Class Definition

This is an optional concrete class that defines an ENI Intent Policy that is implemented using a DSL.

Table 5-117 defines the attributes for this class.

Table 5-117: Attributes of the ENIIntentDSLPolicyStatement Class

Attribute Name	Description
eniIntPolDSLGrammar: String[1..1]	This is a mandatory String attribute, and defines the grammar used by this DSL.
eniIntPolDSLOntology: String[0..1]	This is an optional String attribute, and defines the ontology used to add meaning to the DSL's grammar.

Table 5-118 defines the operations for this class.

Table 5-118: Operations of the ENIDSLIntentPolicyStatement Class

Operation Name	Description
getENIIntPolDSLGrammar() : String[1..1]	This operation returns the current value of the eniIntPolDSLGrammar attribute. This operation takes no input parameters. If the eniIntPolDSLGrammar attribute does not have a value, then this operation **shall** return an error.
setENIIntPolDSLGrammar(in newGrammar : String[1..1])	This operation sets the value of the eniIntPolDSLGrammar attribute. This operation takes a single input parameter, called newGrammar, which defines the new value of the eniIntPolDSLGrammar attribute.
getENIIntPolOntology() : String[1..1]	This operation returns the current value of the eniIntPolDSLOntology attribute. This operation takes no input parameters. If the eniIntPolDSLOntology attribute does not have a value, then this operation **shall** return an error.
setENIIntPolOntology(in newOntology : String[1..1])	This operation sets the value of the eniIntPolDSLOntology attribute. This operation takes a single input parameter, called newOntology, which defines the new value for the eniIntPolDSLOntology attribute.

This class does not define any relationships.

5.3.2.7.3.4 ENIIntentNLPolicy Class Definition

This is an optional concrete class that defines an ENI Intent Policy that uses a restricted natural language.

Table 5-119 defines the attributes for this class.

Table 5-119: Attributes of the ENIIntentPolicyStatement Class

Attribute Name	Description
eniIntPolNLType : ENIIntentPolLang[1..1]	This is a mandatory attribute that defines an approved Restricted Natural Language to use for this ENIIntentNLPolicy. A list of Restricted Natural Languages is defined in the ENIIntentPolLang enumeration.
eniIntPolNLGrammar: String[1..1]	This is a mandatory String attribute, and defines the grammar used by this restricted natural language.
eniIntPolDSLOntology: String[0..1]	This is an optional String attribute, and defines the ontology used to add meaning to the grammar of this restricted natural language.

Table 5-120 defines the operations for this class.

Table 5-120: Operations of the ENIIntentPolicyStatement Class

Operation Name	Description
getENIIntPolNLType() : ENIIntentPolLang[1..1]	This operation returns the current value of the eniIntPolNLType attribute. This operation takes no input parameters. If the eniIntPolNLType attribute does not have a value, then this operation **shall** return an error.
setENIIntPolNLType(in newNL : ENIIntentPolLang[1..1])	This operation sets the value of the eniIntPolNLType attribute. This operation takes a single input parameter, called newNL, which defines the new value of the eniIntPolNLType attribute.
getENIIntPolNLGrammar() : String[1..1]	This operation returns the current value of the eniIntPolNLGrammar attribute. This operation takes no input parameters. If the eniIntPolDSLGrammar attribute does not have a value, then this operation **shall** return an error.
setENIIntPolNLGrammar(in newGrammar : String[1..1])	This operation sets the value of the eniIntPolNLGrammar attribute. This operation takes a single input parameter, called newGrammar, which defines the new value of the eniIntPolNLGrammar attribute.
getENIIntPolOntology() : String[1..1]	This operation returns the current value of the eniIntPolNLOntology attribute. This operation takes no input parameters. If the eniIntPolNLOntology attribute does not have a value, then this operation **shall** return an error.
setENIIntPolOntology(in newOntology : String[1..1])	This operation sets the value of the eniIntPolNLOntology attribute. This operation takes a single input parameter, called newOntology, which defines the new value for the eniIntPolNLOntology attribute.

This class does not define any relationships.

5.3.2.7.4 ENI Policy Clause Extensions

5.3.2.7.4.1 Introduction

Figure 5-29 defines the current set of ETSI ISG ENI extensions to the MPMPolicyStatement class. This enables new types of ENIPolicyClause classes to be used with the MPMIntentPolicy, MPMPolicyStatement, and MPMPolicyClause subclasses to define more specific types of Intent Policies for use by an ENI System.

Figure 5-29: ENI Extensions to the MPMPolicyClause Class

5.3.2.7.4.2 ENIIntentPolicyClause Class Definition

This is an optional concrete class that defines different types of ENI Intent Policies.

Table 5-121 defines the attributes for this class.

Table 5-121: Attributes of the ENIIntentPolicyClause Class

Attribute Name	Description
eniIntClauseParseStatus: ENIIntPolParseStatus[0..1]	This is an optional attribute that defines the parsing status of this ENIIntentPolicyClause. Values are defined in the ENIIntPolParseStatus enumeration.
eniIntClauseRewriteStatus: ENIIntPolRewriteStatus[0..1]	This is an optional attribute that defines the rewriting status of this ENIIntentPolicyClause. Rewriting is needed if a change in abstraction is required. Values are defined in the ENIIntPolRewriteStatus enumeration.
eniIntClauseConflictStatus: ENIIntPolConflictStatus[0..1]	This is an optional attribute that defines the conflict status of this ENIIntentPolicyClause. This is done first, for conflict analysis with other ENIIntentPolicies and second, for other ENIPolicies. Values are defined in the ENIIntPolConflictStatus enumeration.
eniIntClauseAbstractionStatus: ENIIntPolAbstractionStatus[0..1]	This is an optional attribute that defines the abstraction status of this ENIIntentPolicyClause. Changing abstraction (e.g. to a different Policy Continuum level) in general requires a repeat of the above steps. Values are defined in the ENIIntPolAbstractionStatus enumeration.
eniIntClauseTranslateStatus: ENIIntPolTranslateStatus[0..1]	This is an optional attribute that defines the translation status of this ENIIntentPolicyClause. Values are defined in the ENIIntPolTranslateStatus enumeration.

Table 5-122 defines the operations for this class.

Table 5-122: Attributes of the ENIIntentPolicyClause Class

Operation Name	Description
getENIIntClauseParseStatus() : ENIIntPolParseStatus[1..1]	This operation returns the current value of the eniIntClauseParseStatus attribute. This operation takes no input parameters. If the eniIntClauseParseStatus attribute does not have a value, then this operation **shall** return an error.
setENIIntClauseParseStatus(in newStatus : ENIIntPolParseStatus[1..1])	This operation sets the value of the eniIntClauseParseStatus attribute. This operation takes a single input parameter, called newStatus, which defines the new value for the eniIntClauseParseStatus attribute.
getENIIntClauseRewriteStatus() : ENIIntPolRewriteStatus[1..1]	This operation returns the current value of the eniInt eniIntClauseParseStatus RewriteStatus attribute. This operation takes no input parameters. If the eniIntClauseParseStatus attribute does not have a value, then this operation **shall** return an error.
setENIIntClauseRewriteStatus(in newStatus : ENIIntPolRewriteStatus[1..1])	This operation sets the value of the eniIntClauseRewriteStatus attribute. This operation takes a single input parameter, called newStatus, which defines the new value for the eniIntClauseRewriteStatus attribute.
getENIIntClauseConflictStatus() : ENIIntPolConflictStatus[1..1]	This operation returns the current value of the eniIntClauseConflictStatus attribute. This operation takes no input parameters. If the eniIntClauseConflictStatus attribute does not have a value, then this operation **shall** return an error.
setENIIntClauseConflictStatus(in newStatus : ENIIntPolConflictStatus[1..1])	This operation sets the value of the eniIntClauseConflictStatus attribute. This operation takes a single input parameter, called newStatus, which defines the new value for the eniIntClauseConflictStatus attribute.
getENIIntClauseAbstractionStatus() : ENIIntPolAbstractionStatus[1..1]	This operation returns the current value of the eniIntClauseAbstractionStatus attribute. This operation takes no input parameters. If the eniIntClauseAbstractionStatus attribute does not have a value, then this operation **shall** return an error.
setENIIntClauseAbstractionStatus(in newStatus : ENIIntPolAbstractionStatus[1..1])	This operation sets the value of the eniIntClauseAbstractionStatus attribute. This operation takes a single input parameter, called newStatus, which defines the new value for the eniIntClauseAbstractionStatus attribute.
getENIIntClauseTranslateStatus() : ENIIntPolTranslateStatus[1..1]	This operation returns the current value of the eniIntClauseTranslate attribute. This operation takes no input parameters. If the eniIntClauseTranslate attribute does not have a value, then this operation **shall** return an error.
setENIIntClauseTranslateStatus(in newStatus : ENIIntPolTranslateStatus[1..1])	This operation sets the value of the eniIntClauseTranslate attribute. This operation takes a single input parameter, called newStatus, which defines the new value for the eniIntClauseTranslate attribute.

This class does not define any relationships.

5.3.2.7.4.3 ENIIntPolClauseGoal

This is an optional concrete class. It is used to define goals of an ENIIntentPolicy.

This is for further study in Release 4 of the present document.

5.3.2.7.4.4 ENIIntPolClauseSubject

This is an optional concrete class. It is used to define the subject of an ENIIntentPolicy.

In one abstraction of an ENIIntentPolicy, a subject (e.g. ENIIntPolClauseSubject) is used to perform a set of actions (e.g. ENIIntPolClauseAction) on a target (e.g. ENIIntPolClauseTarget).

This is for further study in Release 4 of the present document.

5.3.2.7.4.5 ENIIntPolClauseTarget

This is an optional concrete class. It is used to define the target of an ENIIntentPolicy.

In one abstraction of an ENIIntentPolicy, a target (e.g. ENIIntPolClauseTarget) is the object that a subject (e.g. ENIIntPolClauseSubject) performs a set of actions (e.g. ENIIntPolClauseAction) on.

This is for further study in Release 4 of the present document.

5.3.2.7.4.6 ENIIntPolClauseCondition

This is an optional concrete class. It is used to define goals of an ENIIntentPolicy.

In one abstraction of an ENIIntentPolicy, a condition (e.g. ENIIntPolClauseCondition) is an MPMPolicyClause that determines whether or not the set of actions (e.g. ENIIntPolClauseAction) in this ENIIntentPolicyClause should be executed or not.

This is for further study in Release 4 of the present document.

5.3.2.7.4.7 ENIIntPolClauseAction

This is an optional concrete class. It is used to define the actions of an ENIIntentPolicy.

In one abstraction of an ENIIntentPolicy, an action (e.g. ENIIntPolClauseAction) is an MPMPolicyClause that performs an operation that changes the state of an MCMMangedEntity.

This is for further study in Release 4 of the present document.

5.3.2.7.4.8 ENIIntPolClauseScope

This is an optional concrete class. It is used to define the scope of an ENIIntentPolicy.

In one abstraction of an ENIIntentPolicy, the scope determines the time, context, and applicable MCMManagedEntities and MCMRoles that are affected by this ENIIntentPolicy [i.3] and [i.4].

This is for further study in Release 4 of the present document.

5.3.2.8 ENI Extended Policy Model

The ENI Extended Policy Model **shall** be based on the ENI Extended Core Model (see clause 5.2.4 of the present document). In addition, the ENI Extended Policy Model **shall** be based on the ENI Extended Core Model (see clause 5.2.4 of the present document). The ENI Extended Policy Model **shall** be based on the MPM classes, attributes, operations, and relationships defined in the present document. Finally, the present document defines two class hierarchies (i.e. ENIIntentPolicyStatement and ENIIntentPolicyClause and their respective subclasses) that extend the MPM. These two class hierarchies **should** be used to define the ENI Extended Policy Model.

Additional classes defined by ETSI ISG ENI are for further study in Release 4 of the present document.

6 ENI Data Models

6.1 Introduction

The ENI Extended Core and Policy Information Models have been used to create a first release of the ENI APIs (see clause 7 of [3]. This has created a technology-neutral data model for those specific functions. The rest of this clause is for further study in Release 4 of the present document.

6.2 ENI Technology-Neutral Data Model

Detailed technology-neutral data model requirements are for further study in Release 4 of the present document.

6.3 ENI Technology-Specific Data Models

Detailed technology-specific data model requirements are for further study in Release 4 of the present document.

7 Requirements

7.1 Information Model Requirements

The following clauses define requirements for the ENI information model.

[IM01] The ENI Information Model shall use single inheritance.

[IM02] The ENI Information Model shall not use multiple inheritance.

[IM03] The ENI Information Model shall not use overriding.

[IM04] The ENI Information Model should use the Single Responsibility Principle to design class hierarchies [i.5].

[IM05] The ENI Information Model should use the Liskov Substitution Principle to design class hierarchies [i.5].

[IM06] The ENI Information Model should use Design by Contract to manage behaviour [i.5].

[IM07] The ENI Information Model should use the Dependency Inversion Principle [i.5] to ensure that the model is modular and extensible.

[IM08] The ENI Information Model should strive to realize loose coupling [i.5] whenever possible to ensure that the model is modular and extensible.

[IM09] The ENI Information Model should strive to realize high cohesion [i.5] whenever possible to ensure that the model is modular and extensible.

[IM10] The ENI Information Model should use the Role-Object Pattern [i.8] to abstract individual entities into roles as applicable.

7.2 Data Model Requirements

Detailed data model requirements are for further study in Release 4 of the present document.

7.3 Ontology Requirements

Detailed ontology requirements are for further study in Release 4 of the present document.

8 Future Work

8.1 Open Issues for the Present Document

There are no current open issues for the present document.

8.2 Issues for Future Study

The following are issues for further study in Release 4 of the present document.

From clause 4.7

This is for further study in Release 4 of the present document.

From clause 4.8

This is for further study in Release 4 of the present document.

From clause 4.9

This is for further study in Release 4 of the present document.

From clause 4.10

This is for further study in Release 4 of the present document.

From clause 4.11

A preliminary example of how models were used to build ENI APIs is described in clause 8 of [3]. This will be replaced by a formal and more complete definition of how this was done in Release 4 of the present document.

From clause 5.2.2.4.3.1

NOTE 1: The use of subclasses of MCMUnManagedEntity in the ENI Extended Core Model are for further study.

From clause 5.2.2.4.5.6.3

NOTE 2: Should this version of the ENI Extended Core Model use the three subclasses of MCMFeature?

From clause 5.2.2.4.7

NOTE 3: Should this version of the ENI Extended Core Model use Product classes?

From clause 5.2.2.4.7.5

NOTE 4: These [Product relationships] will be defined if it is decided to use MCMProduct and its subclasses.

From clause 5.2.2.4.8.5

NOTE 5: These will be defined if it is decided to use MCMService subclasses.

From clause 5.2.2.4.9.5

NOTE 6: This [MCMResource Relationship] will be defined if it is decided to use MCMOffer subclasses.

From clause 5.2.2.4.9.6.3

NOTE 7: The use of other MCMVirtualResource subclasses besides those defined in the present document is for further study.

From clause 5.2.2.4.9.6.6

NOTE 8: The use of other MCMLogicalResource subclasses besides those defined in the present document is for further study.

From clause 5.2.2.4.11

NOTE 9: Should this version of the ENI Extended Core Model use MCMParty Objects?

From clause 5.2.2.4.12

NOTE 10: Should this version of the ENI Extended Core Model use MCMBusiness Objects?

From clause 5.2.3

NOTE 11: Additional clauses for describing more required features will be added in the next version of the present document.

From clause 5.2.3.3.6.1

NOTE 12: This figure [ENIEvent class hierarchy] may be augmented with additional subclasses in the future.

From clause 5.2.3.4.6.1

NOTE 13: This figure [ENIBehavior class hierarchy] may be augmented with additional subclasses in the future.

From clause 5.2.3.5.10.2

This class [ENIDigitalIdentity], and its subclasses, are for further study in Release 4 of the present document.

From clause 5.2.3.5.10.3

This class [ENIContextualIdentity], and its subclasses, are for further study in Release 4 of the present document.

From clause 5.2.4

Additional classes defined by ETSI ISG ENI are for further study in Release 4 of the present document.

From clause 5.3.2.7.4.3

This [ENIIntPolGoal] is further study in Release 4 of the present document.

From clause 5.3.2.7.4.4

This [ENIIntPolClauseSubject] is for further study in Release 4 of the present document.

From clause 5.3.2.7.4.5

This [ENIIntPolClauseTarget] classes defined by ETSI ISG ENI are for further study in Release 4 of the present document.

From clause 5.3.2.7.4.6

This [ENIIntPolClauseCondition] is for further study in Release 4 of the present document.

From clause 5.3.2.7.4.7

This [ENIIntPolClauseAction] for further study in Release 4 of the present document.

From clause 5.3.2.7.4.8

This [ENIIntPolScope] is for further study in Release 4 of the present document.

From clause 5.3.2.8

Additional classes defined by ETSI ISG ENI are for further study in Release 4 of the present document.

From clause 6.1

The rest of this clause is for further study in Release 4 of the present document.

From clause 7.2

The rest of this clause is for further study in Release 4 of the present document.

From clause 7.3

The rest of this clause is for further study in Release 4 of the present document.

History

Document history		
V3.1.1	June 2023	Publication