

ISO/IEC JTC 1/SC 32 N 0688

Date: 2001-10-01

REPLACES: --

<p style="text-align: center;">ISO/IEC JTC 1/SC 32</p> <p style="text-align: center;">Data Management and Interchange</p> <p style="text-align: center;">Secretariat: United States of America (ANSI)</p> <p style="text-align: center;">Administered by Pacific Northwest National Laboratory on behalf of ANSI</p>
--

DOCUMENT TYPE	Working Draft Text (for information or comment)
TITLE	Working Draft - Framework for Registering Business Objects
SOURCE	National Body - Japan
PROJECT NUMBER	
STATUS	For the discussion at JTC1/SC32/WG2 Victoria, Canada
REFERENCES	
ACTION ID.	COM
REQUESTED ACTION	
DUE DATE	
Number of Pages	38
LANGUAGE USED	English
DISTRIBUTION	P & L Members SC Chair WG Conveners and Secretaries

Douglas Mann, Secretariat, ISO/IEC JTC 1/SC 32

Pacific Northwest National Laboratory *, 901 D Street, SW., Suite 900, Washington, DC, 20024-2115,
United States of America

Telephone: +1 703 379 6915 x 111; Facsimile: +1 703 379 8934; E-mail: MannD@battelle.org

available from the JTC 1/SC 32 WebSite <http://www.jtc1sc32.org/>

*Pacific Northwest National Laboratory (PNL) administers the ISO/IEC JTC 1/SC 32 Secretariat on behalf of ANSI

Framework for Registering Business Objects

Working Draft

NWI Proposal for ISO/IEC/JTC1 SC32/WG2

24th September 2001

Source: Japan National Body

Status: DRAFT

Action: For the discussion at JTC1/SC32/WG2 Victoria, Canada

Contents

1. OBJECTIVES AND SCOPE	4
2. RATIONALE	5
3. BASIC CONCEPTS.....	8
3.1 OVERVIEW OF MOF	8
3.2 OVERVIEW OF 11179	10
3.3 CLASSIFICATION FRAMEWORK FOR MODELING BO	10
4. OVERVIEW ON HOW THIS DRAFT STANDARD TO BE USED	11
5. MOF+ METAMETAMODEL (NORMATIVE PART OF THIS DRAFT STANDARD)	13
5.1 MOF+ MODEL CLASSES	15
5.1.1 <i>MetaModelElement</i>	15
5.1.2 <i>ViewPoint</i>	17
5.1.3 <i>Classification</i>	17
5.1.4 <i>ModelClassifier</i>	18
5.1.5 <i>TypedModelElement</i>	18
5.1.6 <i>ModelInstance</i>	19
5.1.7 <i>ModelAssociationEnd</i>	19
5.1.8 <i>ModelReference</i>	20
5.1.9 <i>ModelAssociation</i>	21
5.1.10 <i>Pattern</i>	22
5.1.11 <i>Collaboration</i>	23
5.1.12 <i>Component</i>	23
5.1.13 <i>Framework</i>	23
5.1.14 <i>Map</i>	23
5.1.15 <i>MapTarget</i>	24
5.1.16 <i>MapSource</i>	24
5.2 MOF+ MODEL ASSOCIATIONS	25
5.2.1 <i>Contains</i>	25
5.2.2 <i>RefersTo</i>	25
5.2.3 <i>Exposes</i>	26
5.2.4 <i>IsOfType</i>	27
5.2.5 <i>Hierarchy</i>	28
5.2.6 <i>View</i>	28

5.2.7 Identify.....	29
6. EXAMPLES OF MODEL ELEMENTS.....	29
6.1 UML PROFILE FOR EDOC	30
6.2 UMM AND EBXML	32
7. CATEGORIES AND VIEWPOINTS FOR NAMESPACE.....	33
8. REFERENCE.....	36

1. Objectives and Scope

To cope with requirements for the harmonized exchanging of the business objects, such as business information, data among different organizations through EDI, the common data element specifications, message interchanging formats and protocols, such as the ISO/IEC 11179 (SC14, JTC1 SC32) and the ISO 9735 (TC 154) were standardized in the early 90s.

To follow the trends of EC and internet, a lot of industrial consortia have been in charge of the standardization of domain specific business objects including business process models and software components using common modeling facilities and exchanging facilities such as UML and XML. They are endeavor to standardize domain specific business process models which represent the best practices of businesses, and standard modeling constructs such as data elements, entity profiles and value domains at each business domains.

One of the things to be mentioned today is that most of those standard efforts tend to be focused on the contents of metamodel to represent and exchange the semantics of businesses, using the UML Stereotype mechanism and the XML.

The development of metamodels and UML profiles has been progressed through standardization activities such as UN/CEFACT and OASIS for UMM, ebXML, and OMG for MOF, XMI, CWM, EDOC, EAI, etc.

However, every standard group has to specify their metamodel scheme by their own manners. Due to lack of standards that specify common bases for consistent development and registration of metamodels, fudge duplications and inconsistencies have to be brought.

A unified framework for classifying and registering normative model elements could be required to establish harmonization of the metamodels, which are developed independently and to reuse them widely across organizations.

A useful de facto standard or draft standard developed by a standardization organization may be taken up and established as an IS of ISO/IEC/JTC1. Also it is meaningful to build a registry for metamodels based on IS or de facto standard in order to share the information about those model elements. When defining a business object model according to a metamodel and UML profile, stereotype, pattern, component, framework etc. are basic modeling construct elements to be referred as normative. The business model and information system model within an enterprise or among enterprises should be developed consistently based on those normative elements.

2. Rationale

A business object is an object that is identified to build a reusable model of business or reusable software components which should have interoperability within enterprise or in the trade of inter-enterprise. To identify and define a concrete object, firstly, concepts in the modeling target domain must be put in order, and secondly the meaning of business objects that are picked up from those concepts should be defined exactly with the relationship among each business objects.

UML provides a standard method for creating models at analysis and design phase of system development. Especially, it allows visualizing and handling a model before implementing a system. For instance, in the case of developing a building, many designs are drawn. Similarly a model described with UML diagrams allows validation of the model correctness before coding the system. Changing specification can be adopted easily and then the total cost of development and maintenance can be reduced.

Currently, many enterprise information systems are built on the platform of middleware such as CORBA, EJB, CMM. To develop an interoperable software system under such middleware environments with low cost, it is important to use standardized business objects and models. There are some merits as follows

- middleware neutral

If a business application would be built on a middleware using standardized business objects that are middleware neutral, it would be easy to transfer it into another middleware.

- inter-enterprise connection

Using standardized business objects, the inter-enterprise connection among customers, trading partner and business partner would be available. Also, if the compliant model that is created based on common business objects would be adopted, the interoperability among the subsystems of each sections of a same enterprise would be maintained in the organized way.

- Integration of enterprise legacy applications

Generally, in an enterprise, there are many assets of software, i.e. legacy applications; it is a significant issue how to continue their use. However, the integration of new applications and legacy ones could be achieved by rapping the legacy one as a business object. A legacy application can remain running on an

existing platform, and the automatic generation of bridge from a platform to another platform would be allowed.

-standardization for core models

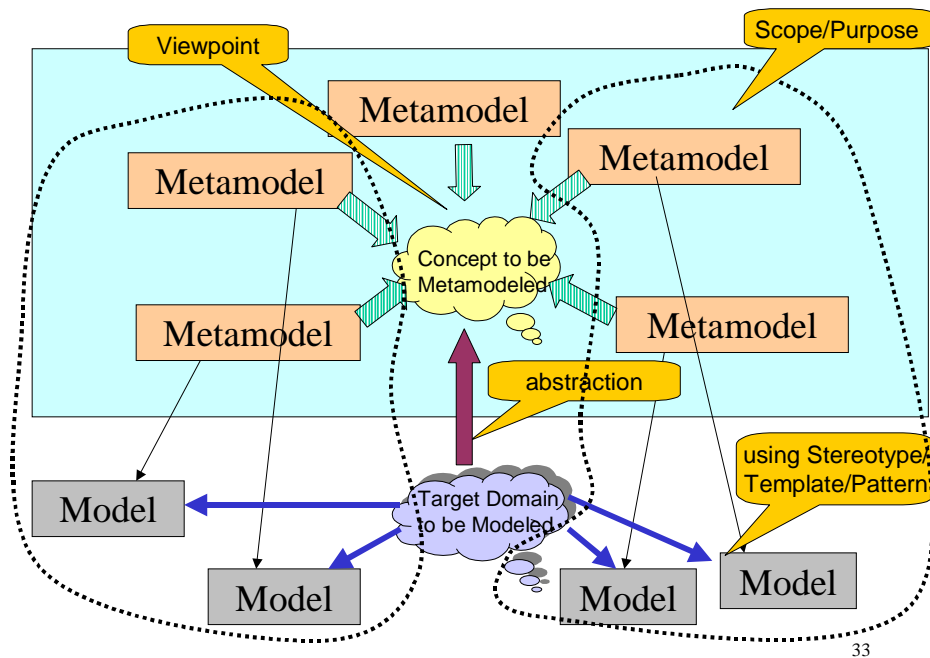
As a standard model among enterprises or in an industry, if business objects and core models would be defined with independent of any particular platform, the development of components and tools conforming standard functions or interoperability would be enable. Then, the solution business by vendors would be promoted.

-stability and reusability of standard model

In some case, the lifecycle of a business application may be longer than one of hardware or platform. If a business application would be built based on a business object model that are independent of any particular platform, remaking a same application on a new hardware or platform would be easy and flexible by generating the code of its application from the model. The much of maintenance cost of lifecycle could be reduced.

The terminology “business object” is used from various aspects, many kinds of business objects or modeling could be considered with granularities, abstraction levels of a modeling target, purposes, intents and viewpoints. For example, modeling message exchange of e-commerce, as an upper level model, a business process model such a process scenario about interaction among parties, could be considered. As a lower level model, a message format and protocol could be treated for modeling target. Moreover, business objects such as a component or framework might be developed and used for implementation level model.

Figure 1 shows the relationship between metamodel and model. Various models, for each modeled target and business domain, would be developed. In those cases, it is an important issue which stereotypes or patterns should be used as modeling construct elements. The specification of those modeling construct elements may be provided with metamodels. As shown figure 1, the abstraction of target domain to be modeled is performed in our mind and concepts to be metamodeled are emerged. The specification of those concepts is defined through matamodels from various scopes, purposes and viewpoints.



33

Figure 1. Metamodel and Model

A metamodel provides the definition and specification of a conceptual domain for its concept. The concrete stereotypes and patterns conforming the metamodel are corresponding its value domain and used to build a concrete model.

Such a metamodel is a kind of standard, stereotypes and patterns conforming its standard are also standard defined as a profile. By sharing and reusing those elements, standardization of business objects models could be archived. To improve sharable capability and reusability of object models, normative modeling constructs such prototypes and patterns must support the following features:

- The model must be consisted predefined normative modeling constructs, not only with modeling manners and notations.
- Predefined modeling constructs should include the common atomic objects, such as, Date, Currency, Country-code, which are needless to be discussed when they are used.
- Common aggregated objects, such as Customer, Company, or Order, which represent business entities, also should be predefined as normative modeling constructs, using the normative atomic objects.

- Business concept, such as, Trade, Invoice, or Settlement, which are typically represented as relationship among objects, should be defined as aggregations of the common elementary aggregated objects or simple objects. They also have to be predefined as normative modeling constructs.
- Those aggregations that can be predefined using more basic and elementary patterns as base, could be defined as object patterns.
- Patterns can represent business concept where they provide for aggregation of more elementary patterns. Therefore, aggregation or composition mechanism must be provided in patterns.
- Business rules that govern business concept can be represented with a pattern with constraints encapsulated in it. Thus, the mechanism for constraint inheritance among patterns must be provided.

Anyway, for modeling business objects, it is important issue to standardize conceptual models (i.e. metamodel) of each target domains. Nowadays, the active discussion about development of standards is continuing with organizations such as OMG, ebXML, UN/CEFACT and so on. For example, OMG that has established the MOF (Meta Object Facility) and metamodel architecture, is promoting the MDA (Model Driven Architecture).

3. Basic Concepts

This proposal draft standard is based on ISO/IEC/JTC1 FDS 11179-3 and MOF adopted by OMG. First of all in this section, basic concepts of MOF and 11179-3 are introduced. Then, we show how the concepts of 11179 are merged into the concepts of MOF. Its enhanced MOF is called MOF+.

3.1 Overview of MOF

The MOF is an adopted technology by OMG for defining metadata and representing it as CORBA objects. Metadata is itself a kind of information, and can accordingly be described by other metadata. In MOF terminology, metadata that describes metadata is called meta-metadata, and a model that consists of meta-metadata is called a metamodel.

One kind of metamodel plays a central role in the MOF. A MOF metamodel defines the abstract syntax of the metadata in the MOF representation of a model. Since there are many kinds of metadata in a typical system, the MOF framework needs to support many different MOF metamodels. The MOF integrates

these metamodels by defining a common abstract syntax for defining metamodels. This abstract syntax is allied the MOF Model and is a model for metamodels; i.e. a meta-metamodel. The MOF metadata framework is typically depicted as a four layer architecture (M0,M1,M2,M3) . There are following features on MOF-based metamodel and UML profile;

- Using a restricted subset of the UML notation
- Providing common style of describing metamodel
- A metamodel at M2 level is as an abstract syntax of models at M1level
- A model at M1level is an expression of the metamodel
- UML profile (stereotype, tagged values etc.) is as additional constraints for metamodel
- Mapping between the metamodel and the UML profile is needed as a basis for the development of toolsEnable exchanging metamodel/model/data between toolsIn the proposed draft standard, the features of MOF are preserved and the following facilities are enhanced;

- specifying relationships among MOF-based metamodels (of each standard) or between M2 level and M1level
- classifying and registering (based on ISO/IEC 11179-3 rev.) namespaces of each metamodel and UML profile, also patterns as a model instance
- defining mapping a part of metamodel at M2 level or model or data at M1 level between MOF-based metamodels

Overview of the Metamodel Framework

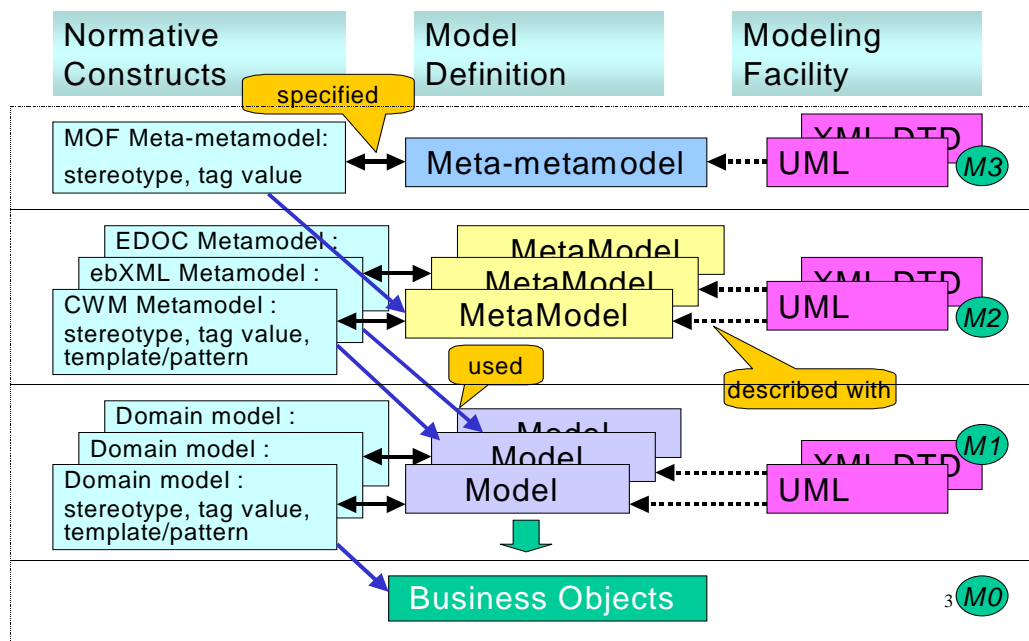


Figure 2. Metamodel Framework

3.2 Overview of 11179

The metamodel for metadata registry in ISO/IEC 11179-3 rev. is developed. Data modeling is founded on the theory that all data describes properties (attributes) of objects in the natural world. Data represents the properties of these things. The basic units of data are data elements. This metamodel uses many of the same conceptual data structures used in data modeling.

The more important key components of the metadata registry are as follows.

Data Element Concept:

An idea that can be represented in the form of a data element, described independently of any particular representation.

Conceptual Domain:

A set of possible value meanings of a data element expressed without representation.

Value Domain:

Restricted permissible values.

Data Element:

A unit of data for which the definition, identification, representation, and permissible values are specified by means of a set of attributes.

3.3 Classification Framework for Modeling BO

11179	MOF+	Description
Data Element Concept:	Namespace	Name of ModelElement to be classified, e.g. stereotype name, pattern name, component name, etc.
Conceptual Domain:	TypedModelElement	Specification of Typed ModelElement, e.g. metamodel, profile, standard etc.
Value Domain:	ModelInstance	Represented Model Instance of Typed Model Element, e.g. pattern, stereotyped element, etc.
Data Element:	ModelElement	Concrete ModelElement provided with specific Model Instance.

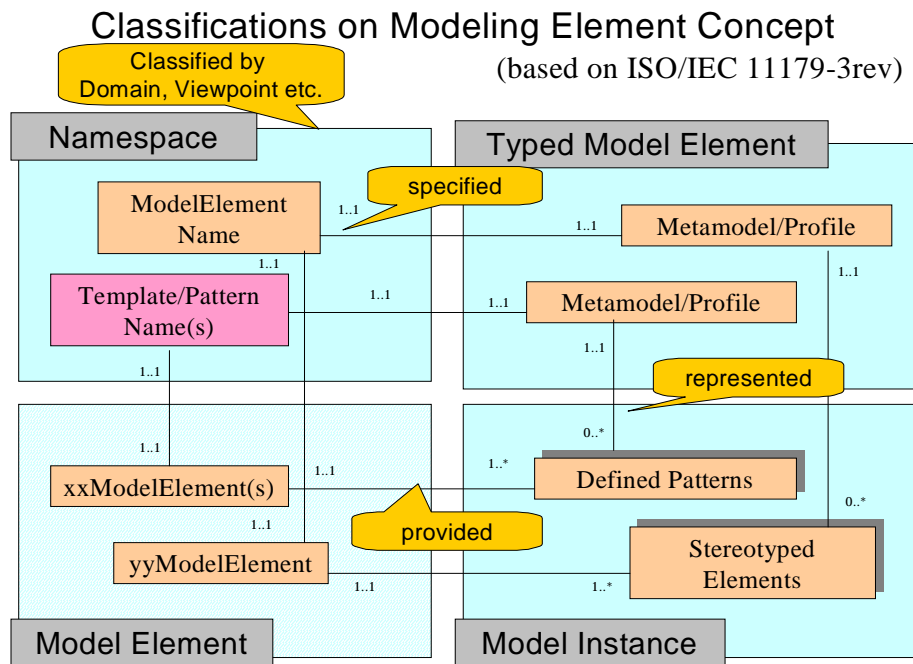


Figure 3. Modeling Element Concept

4. Overview on how this draft standard to be used

This draft standard is to define rigorously the relationship between metamodel and model, the framework of registering, sharing and reusing normative modeling constructs. In the layer M2 of the metamodel architecture, standards related to business objects modeling, which are developed by each standardization organizations, are registered with those namespace including defined concepts. In addition, the model instances conforming those standard, for instance concrete stereotypes or patterns, also are registered. Users of the registry, such as modelers, picks up and uses some stereotypes and patterns that are appropriate to build own business objects model in the M1 layer. The M1 layer has similar structure to the M2 layer that consists of namespace, typed model constructs, model instance and model element.

For example, names of business processes are registered in “namespace” of the M1 layer and business process models defined using normative modeling constructs of the M2 layer are registered in “typed model element” of the M1 layer. In “model instance” of the M1 layer, concrete products satisfying the specification of those models are registered by each vendors that develop and ship them.

Also, the relationship among typed model elements such a reference needs to be described definitely. In this draft standard, the facility to define reference of any metamodel over different standards or mapping for data exchange should be provided. Moreover, in the case of registering the model instance, it is necessary to describe the conformance level such how much and what parts of metamodels each model

instance accords with or not. Then this draft standard should provide the framework of defining those things.

Table 1 shows that there are different registers and users against the registry conforming this draft standard. Supposed organizations and users who register metamodels or patterns for the M2 layer and models or products for the M1 layer respectively, are listed.

In this draft standard, concerning metamodel for the M2 layer and model for the M1 layer, as described above, the purpose is specifying the framework for classifying namespace and defining the relationship and meaning among metamodel elements or metamodel layers. The M3 layer of MOF might be enhanced to provide the facility for treating those concepts.

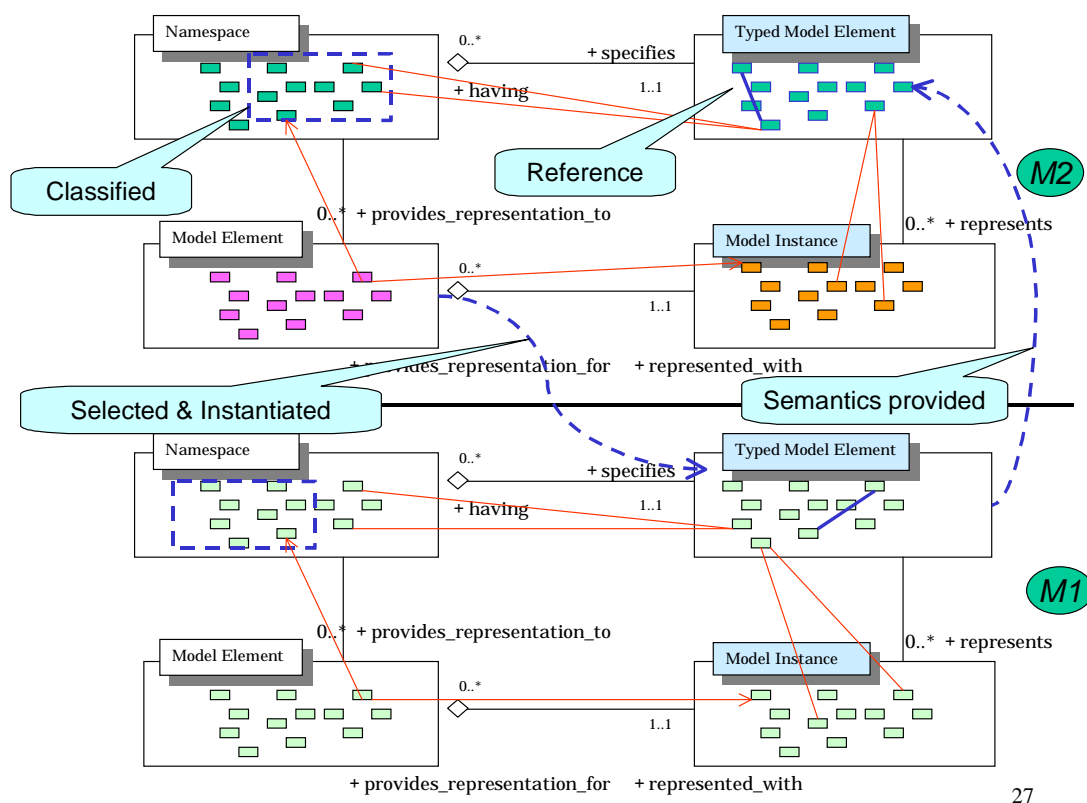


Figure 4.

Table 1.

Layer	Metamodel Elements	Developer/Register	User
M 2	Namespace	Standardization organization	Standard maker Developer of patterns and stereotypes
	Typed model element	Standardization organization	Standard maker Developer of patterns and stereotypes
	Model instance	Standardization organization Developer of patterns and stereotypes	Industry standard maker
	Model element	(not public. Managed by each industries)	(within each industries)
M1	Namespace	Organization for developing industry standard model	Developer of products conforming standard model
	Typed model element	Organization for developing industry standard model	Developer of products conforming standard model
	Model instance	Enterprise of developing products conforming standard model.	User of products
	Model element	(not public. Managed by within enterprise)	(within each enterprises)

5. MOF+ metamodel (normative part of this draft standard)

In this section, the metamodel of MOF+ is defined based on MOF. Figure xx shows the class diagram of MOF+ that consists of metamodel elements defined by MOF and colored ones enhanced by this draft standard. The diagram of Figure xx is the detail of enhanced parts of MOF+.

The precise specification of MOF+ should be developed through technical discussion of SC32 after NWI proposal are adopted. Here, as a working draft, the overview of specification are presented.

This model also provides a mechanism that can describe mapping relationship between meta-elements that are contained in different metamodels. As a rule, mappings are set onto meta-objects that have some

similarity or classified as a same category by the classification mechanism in this document.

In this document, to realize classification and registration of metamodels, we introduced special type of Classification called Map, and special type of Association called MapSource and MapTarget., as an addition to MOF. Although the MOF metamodel can realize these meta-objects, we deliberately introduced above subclasses to express mapping relations explicitly. Therefore, this addition is not an intrinsic extension.

This architecture provides just a framework (or a container) of mapping, not a mapping definitions themselves (actual mapping).

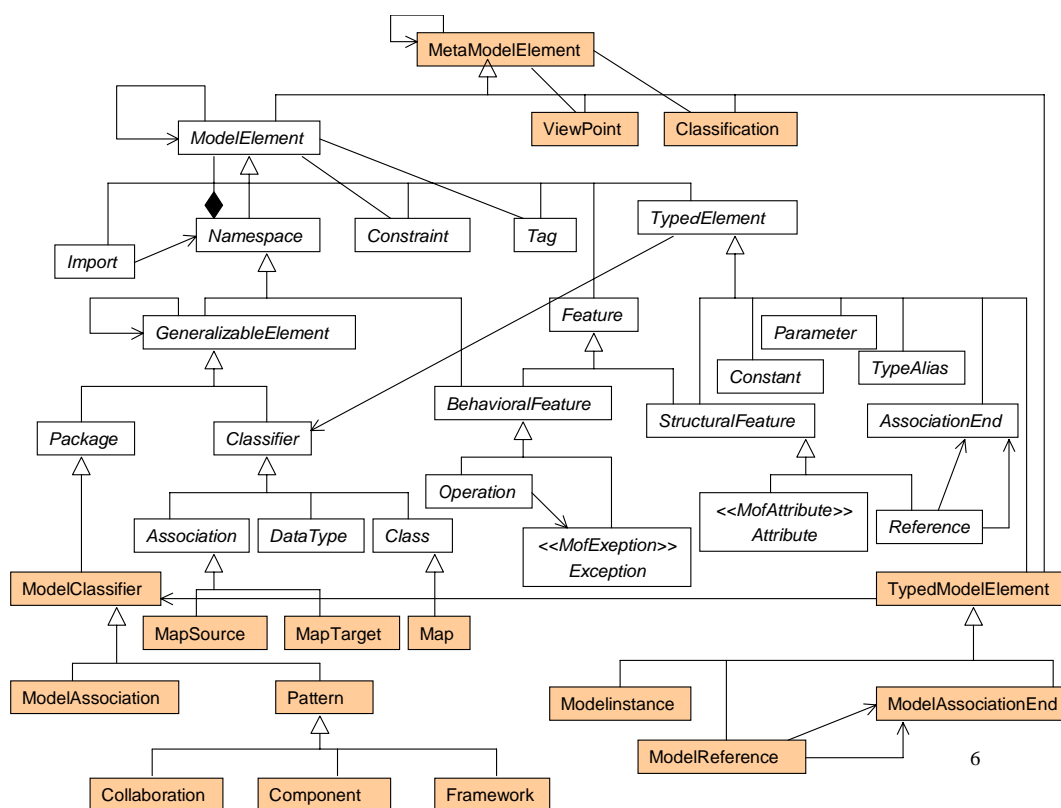


Figure 5. MOF and MOF+

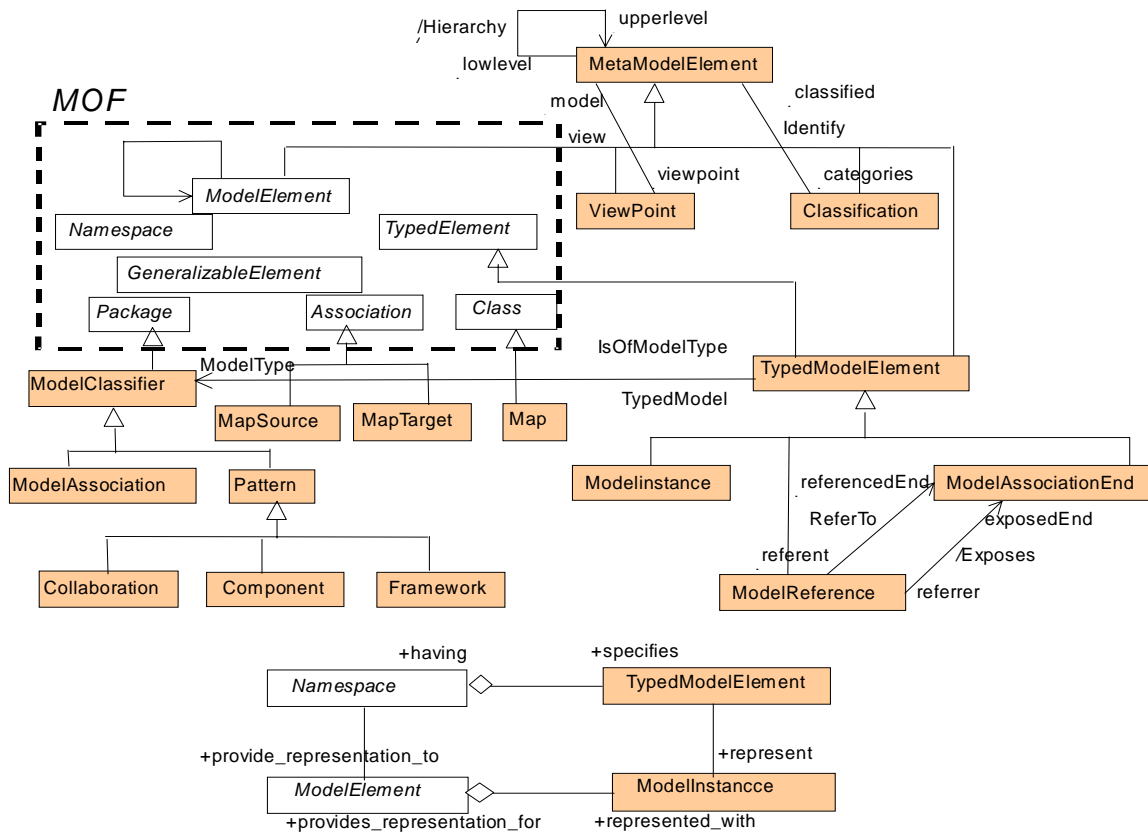


Figure 6. MOF+

5.1 MOF+ Model Classes

5.1.1 MetaModelElement

The **MetaModelElement** is a root class of MOF+ metamodel that is to classify basic and atomic construct elements of metamodels.

Class Heading

Superclasses

Contained Elements

Attributes

Name: Provides a meta-modeler supplied name that uniquely identifies the **MetaModelElement** in the context of the **MetaModelElement**'s containing **Namespace**. When choosing a **MetaModelElement**'s name, the meta-modeler should consider the rules for translating names into identifiers in the relevant mappings.

Annotation: Provides an informal description of the **MetaModelElement**.

QualifiedName: Provides a unique name for the MetaModelElement within the context of its outermost containing Package. The qualifiedName is a list of NameType values consisting of the names of the MetaModelElement, its container, its container's container and so on until a non-contained element is reached. The first member of the list is the name of the non-contained element.

References

Container: identifies the Namespace that contains the MetaModelElement. Since the Contains Association is a Composite Association, any MetaModelElement can have at most one container, and the containment graph is strictly tree shaped.

RequiredElements: Identifies the MetaModelElements on whose definition the definition of this MetaModelElement depends. For a definition of dependency, see Section xx, "DependsOn," on page xx.

Constraints: Identifies the set of Constraints that apply to the MetaModelElement. A Constraint applies to all instances of the MetaModelElement and its sub-Classes.

Operations

Verify: Each MetaModelElement is capable of checking its own correctness, as defined by the inherent properties of meta-models described in this specification, and constraints that hold over the MetaModelElement. The client of the operation specifies whether the operation should propagate to any MetaModelElements that this MetaModelElement might contain (if it is capable of containing elements), or whether it should return after only checking itself. The verify operation checks inherent constraints on the object and its attributes plus any constraints contained by the object. The operation returns valid if all verification checks passed; otherwise, it returns invalid. A parameter returns representations of any constraint violations detected. If the operation returns invalid, this parameter must not be empty. When the depth argument is deep, and this element (and, by definition, all its dependent elements) are published, the operation returns published.

IsFrozen: Reports the freeze status of a MetaModelElement. A MetaModelElement, at any particular time, is either frozen or not frozen. All MetaModelElements of a published model are permanently frozen.

FindRequiredElements: Supports selecting a subset of the MetaModelElements that this one depends on, based on their dependency categories. The "kinds" argument gives a bag of DependencyKinds that are of interest to the caller. String constants for the standard dependency categories are given in Section xx, "MOF Model Constants," on page xx and their meanings are defined in Section xx, "DependsOn," on page xx. In this context, the AllDep pseudo-category (i.e., "all") is equivalent to passing all of the standard categories in the "kinds" argument, and the IndirectDep pseudo-category (i.e., "indirect") is ignored. If the "recursive" argument is "false", the operation return the direct dependents only. If it is "true" all dependents in the transitive closure of the specified "kinds" are returned.

IsVisible: Returns true. This operation is reserved for future use when the MOF visibility rules have stabilised. Then it will determine whether the supplied otherElement is visible to this MetaModelElement.

IsRequiredBecause: This operation performs two functions:

- It checks whether this *MetaModelElement* directly or indirectly depends on the *MetaModelElement* given by “*otherElement*”. If it does, the operation’s result is “true;” otherwise, it is “false.”
- If a dependency exists (i.e., the result is “true”), the operation returns a *DependencyKind* in “reason” that categorizes the dependency. String constants for the *DependencyKind* categories are given in Section xx, “MOF Model Constants,” on page xx and their meanings are defined inSection xx, “DependsOn,” on page xx. If the dependency is indirect, *IndirectDep* is returned. If there are multiple dependencies, any category that applies may be returned in “reason.” If no dependencies exist, an empty string is returned in “reason.”

Constraints

A MetaModelElement that is not a Package must have a container.

The attribute values of a MetaModelElement which is frozen cannot be changed.

A frozen MetaModelElement which is in a frozen Namespace can only be deleted, by deleting the Namespace.

The link sets that express dependencies of a frozen Element on other Elements cannot be explicitly changed.

5.1.2 ViewPoint

Class Heading

Superclasses

Contained Elements

Attributes

References

Operations

Constraints

5.1.3 Classification

Class Heading

Superclasses

Contained Elements

Attributes

References

Operations

Constraints

5.1.4 ModelClassifier

A package is formed as a composition of MetaModelElements. A package defines a modeling unit, models are constructed and presented as packages. A model is a package. Packages are also used as organizational constructs in modeling. Nesting, importation, and generalization are used to manage the complexity of models.

Class Heading

Superclasses

GeneralizableElement

Contained Elements

Package, Class, Association, DataType, Exception, Import, Constraint, Constant

Attributes

References

Operations

Externalize: *Externalize the Package and all of its MetaModelElements (transitive closure on the containment hierarchy) in a format specified by the format parameter, into a stream of type any.*

Internalize: *Reify a model encoded in “stream” in some external format specified by “format” as a MOF Package.*

Constraints

A Package may only contain Packages, Classes, DataTypes, Associations, Exceptions, Constraints, Imports and Tags.

Packages cannot be declared as abstract.

5.1.5 TypedModelElement

The TypedModelElement type is an abstraction of TypedElement and MetaModelElements that require a type as part of their definition. A TypedModelElement does not itself define a type, but is associated with a ModelClassifier.

Class Heading

Superclasses

Contained Elements

Attributes

References

Type: *Provides the representation of the type supporting the TypedModelElement through this reference.*

Operations

Constraints

An Association cannot be the type of a TypedModelElement.

A TypedModelElement can only have a type that is visible to it.

5.1.6 ModelInstance

Class Heading

Superclasses

Contained Elements

Attributes

References

Operations

Constraints

5.1.7 ModelAssociationEnd

A ModelAssociation is composed of two ModelAssociationEnds. Each ModelAssociationEnd defines a Classifier participant in the Association, the role it plays, and constraints on sets of the Classifier instances participating. An instance of a ModelAssociationEnd is a LinkEnd, which defines a relationship between a link, in instance of an Association, and an instance of the ModelAssociationEnd's Classifier, provided in its type attribute.

Class Heading

Superclasses

Contained Elements

Attributes

Multiplicity: *Multiplicity defines constraints on sets of instances. Each instance of the Classifier defined by the opposite ModelAssociationEnd's type defines a set which this multiplicity attribute constrains. Given one of those instances, x, the set is defined as the instances connected by LinkEnds of this ModelAssociationEnd to that instance x. Refer to Section xx, "MultiplicityType," on page xx for a description on how the multiplicity attribute constrains a set. In its use in describing ModelAssociationEnds, isUnique has been constrained to be true, as a simplification. This constraint means that the same two instances cannot participate in more than one Link while participating under the same ModelAssociationEnd. Normally, two instances cannot be linked by more than one Link of an Association at all. But when the ModelAssociationEnd types allow the two instances switch ends, they can form a second Link without violating the isUnique constraint.*

Aggregation: *Certain associations define aggregations - directed associations with additional semantics (see Section xx, "Aggregation Semantics," on page xx). When an ModelAssociationEnd is defined as*

composite or shared, the instance at “this” end of a Link is the composite or aggregate, and the instance at the “other” end is the component or subordinate.

IsNavigable: *The isNavigable attribute determines whether or not the ModelAssociationEnd supports link “navigation”. This has two implications:*

- *A Class defined with an appropriate Reference supports navigation of links from one Class instance to another. If isNavigable is false for an AssociationEnd, no such References may be created.*
- *Setting isNavigable to false also suppress a mapping’s mechanisms for indexing links based on this ModelAssociationEnd.*

IsChangeable: *The isChangeable attribute restricts the capability to perform actions that would modify sets of instances corresponding to this ModelAssociationEnd (the same sets to which multiplicity is applied). Specifically, the set may be created when the instance defining the set - the instance at the opposite end of the Links - is created. This attribute does not make the set immutable. Instead, it affects the generation of operations in Model Elaboration which would allow modification of the set. For IDL generation, the only operation that allows the set to be modified would be one or more factory operations that create the instance and create the set. The modeler is free to define specific operations that allow modification of the set. Note that defining this ModelAssociationEnd with isChangeable equals false places restrictions on the changeability of the other ModelAssociationEnd, due to their interdependence.*

References

Operations

OtherEnd: *Provides the other ModelAssociationEnd (i.e., not this one) in the enclosing Association.*

Constraints

The type of an ModelAssociationEnd must be Class.

The “isUnique” flag in an ModelAssociationEnd’s multiplicity must be true.

An Association cannot have two ModelAssociationEnds marked as “ordered.”

An Association cannot have an aggregation semantic specified for both ModelAssociationEnds.

5.1.8 ModelReference

A ModelReference defines a Classifier’s knowledge of, and access to, links and their instances defined by an Association. Although a ModelReference derives much of its state from a corresponding ModelAssociationEnd, it provides additional information; therefore, the MOF cannot adequately represent some meta-models without this mechanism. The inherited attributes defined in StructuralFeature (multiplicity and is_changeable) are constrained to match the values of its corresponding ModelAssociationEnd. However, it has its own visibility, name, and annotation defined. For further discussion on ModelReference, its purpose, and how it derives its attributes, see Section xx, “ModelAssociations,” on page xx.

Note – *When creating a ModelReference, values for the inherited attributes of multiplicity and is_changeable must be supplied. These must be the same as the corresponding attributes on the ModelAssociationEnd to which*

the ModelReference will subsequently be linked.

Class Heading

Superclasses

Contained Elements

Attributes

References

ModelReferencedEnd: *The ModelReferencedEnd of a ModelReference is the end representing the set of LinkEnds of principle interest to the ModelReference. The ModelReference provides access to the instances of that ModelAssociationEnd's class, which are participants in that ModelAssociationEnd's Association, connected through that ModelAssociationEnd's LinkEnds. In addition, the ModelReference derives the majority of its state information - multiplicity, etc., from that ModelReference.*

ExposedEnd: *The exposedEnd of a ModelReference is the ModelAssociationEnd representing the end of the ModelReference's owning Classifier within the defining ModelAssociation.*

Operations

Constraints

The multiplicity for a ModelReference must be the same as the multiplicity for the referenced ModelAssociationEnd.

Classifier scoped ModelReferences are not meaningful in the current MI level computational model.

A ModelReference can be changeable only if the referenced ModelAssociationEnd is also changeable.

The type attribute of a ModelReference and its referenced ModelAssociationEnd must be the same.

A ModelReference is only allowed for a navigable ModelAssociationEnd

The containing Class for a ModelReference must be equal to or a subtype of the type of the ModelReference's exposed ModelAssociationEnd.

The referenced ModelAssociationEnd for a ModelReference must be visible from the ModelReference.

5.1.9 ModelAssociation

An association defines a classification over a set of links, through a relationship between Classifiers. Each link which is an instance of the association denotes a connection between object instances of the Classifiers of the ModelAssociation. The MOF restricts associations to binary, restricting each link to two participating objects. This restriction also means that the association is defined between two Classifiers (which may be the same Classifier). The name of the ModelAssociation is considered directional if it provides a clearer or more accurate representation of the association when stated with one participating class first rather than the other. For instance, Operation CanRaise Exception is correct; Exception CanRaise Operation is incorrect.

The definition of a ModelAssociation requires two ModelAssociationEnds. If the name of the association is directional, the name is understood to read in the order: first contained element; association name; second

contained element. These contained elements are *ModelAssociationEnd* instances, and the reading of the subject; verb; object uses either the *ModelAssociationEnd* name or the *ModelAssociationEnd*'s class name. The onus is on the MOF user to determine whether the name is directional, and to place the *ModelAssociationEnd*s in proper order within the *ModelAssociation*'s contents to support the name direction. The representation of a Classifier's knowledge of its participation in an association requires the use of a *ModelReference*.

Class Heading

Superclasses

Contained Elements

Attributes

IsDerived: A derived association has no Links as instances. Instead, its Links are derived from other information in a meta-model. The addition, removal, or modification of a derived *ModelAssociation*'s Link causes the information upon which the *ModelAssociation* is derived to be updated. The results of such an update are expected to appear, upon subsequent access of the derived *ModelAssociation*'s Links, to have the same effect as an equivalent operation on an *ModelAssociation* which is not derived

References

Operations

Constraints

An Association may only contain ModelAssociationEnds, Constraints and Tags.

Inheritance / generalization is not applicable to ModelAssociations.

The values for "isLeaf" and "isRoot" on an ModelAssociation must be true.

An ModelAssociation cannot be abstract.

ModelAssociations must have visibility of "public."

An ModelAssociation must be binary (i.e., it must have exactly two ModelAssociationEnds).

5.1.10 Pattern

The pattern is a kind of model construct elements that is a reusable piece of model and generally applicable to the similar model. It can be treated as a type (or template). The metameta model elements such Collaboration, Component, Framework are subclasses of the pattern.

Class Heading

Superclasses

Contained Elements

Attributes

References

Operations

Constraints

5.1.11 Collaboration

Class Heading

Superclasses

Contained Elements

Attributes

References

Operations

Constraints

5.1.12 Component

Class Heading

Superclasses

Contained Elements

Attributes

References

Operations

Constraints

5.1.13 Framework

Class Heading

Superclasses

Contained Elements

Attributes

References

Operations

Constraints

5.1.14 Map

Map provides a mechanism that can describe mapping relationship between meta-elements that are contained in different meta-models. As a rule, mappings are set onto meta-objects that have some similarity or classified as a same category by the classification mechanism in this document.

SuperClasses

Classifier.

Attributes

References

Operations

none

Constraints

<TBD>

5.1.15 MapTarget

MapTarget represents a target meta-class of a map.

SuperClasses

Association

Attributes

References

Operations

none

Constraints

<TBD>

5.1.16 MapSource

MapSource represents a source of meta-class of a map.

SuperClasses

Association

Attributes

References

Operations

none

Constraints

<TBD>

5.2 MOF+ Model Associations

5.2.1 Contains

A meta-model is defined through a composition of *MetaModelElements*. A *Namespace* defines a *MetaModelElement* which composes other *MetaModelElements*. Since *Namespace* has several subclasses, there is a sizable combinatorial set of potential *Namespace-MetaModelElement* pairings. However, some of these pairings are not appropriate for building an object-oriented meta-model, such as a *Class* containing a *Package* (see Section xx “The MOF Model Containment Hierarchy,” on page xx. This approach factors the container mechanisms into one abstraction, and allows the greatest flexibility for future changes to the MOF Model.

Association Heading

Ends

container

<i>Each Namespace is a composition of zero or more MetaModelElements.</i>	
<i>class:</i>	<i>Namespace</i>
<i>multiplicity:</i>	<i>zero or one</i>
<i>aggregation:</i>	<i>Namespace forms a composite aggregation of MetaModelElements</i>

containedElement

<i>Each MetaModelElement, with the exception of top-level packages participates in the association as a containedElement.</i>	
<i>class:</i>	<i>MetaModelElement</i>
<i>multiplicity:</i>	<i>zero or more; ordered</i>

Derivation

DataTypes

Exceptions

Constants

Constraints

5.2.2 RefersTo

A *ModelReference* derives most of its state from the *ModelAssociationEnd* that it is linked to based on this Association. For a *Class* defined with a *ModelReference*, each of its instances can be used to access the

referenced object or objects. Those referenced objects will be of the Class defined by this referencedEnd ModelAssociationEnd, playing the defined end.

Association Heading

End

referent

<i>The Reference which is providing the reference through which instances playing the end-defined by the AssociationEnd can be accessed.</i>	
<i>class:</i>	<i>Reference</i>
<i>multiplicity:</i>	<i>zero or more; not ordered (an AssociationEnd may or may not be used by any number of References).</i>

referencedEnd

<i>The AssociationEnd which provides the majority of information for the Reference, including the LinkEnds which supply the referenced instances.</i>	
<i>class:</i>	<i>AssociationEnd</i>
<i>multiplicity:</i>	<i>exactly one</i>

Derivation

DataTypes

Exceptions

Constants

Constraints s

5.2.3 Exposes

A ModelReference defines a reference for a Class. For an instance of that class, which holds one or more links to some object or objects conforming to the reference, the instance will be playing the role (end) defined by the ModelAssociationEnd in this Association. Although this association can be derived in the current MOF, the use of n-ary associations, where a single Class has multiple ends specification of this Association, is necessary.

Association Heading

derived

Ends

referrer

<i>The ModelReference that is providing the exposedEnd's class instances within the ModelReference's Classifier.</i>	
<i>class:</i>	<i>ModelReference</i>

<i>multiplicity:</i>	<i>zero or more; not ordered (a ModelAssociationEnd may or may not be used by any number of ModelReferences).</i>
<i>changeable:</i>	<i>yes</i>

exposedEnd

<i>The ModelAssociationEnd representing the ModelReference's owning Classifier's end in the Association.</i>	
<i>class:</i>	<i>ModelAssociationEnd</i>
<i>multiplicity:</i>	<i>exactly one</i>
<i>changeable:</i>	<i>yes</i>

Derivation

DataTypes

Exceptions

Constants

Constraints

See For a given ModelReference, the Link of this ModelAssociation is derived as follows:

- *The referrer's ModelReference is the given ModelReference.*
- *The exposedEnd's ModelAssociationEnd is the given ModelReference's referent's container ModelAssociation's other ModelAssociationEnd.*

5.2.4 IsOfType

A Link between a TypedElement subclass and a Classifier supports the definition of the TypedElement.

Association Heading

Ends

type

<i>The type defining the TypedElement.</i>	
<i>class:</i>	<i>Classifier</i>
<i>multiplicity:</i>	<i>exactly one</i>

typedElements

<i>The set of typed elements supported by a Classifier.</i>	
<i>class:</i>	<i>TypedElement</i>
<i>multiplicity:</i>	<i>zero or more</i>

Derivation
DataTypes
Exceptions
Constants
Constraints

5.2.5 Hierarchy

Association Heading

Ends

upperlevel

<i>The type defining the TypedElement.</i>	
<i>class:</i>	<i>Classifier</i>
<i>multiplicity:</i>	<i>exactly one</i>

lowerlevel

<i>The set of typed elements supported by a Classifier.</i>	
<i>class:</i>	<i>TypedElement</i>
<i>multiplicity:</i>	<i>zero or more</i>

Derivation
DataTypes
Exceptions
Constants
Constraints

5.2.6 View

Association Heading

Ends

model

<i>The type defining the TypedElement.</i>	
<i>class:</i>	<i>Classifier</i>
<i>multiplicity:</i>	<i>exactly one</i>

viewpoint

<i>The set of typed elements supported by a Classifier.</i>	
<i>class:</i>	<i>TypedElement</i>
<i>multiplicity:</i>	<i>zero or more</i>

Derivation

DataTypes

Exceptions

Constants

Constraints

5.2.7 Identify

Association Heading

Ends

classified

<i>The type defining the TypedElement.</i>	
<i>class:</i>	<i>Classifier</i>
<i>multiplicity:</i>	<i>exactly one</i>

categories

<i>The set of typed elements supported by a Classifier.</i>	
<i>class:</i>	<i>TypedElement</i>
<i>multiplicity:</i>	<i>zero or more</i>

Derivation

DataTypes

Exceptions

Constants

Constraints

6. Examples of model elements

6.1 UML profile for EDOC

As an example, the metamodels on UML profile for EDOC that are being developed in OMG, are taken to show the framework of this draft standard. In EDOC, the concepts such as Business Transaction, Business Collaboration, Document Flow and Choreography within Collaboration are specified under metamodels such as Business Transaction Metamodel, Business Collaboration Metamodel. A set of stereotypes is also defined as a UML profile. Some candidates for patterns are suggested in the EDOC submission and more elaborated patterns might be developed. In figure 7, those patterns are given as Business Transaction Patterns and Business Collaboration Patterns.

In the EDOC, the basic concepts, that are needed to build the model of business process for an enterprise, are defined in its metamodels. Those concepts are common modeling constructs for modeling various businesses. For instance, when standard business models of an industry are developed, appropriate stereotypes and patterns to express particular business processes may be selected, the detail of industry standard models is specified using those modeling constructs. In figure xx, the `OurBusinessCollaborations` and `OurBusinessTransactions` are normative modeling constructs. And in the cases of modeling procurement business process for a specific industry, we could consider “`ProcurementBusiness Collaboration`” and “`ProcurementBusiness Transactions`” as elements of a namespace and register their specifications described with metamodels as typed model elements.

There would be lots of products satisfying the specification of a standard model, would be registered by vendors. For instance, “`MyBusinessCollaboration Subsystem`” and “`MyBusinessTransaction Subsystem`” in the model elements are declared that my company’s subsystem is implemented using concrete products selected from the registry. If those information about subsystems could be exchanged between trading parties, the high interoperability such data exchange among parties could be achieved easily. Figure 7 and 8 show the examples of registry for patterns and stereotypes respectively.

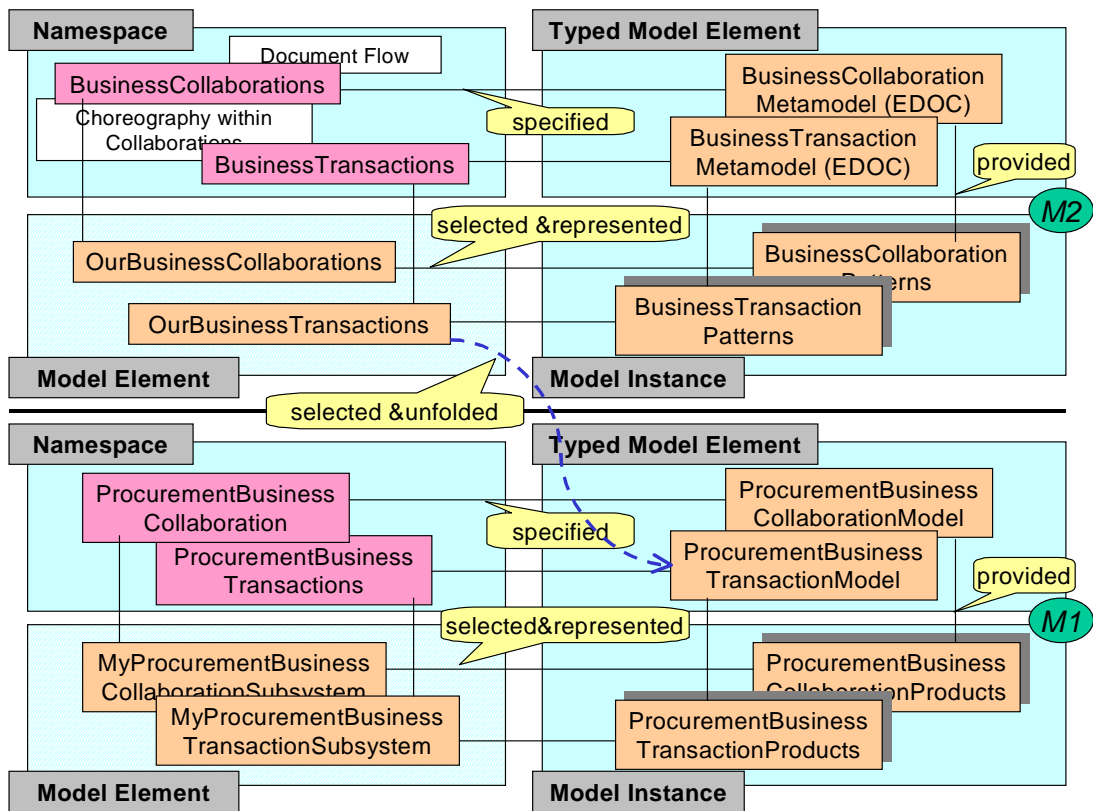


Figure 7.

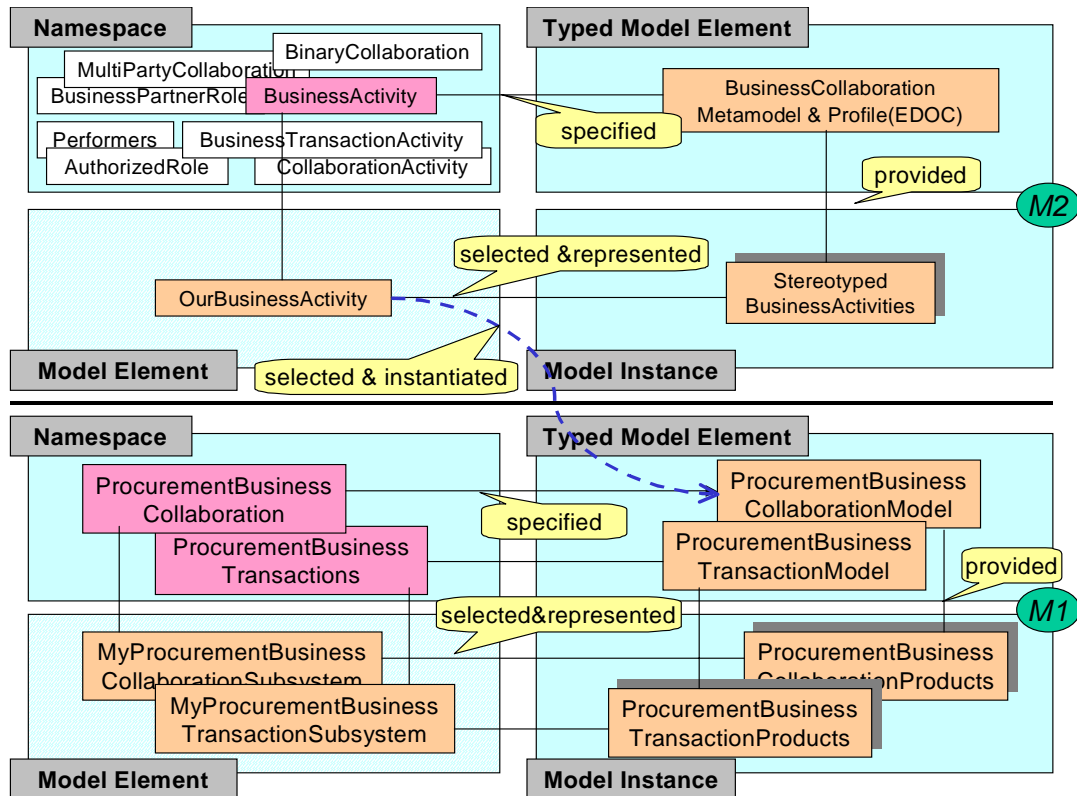


Figure 8.

6.2 UMM and ebXML

The standardization activities for e-business are in progress by UN/CEFACT UMM and ebXML. In the modeling methodology UMM, the basic concepts are considered from viewpoint of such “Business Requirement View” and “Business Transaction View”. The metamodels for those concepts are provided as a specification. The patterns such “Business Requirement Patterns” and “Business Transaction Patterns” are suggested as modeling constructs conforming metamodels. In the same as an EDOC example, modeling the business process of procurement for a particular industry is performed using those patterns and products according to the model are developed and registered by vendors. Also, concerning stereotypes, their specification, registration and reuse are the same way. See figure 9.

The business object models based on UMM and ebXML are related to ones based on EDOC. By defining the relationship between both metamodels, the high interoperability of data exchange between both subsystems that adopt different metamodels respectively, can be obtained. For that reason, it is necessary to define a kind of mapping for data exchange, in this draft standard, the framework such data mapping among metamodels would be provided.

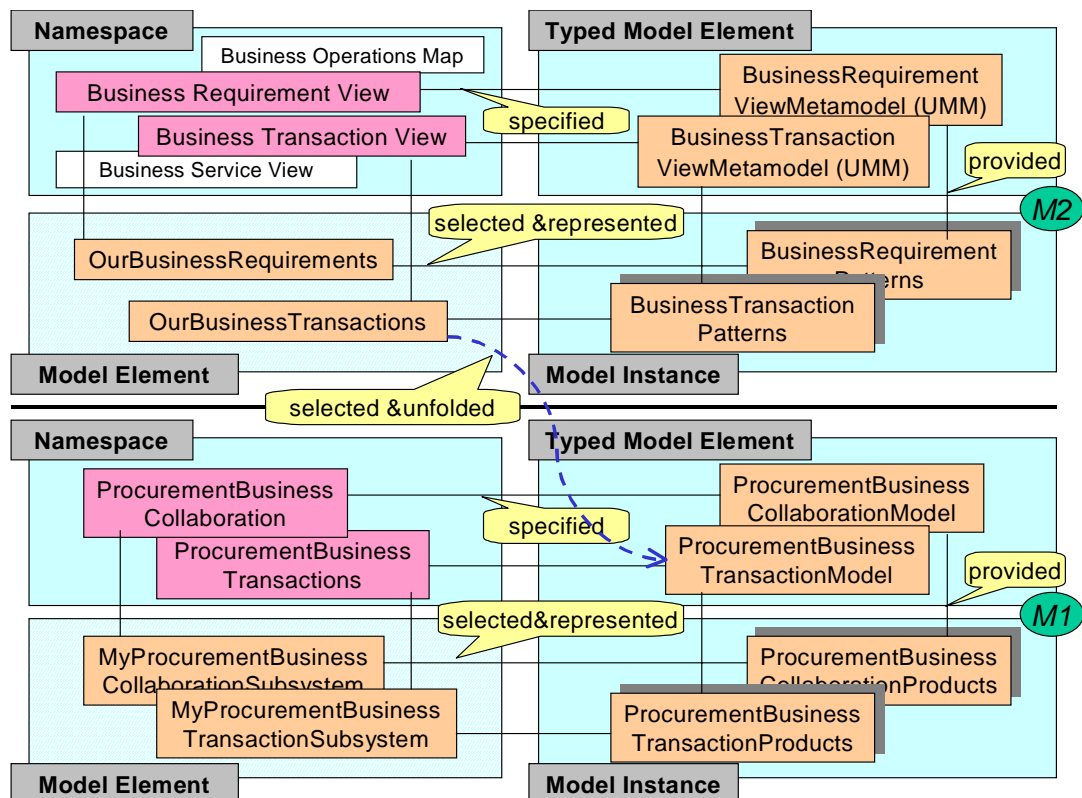


Figure 9.

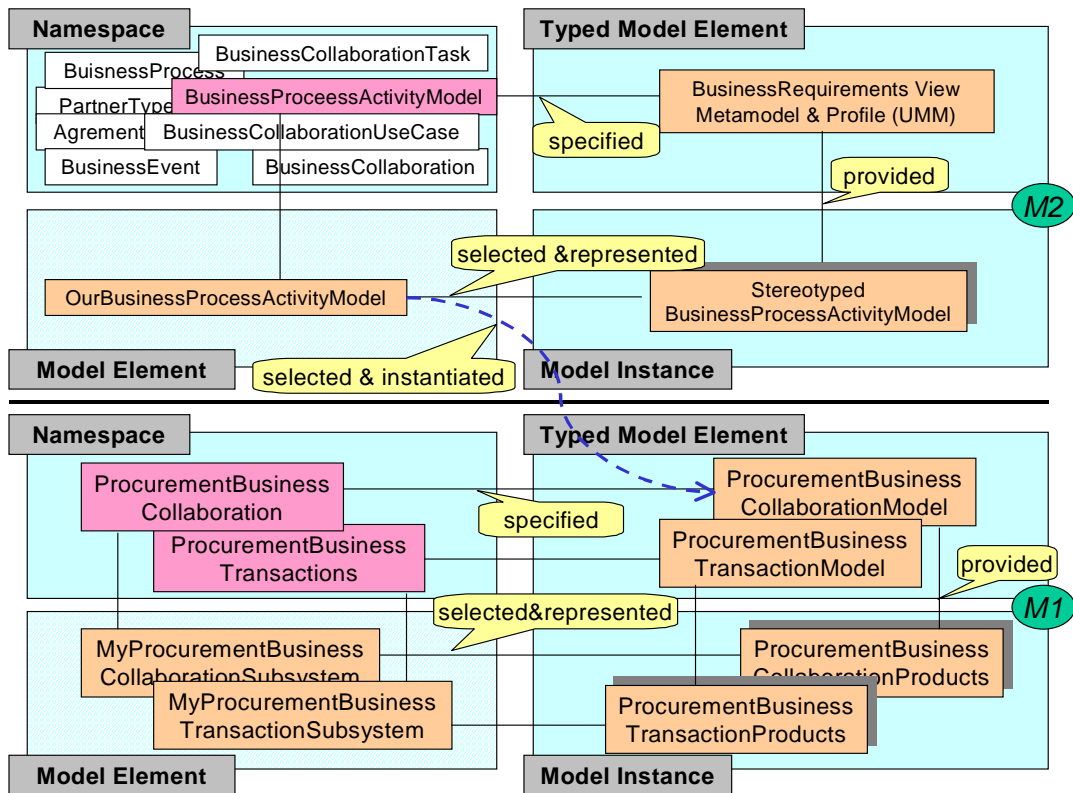


Figure 10.

7. Categories and Viewpoints for Namespace

The namespace would be managed with classifying concepts by their categories and viewpoints. The namespace compose of sets of names, a set of names related to a particular standard is registered as an element into the namespace. A namespace can have some sub-namespaces that is also registered with the more detail category and viewpoint.

Figure xx shows the example of namespace of M2 layer including the names of EDOC and UMM described above. The namespace identifiers BT, BC, BP and BS stand for sub-namespaces. Table xx shows the defined list of categories and viewpoints assigned for each namespace or name. In table xx, the candidates of business object categories and viewpoints are picked up. Those categories and viewpoints, and their values should be defined previously and the element of namespace would be registered with selecting appropriate values of categories or viewpoints.

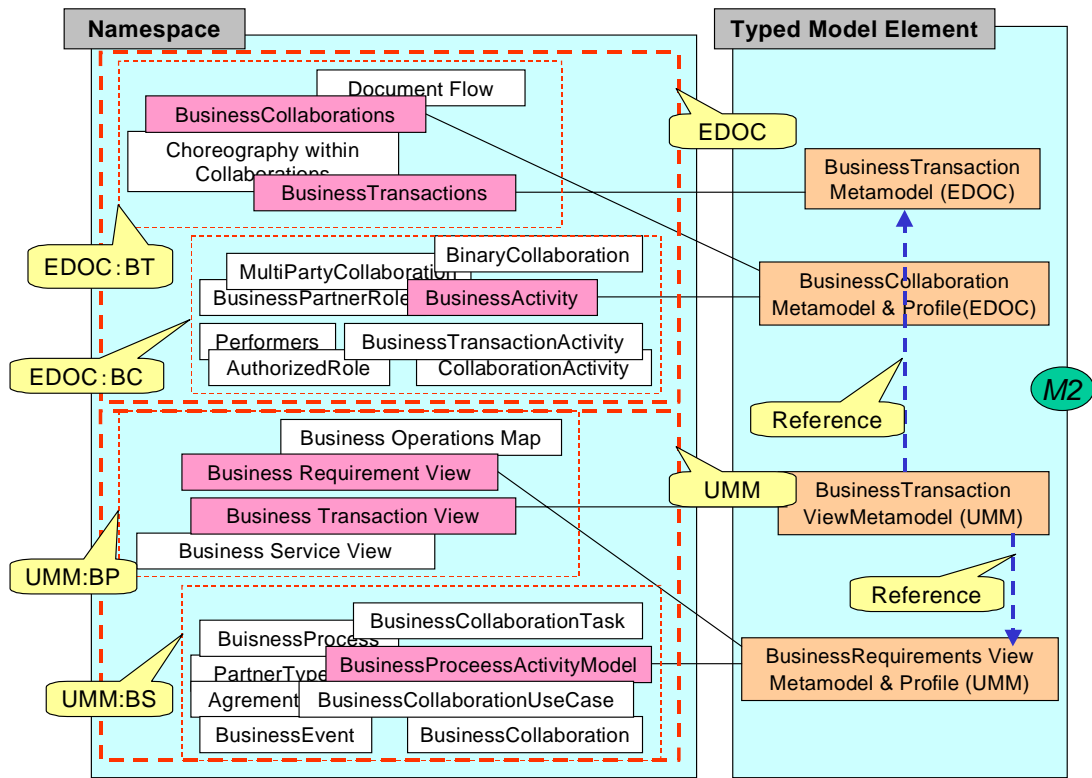


Figure 11.

Namespace	Name	Categories	Viewpoints
EDOC			
EDOC:BT	Business Collaborations		
	Business Transactions		
EDOC:BC	Business Activity		
UMM:BP	Business Requirement View		
	Business Transaction View		
UMM:BS	Business Process Activity Model		

Table xx. Examples of Category and Viewpoint

Concept/Attribute	Category/Viewpoint	
Industry	Area Category	
Business Process	Process Category	
Product	Product/Service Category	
Physical Geography/Conditions/Region	Area Category	
Geo-Political Legislative/Regulatory/Cultural	Area Category	
Application Processing	Process Category	
Business Purpose/Domain	Viewpoint/Area Category	
Partner Role	Role Category	
Service Level (profiles-not preferences)	Artifact/Service Category	
Conformance Level	Standard Category	
Virtual Marketplace	Area Category	
Info Structural Context	Area/Artifact Category	
Contracts/Agreements	Artifact Category	
Scope	Viewpoint	
Boundary of the Business Area	Viewpoint	
Reference	Reference Category	
Constraints	Specification Category	
Stakeholders		
Objective		
Business Opportunity	Viewpoint	
Description	Specification Category	
Business Operational View (BOV)	Viewpoint	
Functional Service View (FSV)	Viewpoint	
Business Knowledge	Specification Category	
Core Component	Model Element Category	
Core Process	Model Element Category	
Business Modeling Artifact	Artifact Category	
Requirements Artifacts	Artifact Category	
Analysis Artifacts	Artifact Category	
Design Artifacts	Artifact Category	
Business Modeling	Specification Category	
Requirements	Specification Category	
Analysis	Specification Category	
Design	Specification Category	
Implementation	Specification Category	
Pattern	Model Element Category	
Framework	Model Element Category	
RM-ODP Viewpoint	Viewpoint	
Platform	IT Category	
Subsystem	IT Category	
System	IT Category	
Protocol	Viewpoint/Model Element Category	
Scenario	Viewpoint/Model Element Category	

8. Reference

- [IS11179] ISO/IEC FCD 11179-3 Information Technology-Data Management and Interchange-Metadata Registry(MdR)- Part 3: Registry Metamodel (MdR3)
- [TR15452] ISO/IEC TR 15452:2000, Information technology – Specification of data value domains
- [ISO 9735 (TC 154)]
- [MOF] Meta Object Facility (MOF) Specification, Needham: OMG, 2000, formal/00-04-03
- [CWM] Common Warehouse Metamodel (CWM) Specification: OMG, 2000, ad/2000-01-01, ad/2000-01-02, ad/2000-01-03, ad/2000-01-11
- [MDA] Policies and Procedures for MDA: OMG, 2001, pp/2001-09-01
- [EDOC] UML Profile for EDOC submission: OMG, 2001, ad/2001-06-09
- [EAI] UML Profile for EAI submission: OMG, 2001, ad/2001-08-02
- [bpWS] ebXML Business Process Analysis Worksheets and Guidelines. V1.0, May 11 2001. ebXML Business Process Project Team.
- [ebBPBIAO] ebXML Business Process and Business Information Analysis Overview . Version 1.0 May 11 2001. ebXML Business Process Project Team.
- [ebBPSS] ebXML Business Process Specification Schema. Version 1.0 May 11 2001. Context/*Meta Model* Group of the CC/BP Joint Delivery Team.
- [ebPROC] ebXML Catalog of Common Business Processes. Version 1.0, May 11 2001. ebXML CC/BP Analysis Team.
- [bpPATT] ebXML Business Process and Simple Negotiation Patterns. Version 1.0, May 11 2001. ebXML Business Process Project Team.
- [ISO14662] Information Technologies - Open-EDI Reference Model. ISO/IEC 14662:1997(E). International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC).

- [ebCCD&A] ebXML Methodology for the Discovery and Analysis of Core Components. V1.0, May 11 2001. ebXML Core Components Project Team.
- [ebCNTXT] The role of context in the re-usability of Core Components and Business Processes. Version 1.0, May 11 2001. ebXML Core Components Project Team.
- [ebGLOSS] ebXML. TA Glossary. Version 1.0, May 11 2001 . Technical Architecture Project Team.
- [ebTA] ebXML Technical Architecture Specification. Version 1.0.4. 16 February 2001. ebXML Technical Architecture Project Team.
- [ebCCDOC] ebXML specification for the application of XML based assembly and context rules. Version 1.0, 11 May 2001. ebXML Core Components.
- [ebCNTXT] *ebXML Concept - Context and Re-Usability of Core Components*. Version 1.01. February 16, 2001. ebXML Core Components Project Team.
- [ebRIM] *ebXML Registry Information Model*. Version 0.56. Working Draft. 2/28/2001. ebXML Registry Project Team.
- [ebRS] *ebXML Registry Services*. Version 0.85. Working Draft. 2/28/2001. ebXML Registry Project Team.
- [ebTA] *ebXML Technical Architecture Specification*. Version 1.0. 4 January 2001. ebXML Technical Architecture Project Team.
- [bpOVER] *Business Process and Business Information Analysis Overview*. Version 1.0. Date 11 May 2001. ebXML Business Process Project Team
- [bpPROC] *ebXML Catalog of Common Business Processes*. Version 1.0. Date May 11, 2001. ebXML Business Process Project Team
- [UMM] UN/CEFACT Modeling Methodology. CEFACT/TMWG/N090R9. February 2001. UN/CEFACT Technical Modeling Working Group.