MOF 2.0 Facility and Object Lifecycle Specification (MOFFOL), Beta 1

OMG Adopted Specification

OMG Document Number: ptc/2008-02-20

Standard document URL: http://www.omg.org/spec/MOFFOL/1.0/PDF

Associated Schema File(s)*: http://www.omg.org/spec/MOFFOL/20070901

- original file: ad/07-09-01

This OMG document replaces the submission document (ad/2007-06-05, Alpha). It is an OMG Adopted Beta Specification and is currently in the finalization phase. Comments on the content of this document are welcome, and should be directed to issues@omg.org by March 21, 2008.

You may view the pending issues for this specification from the OMG revision issues web page *http://www.omg.org/issues/*.

The FTF Recommendation and Report for this specification will be published on October 3, 2008. If you are reading this after that date, please download the available specification from the OMG Specifications Catalog.

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

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

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

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

GENERAL USE RESTRICTIONS

DISCLAIMER OF WARRANTY

RESTRICTED RIGHTS LEGEND

TRADEMARKS

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

Table of Contents

1. Scope 6

2. Conformance 6

3. References 7

3.1 Normative References 7

3.2 Informative References 7

4. Terms and Definitions 7

5. Additional Information 8

5.1 Migration from MOF 1.4 8

5.2 Acknowledgements 9

6. Specification 10

6.1 Structure 10

6.2 Facilities 10

6.3 Object Lifecycle 17

6.4 Connection and Location 17

6.5 Federation 22

6.6 Authentication 22

6.7 Exception Handling 23

6.8 Constraint Handling 24

6.9 Transactions 25

6.10 Import/Export 26

7. Changes or Extensions Required to Adopted OMG Specifications 29

1. Scope

NOTE to author: Please insert Scope text here.

2. Conformance

With respect to compliance, the classes defined in this specification are to be treated as interfaces: compliant implementations must implement the properties and operations mapped to an appropriate language binding using one of the MOF2 language binding specifications (e.g., IDL) Binding). This specification is not intended to imply any specific implementation of these interfaces.

Where full compliance is not to be provided, implementations must throw a NotSupported exception.

It is possible for a facility to indicate its level of compliance using the properties defined on Facility (inherited from Workspace).

The following Compliance Points are defined. Exactly one of the Base levcels must be supported, and any number of the other levels.

BaseFacility: this includes everything in this specification except those listed as specific compliance points below.

BaseReadOnlyFacility: provided in order to allow MOF interfaces to be used as a wrapper to other systems. As for BaseFacility except does not require support of any update operations (including those specified in other MOF2 specifications)

Stores: Support for the Store class. If this is not supported, implementations will need to implement their own storage scheme.

Transactions: Support for LocalTransaction interface, and implicit transactional semantics on each request

Versioning: the interfaces obtained by a package merge of the MOF 2 Versioning specification model with that defined here. This will add capability to the Session and Workspace classes.

3. References

3.1 Normative References

MOF 2 Core

MOF 2 Versioning and Development Lifecycle

XMI 2.1

3.2 Informative References

JMI – Java Metadata Interface (JSR 40)

MOF 1.4

JCA – J2EE Connector Architecture Specification (JSR 112) version 1.5

4. Terms and Definitions

Client: A MOF client is one that makes use of MOF services. The term is not meant to imply any sort of physical separation or use of middleware (though these are of course not excluded).

Facility: A MOF Facility consists of MOF software capable of servicing client requests together with specific MOF data that is accessed or manipulated. Typically the MOF data will comprise one or more extents.

Connection: As used in this specification 'connection' may refer to a 'session connection' that persists over many client requests (effectively establishing a 'session') or to a 'transient connection' that is established for one client request only (e.g., as for a web services request). Session connections represent typical use of MOF although transient connections will become more important, especially for bulk interchange

Outermost Instance: An object that is not owned by another object.

Resource Adapter (generalized from JCA): To achieve a standard system-level pluggability between application servers and Enterprise Information Systems (EIS), the J2EE Connector architecture defines a

standard set of system-level contracts between an application server and EIS. The resource adapter implements the EIS-side of these system-level contracts.

A resource adapter is a system-level software driver used by an application server or an application client to connect to an EIS. By plugging into an application server, the resource adapter collaborates with the server to provide the underlying mechanisms, the transactions, security, and connection pooling mechanisms. A resource adapter is used within the address space of the application server.

5. Additional Information

5.1 Migration from MOF 1.4

The following outlines the equivalent to MOF 1.4 concepts. Note that MOF 1.4 had no notion of Facility, so many of the aspects in this specification are new to the MOF standard (though something equivalent was provided of necessity by most implementations):

Concept	MOF 2 Equivalent
Extent	Repository
XPackage Interfaces	Creation moved to Factory interface(s)
XClass interfaces	Creation and finding moved to Factory and Package interfaces respectively
Package clustering (clustered import)	Not explicitly supported, however an Extent is package independent and a Factory may be associated with any package – which could be created by importing or merging other packages to give the same effect as clustering
Package inheritance	None: in UML2/MOF2 Packages are not generalizable
Package nesting	No direct equivalent: though packages can be nested in metamodels this is not used only for naming and not to determine nesting of Extents which is not supported.
Package Import	Package or Element Import

5.2 Acknowledgements

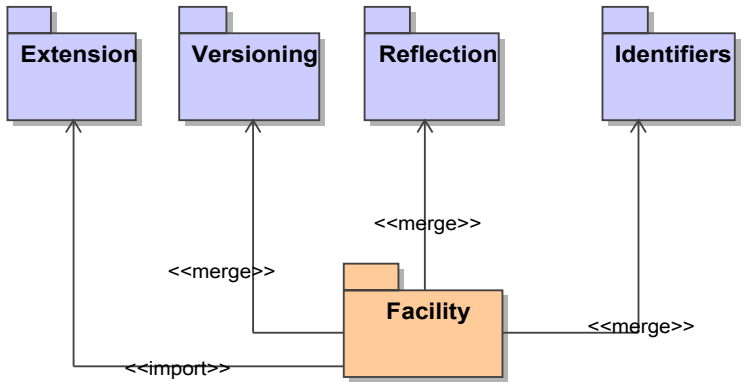
This specification was submitted and/or supported by the following companies/individuals:

- Adaptive
- Compuware Corporation
- Constantine Plotnikov
- INRIA
- Interactive Objects
- Sun Microsystems

6. Specification

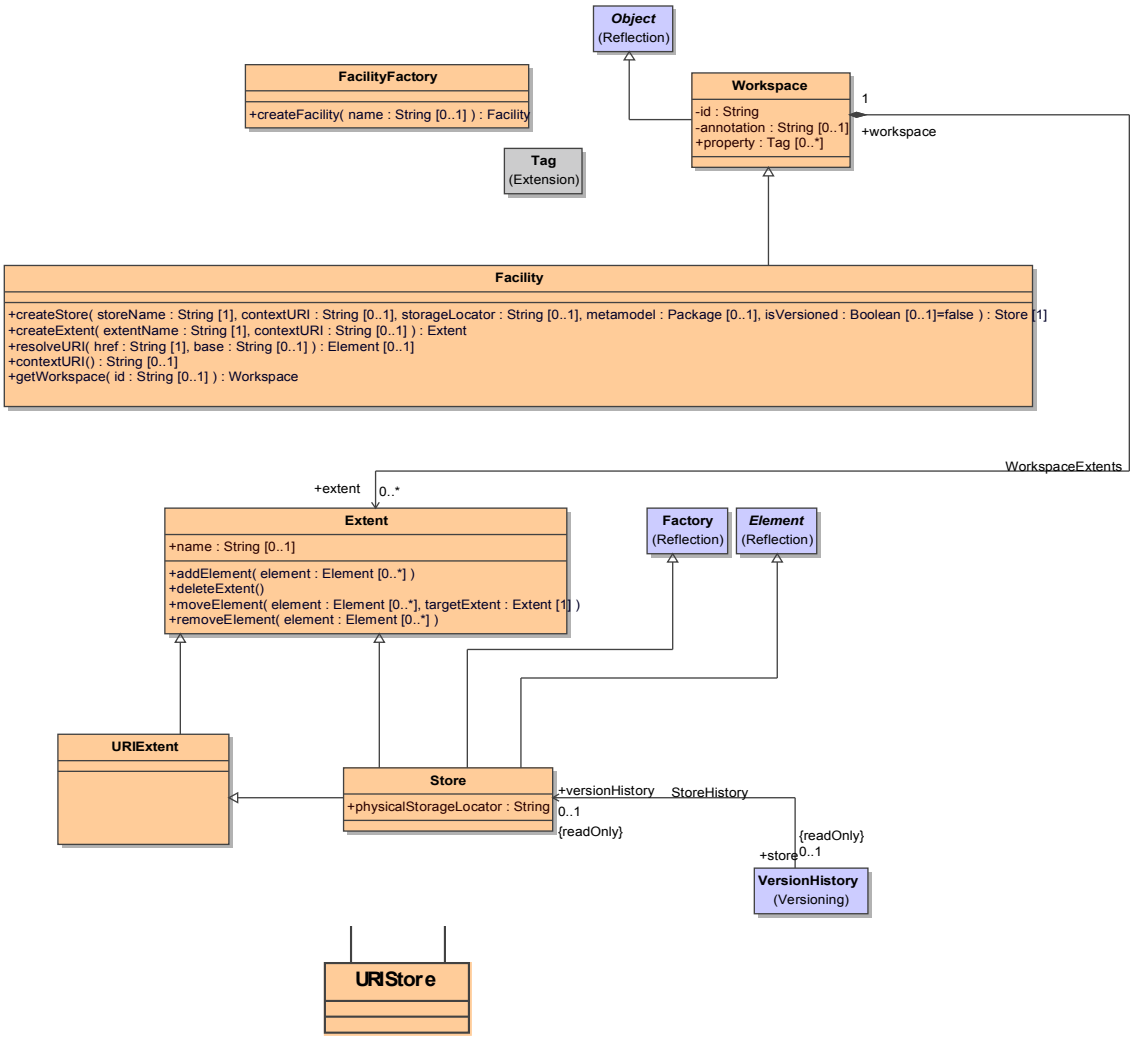
6.1 Structure

The following diagram illustrates the dependencies of facility package on other MOF packages. Note, however, that not all compliance levels require Versioning.



6.2 Facilities

This chapter describes elements that extend MOF to provide entry points to the instances of metamodels and a notion of a store; as a special kind of extent capable of physically storing the metadata. The platform independent model of the MOF Facility is as follows.



6.2.1 Facility

The Facility class provides the connection point for clients. Through inheritance from Workspace it contains Extents, some of which may be Stores and it acts as a factory for these.

6.2.1.1 Generalizations

Workspace

6.2.1.2 Properties

None

6.2.1.3 Operations

createExtent(name: String,	The URI to be used to address the Extent. If specified, an instance of URIExtent will be created.
contextURI: String[0..1]	The URI to be used to address the Extent. If specified, an instance of URIExtent will be created.
) : Extent	Creates a new Extent (or URIExtent) and adds it to the *extent* property for the facility.
createStore(name: String,	The name to be given to the new Store. It must not previously have been used by any Extent in *extent* for this facility.
contextURI: String[0..1]	The URI to be used to address this Store. If specified, an instance of URIStore will be created.
storageLocator: String[0..1]	Will be used to set the physicalStorageLocator property (q.v) of the new Store.
metamodel:MOF::Package[0..1]	Used to set the inherited *Factory::package* property on the created store.
isVersioned:Boolean[0..1]=false	Used to indicate whether the store to be created is to be enabled for versioning.
) : Store	As for createExtent but creates an instance of Store. If isVersioned is true, then a VersionHistory and initial Version is also created and associated with the Store (see MOF Versioning for details of these classes).
resolveURI(href: String[1]	The URI to be dereferenced.
base: String[0..1]	The base URI for the facility, to be stripped from the supplied href in order to get a unique id within this facility.
) : Element[0..1]	Dereferences the supplied URI down to an element (if possible) which is returned.

contextURI() : String[0..1]	Returns the URI for the facility, used to connect.
getWorkspace(id: String[0..1]	An optional identifier to be matched.
) : Workspace[0..*]	Returns Workspaces contained by the Facility. If id parameter is supplied, then only the workspace (if any) with that id is returned; otherwise all Workspaces are returned.
createWorkspace(id: String[0..1]	The identifier for the new Workspace. It must be unique for the Facility.
) : Workspace	Creates a new Workspace that will be added to the set returned by getWorkspace for this facility.

6.2.1.4 Constraints

No additional constraints.

6.2.1.5 Semantics

A Facility represents the means for clients to 'connect' to models and elements, which are accessed via Extents.

6.2.1.6 Rationale

There needs to be a top-most object serving as a locator.

6.2.1.7 Changes from MOF 1.4

There was no notion of anything above RefPackage – a package extent. In MOF 1.4 terms, Facility can be thought of as a container of zero to many package Extents. MOF 1.4 did not include a platform independent model of facilities and reflection. It was left up to the specific mappings to specify those if desired.

6.2.2 Extent

This adds incremental operations, through package merge, to the Extent class, originally defined in MOF 2 Core.

6.2.2.1Generalizations

None

6.2.2.2 Properties

None

6.2.2.3 Operations

addElement(addedElement: Element)

This adds an existing Element to the extent. This is a null operation if the Element is already in the Extent.

removeElement (removedElement: Element)

This removes the Element from the extent. It is an error if the Element is not a member. This may or may not result in it being deleted (see 2.3.2 Deletion Semantics).

moveElement (movedElement: Element, targetExtent: Extent)

This is an atomic combination of addElement and removeElement. targetExtent must be different from the extent on which the operation is invoked.

deleteExtent()

This deletes the Extent, but not necessarily the Elements it contains (see 2.3.2 Deletion Semantics).

6.2.2.4 Constraints

No additional constraints

6.2.2.5 Semantics

See MOF 2 Core, and 2.2.3 below.

6.2.2.6 Changes from MOF 1.4

At MOF 1.4, extent membership was quite static: an Element was created within one extent and could not be moved. This is more akin to the notion of Store at MOF 2: Extent is more of a loose grouping of Elements providing a scope/content for various operations. An Element may be in more than one Extent and may be moved between them.

6.2.3 Store

Store can be understood as a special kind of extent with an ability to actually store the models. The lifecycle of an Element can be managed by at most one Store.

In the case where the storage is not being managed by MOF then Stores are not required to be used. It is still possible to create Elements provided that appropriate implementations of Factory are available.

6.2.3.1 Generalizations

Factory, Element, URIExtent.

6.2.3.2 Properties

PhysicalStorageLocator : String

[1]	Specifies a storage location to be used for the physical storage. The usage of the value is implementation dependent and could, for example, represent the name of a file or a database.

6.2.3.3 Operations

No additional operations.

6.2.3.4 Constraints

A Store is able to manage only Elements described by metaElements contained in the metamodel referenced by the inherited *package* property of the store.

6.2.3.5 Semantics

A Store manages the lifecycle of the elements associated with it – it provides a physical home for them. Store inherits from Factory, which means that it can create the elements. All the elements created by the store automatically become associated with it as an Extent.

6.2.3.6 Changes from MOF 1.4

The Store concept was not present in MOF 1.4, although it is similar to an outermost package extent/proxy from MOF 1.4. The difference is that rather than consisting of individual class and association proxies, Store is able to handle creation of instances of any class (or Association in CMOF) itself.

6.2.4 Workspace

This is a simplification of the concept in MOF Versioning, where it provides access to specific Versions of extents. When used with Versioning the classes will be merged. In the context of this Facility specification, Workspace (which is inherited by Facility itself) provides a scoping mechanism in terms of the Extents that are available.

6.2.4.1 Generalizations

Object

6.2.4.2 Properties

id : String	The name or identifier for the Workspace.
annotation : String[0..1]	Optional descriptive text.
properties: Tag [0..*]	Provides a placeholder for implementation-specific details.
extent: Extent [0..*]	The set of Extents that the workspace provides access to.

6.2.4.3 Operations

No additional operations

6.2.4.4 Constraints

None

6.2.4.5 Semantics

A Workspace is the container for Extents.

6.2.4.6 Changes from MOF 1.4

The Workspace concept was not present in MOF 1.4.

6.2.5 FacilityFactory

This is used to create Facilities.

6.2.5.1 Generalizations

None

6.2.5.2 Properties

None

6.2.5.3 Operations

createFacility(id:String): Facility	The id to be given to the new Facility

This creates a new Facility.

6.2.5.4 Constraints

None

6.2.5.5 Semantics

This is a pure Factory interface which can have multiple implementations.

6.2.5.6 Changes from MOF 1.4

The Facility concept was not present in MOF 1.4.

6.3 Object Lifecycle

6.3.1 Lifecycle Operations

This section summarizes how the different lifecycle operations are provided by the MOF interfaces.

Creation: An element is created by a Factory. Typically this will be a Store but not necessarily.

Deletion: Each element has a delete interface; however see 6.3.2 below.

Copy: An element may be added to another Extent: this may mean it is accessible via more than one URI. However there is only one underlying object so changes made via one extent will be reflected in the other.

Move: An element may be added to another extent and then removed from the original as an atomic operation

Compare: MOF Core provides a general comparison by identity. No further comparison capability based on element content is provided - since the scope (e.g. whether deep or shallow) will tend to be application-specific.

6.3.2 Deletion Semantics

If an Element is removed from an Extent that is a Store then it is deleted. Removing an object from any other Extent has no deletion implications (though custom subclasses of Extent may implement different policies).

The explicit delete operation will remove the element from its Store (if it is in one) and any other Extents.

6.4 Connection and Location

6.4.1 Location

The only location capability provided is via URI. URIs may be used to locate:

- Facilities
- Extents

- Elements from a metamodel
- Instances

The Facility interface provides a single point for resolving URIs, using the resolveURI operation. This may in turn delegated to URIExtents or Stores but this is hidden from the caller.

6.4.1.1 Basic URI Encoding Scheme

In order to provide a base level of interoperability, this specification includes a basic scheme for mapping URIs to a metamodel structure. There is nothing to stop implementations using any internal/proprietary scheme for this mapping.

The scheme covers:

- How to generate a URI for an element
- How to locate an element from a URI

It only applies to Elements that have non-empty values for their isID property and which have composite parents likewise. It makes the following assumptions about models:

- The values of isID properties are unique within the same composite parent
- The values of isID properties are syntactically valid components of a URI

Note that it is possible for isID properties to be derived properties: the derivation may be arbitrary manner. However it is the responsibility of the Facility to provide efficient location of elements via URI (which for derived properties would tend to imply using some sort of cached lookup table or directory).

The basic Scheme encodes absolute URIs as follows (see also RFC 2396):

http://*Facility-ContextURI***/***Facility-Name***/***Extent-Name***[?***Top-Element-Id***{/***Element-Id***}*|;**uuid=***uuid]*

Note that ? is used in the middle of the hierarchy to separate the extent from the contents.

The *Facility-ContextURI* is the value returned by the operation Facility::ContextURI() and should be mappable to an IP address using normal TCP-IP mechanisms such as DNS, or local mechanisms such as XML Catalog. It may include a port number. The IP address is used for connection (see section 2.4.2).

Facility -Name is the property Facility::name defined against the Facility class

Extent-Name is the property Extent::name defined against the Extent class

Using this scheme the value returned by URIExtent::contextURI() is thus **http://***Facility-ContextURI***/***Facility-Name***/***Extent-Name*

Top-Element-Id is the identifier (value of the property with Property:isID=true) for a TopElement. A TopElement is a Reflective::Element in the Extent with no composite owner (all properties with opposites having isComposite=true are empty for this Element). There can be many such TopElements for an Extent.

Each *Element-Name* is the value of the property with Property:isID=true for an element whose compositeOwner is the element identified by the URI to the left.

Here are some examples:

a) the metaclass UseCase in the UML2 metamodel:
http://mof.omg.org/MetamodelFacility/UML2Store?uml2/UseCase

b) the activity CalculateHoursWorked.
http://mof.adaptive.com:8083/ModelsFacility/PayrollModels?ProcessModel/MonthlyProcess/CalculateHoursWorked

c) the element with uuid X12345 in extent PayrollModels
http://mof.adaptive.com:8083/ModelsFacility/PayrollModels;uuid= X12345

Relative URIs

In the Basic Scheme URIs are always relative to an Extent and not to composite parents. So, for example, xmi:idref=”Customer.order” would be invalid for referencing Customer::order: a TopElement would be needed (i.e., the Package): for example Customer.Customer.order.

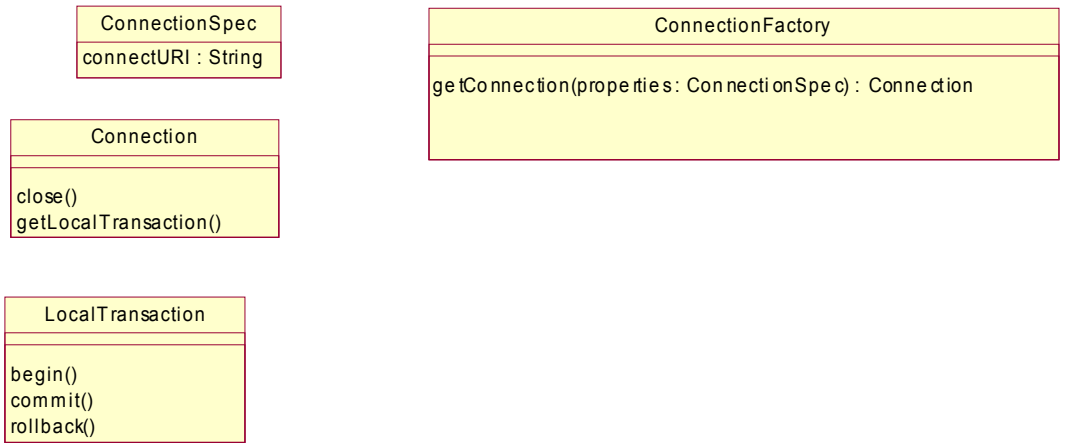
For the purpose of cross-referencing within an XMI file then the XMI file itself is treated as an Extent. The MOF hierarchy not the XML hierarchy should be used. Normally they are the same, but for example the *XMI* XML element is ignored for addressing purposes and not counted as a TopElement. And Tags (MOF) and Stereotype instances (UML), which are not nested in the XML structure (due to the lack of a composite ownedAttribute), are nested in the MOF hierarchy.

Relative addressing using . and .. may be used within URI up to the ? (if present).

6.4.2 Session

Connections can be established implicitly per client request or as a session – represented via a Session object (which is transient). In both cases a ConnectionSpec is used to hold the connection information: this is an extensible structure.

The metamodel for Sessions is as follows. This is based on the interfaces provided by the J2EE Connection Architecture (JCA) in order to provide interoperability with clients and application servers using that standard (assuming a Java binding).



Once the Session is established, it should be passed to subsequent requests: the mechanism for this is platform dependent.

The following capabilities are only available on explicit Session:

Transaction interface (otherwise transactions are implicit per request)

6.4.2.1 SessionFactory

This is used to create Sessions.

6.4.2.1.1 Generalizations

None

6.4.2.1.2 Properties

None

6.4.2.1.3 Operations

createSession(properties:ConnectionSpec	The parameters for the session. By default this is just the connection URI, but the ConnectionSpec is designed to be extensible.
): Session	

This creates a new Session connected to the specified Facility.

6.4.2.1.4 Constraints

None

6.4.2.1.5 Semantics

This is a pure Factory interface that can have multiple implementations.

6.4.2.1.6 Changes from MOF 1.4

The Session concept was not present in MOF 1.4.

6.4.2.2 Session

This represents a client session with a facility; it is not a persistent class. When used with Versioning this is designed to merge with the Session class in that specification.

6.4.2.2.1 Generalizations

None

6.4.2.2.2 Properties

workspace: Workspace[0..1].	This is the workspace that scopes the extents available in this session. It may be updated within the session.

6.4.2.2.3 Operations

Close()	This closes the connection and deletes the Session.
getLocalTransation(): LocalTransaction	This creates a new transaction proxy, but does not begin it yet. See section 6.10.

6.4.2.2.4 Constraints

None

6.4.2.2.5 Semantics

This represents a single connection. Depending on the facility implementation, a client may maintain multiple concurrent Sessions (if not supported, attempts to create a further Session will throw an exception).

6.4.2.2.6 Changes from MOF 1.4

The Session concept was not present in MOF 1.4.

6.4.2.3 ConnectionSpec

This is a data type designed to be extended for more sophistication in connection.

6.4.2.3.1 Properties

connectURI: String	The URI to use for connecting to a facility. A Connection is to a Facility so the value should match Facility::ContextURI()/Facility::name (i.e., the values of these two properties separated by a '/').

6.5 Federation

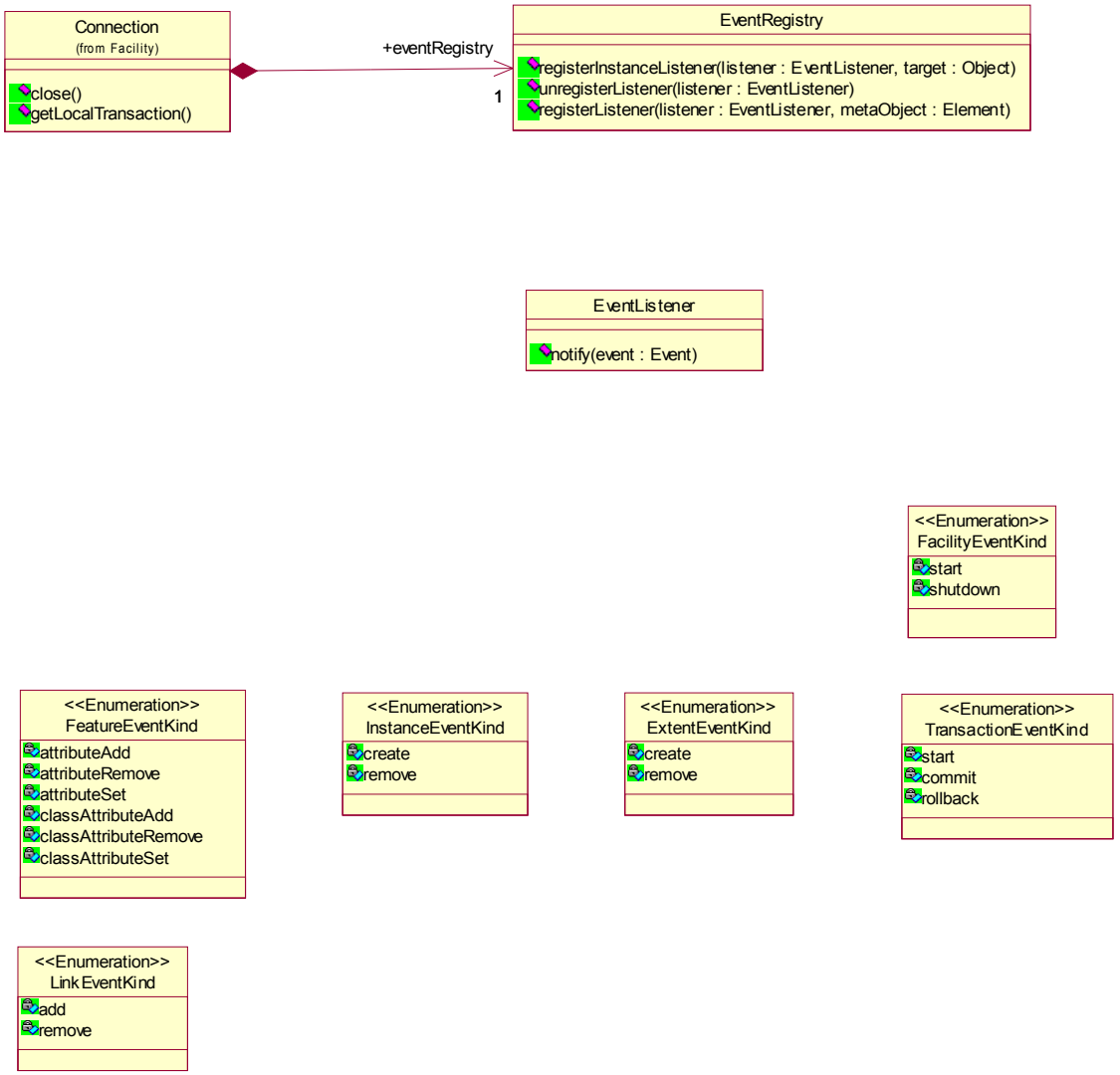
At the level of client APIs such as Reflection it is possible to seamlessly traverse associations that span multiple physical stores or even facilities. This is handled by Facilities behind the scenes.

6.5.1 Local Copies of Elements

The ability for elements to appear in many extents allows for local copies. This can be combined with a (URI) resolution mechanism such as XML Catalogs for mapping URIs to a local XMI namespace and hence Extent.

6.6 Authentication

ConnectionSpec is designed to be extended to contain credentials which can be authenticated by ConnectionFactory implementations. This specification does not mandate a specific scheme.



6.7 Exception Handling

6.7.1 Exception Definition

In UML Infrastructure (and hence MOF2 Core), exceptions are represented as Types associated with an Operation via the multivalued *raisedException* association. The types have names and possibly structure (e.g., holding details of an error) and may use inheritance. However the Types for the exceptions in MOF Core itself have not been modeled (they are referred to only by name in the text for different Operations). Furthermore UML2 has no **notation** defined for actually modeling Types associated with Operations as raisedExceptions.

Therefore it is proposed to:

- Determine in conjunction with UML2 RTF the notational means to associate exceptions with Operations.
- Explicitly model all the Exceptions for all the MOF2 specs, including Facility, to form an integrated consistent set with appropriate reuse, drawing on the exception parameters from MOF 1.4 and the MOF2 IDL binding, and propose to MOF2 RTFs/FTFs.

6.7.2 Exception Effect

When using transactions (implicitly or explicitly), the effect of an exception occurring is to abort the current transaction and roll back any changes. Without transaction support then the system should be considered in an undefined state.

In order to permit the re-running of such failed operations, where possible operations should be idempotent (that is they can be repeated). So, for example, adding an element to a Set when it is already present should not cause an exception but have a null effect. This is not always achievable (e.g., when adding an element to a bag or list) but provides a good principle for exception design.

6.7.3 Exception Returns

Language-specific mechanisms (for example Java exceptions) are used to return exception information (an instance of the Type associated with the Operation) to the client program.

6.8 Constraint Handling

MOF 1.4 distinguished between *immediate* (executed immediately after any change) and *deferred* constraints. (executed on demand only). There were also implicit constraints e.g. those implied by the multiplicity of Properties.

MOF 2.0 should reflect the fact that constraint checkers (e.g., OCL engines) are now usually separate components.

Clearly one reason why a change might be refused is through constraint violation, and it should be possible to configure constraint checking through software acting in the role of an agent (e.g., an OCL engine).

6.8.1 Information

Constraint failure raises a specific exception (see 6.8) which has properties to indicate the constraint failed and the failing element(s). Optionally many constraint exceptions may be grouped in an overall Constraint Exception: however it is permissible to return only the ‘first.’

6.9 Transactions

LocalTransaction objects can be created via the Session interface – see section 6.4.2. This is again based on JCA which has been designed to be able to interface with many different back-end systems.

6.9.1 LocalTransaction

This is a proxy for a local transactions ‘thread’. The same LocalTransaction can be used to begin and end (commit or rollback) many actual transactions. Nesting is not supported.

6.9.1.1 Generalizations

None

6.9.1.2 Properties

None

6.9.1.3 Operations

begin()	This starts a new transaction. An exception is thrown if there is a transaction in progress.
commit()	This commits the current transaction and ends it. Any changes since the transaction begin() are persistently recorded. An exception is thrown if there is no transaction in progress.
rollback()	This aborts the current transaction and ends it. Any changes since the transaction begin() are lost. An exception is thrown if there is no transaction in progress.

6.9.1.4 Constraints

None

6.9.1.5 Semantics

ACID semantics must be implemented.

6.10 Import/Export

6.10.1 XMI Cross-Referencing

In order to allow seamless XMI processing, access via hrefs in XMI files must yield the same XMI stream regardless of whether the content is stored in a local file or comes from a remote MOF facility. XML Catalog may be used to redirect the request to local storage – for example by mapping part of the URI to a local XMI file.

Note that this gives the choice of using URIs to indicate specific elements or using a URI to access an Extent and then using XPath to navigate the XML stream for the whole extent.

6.10.2 Import/export via HTTP

Due to the use of URIs to access XMI streams then one export option is to use http (or https) *get* on a URI. This may be limited and is not for use in secure environments, unless a Single Sign On mechanism is in place since it does not provide for explicit Connection.

Likewise an http *put* can be used for imports.

This style of interface is commonly referred to as REST (Representational State transfer) and is widely used in web applications.

6.10.3 Import/Export Operations

An import/export is inherently transactional (it completely succeeds or fails). However it may be part of a larger transaction set up on the Session.

These operations are provided on Session since it has the capability to select a Workspace for the operation to act on. In general the behavior is as described in the XMI specification.

importURI (uri: String[0..1],	The target Extent for the import. It must be visible via the Session. If empty the Workspace of the Session will be used
Stream: String,	The information to be imported as an XMI stream.
baseURI: String[0..1],	The base URI for the facility, to be stripped from contained hrefs in order to get a unique id within this facility.
options: ImportOptions): Element[0..*]	The import options (see description of the data type).

exportURI (uri: String,	The target Extent for the import. It must be visible via the Session. If empty the Workspace of the Session will be used.
baseURI: String[0..1],	The base URI for the facility, to be added to the contained hrefs in order to get a unique id within this facility.
options: ExportOptions): String	The export options (see description of the data type) The value returned is the exported XMI stream.

exportElements(elementSet:Example[0..*],	The set of Elements to export.
baseURI: String[0..1],	The base URI for the facility, to be added to the contained hrefs in order to get a unique id within this facility.
options: ExportOptions	The export options (see description of the data type)
): String	The value returned is the exported XMI stream.

6.10.3.1 ImportOptions

This is a DataType to control import behavior through a number of Boolean flags:

6.10.3.1.1 Properties

clearFirst: Boolean[0..1] = false	Before starting, the import will clear the target extent (i.e., delete all elements).
tidyOrphans: Boolean[0..1] = false	After completing the import this will delete all 'orphans' i.e., elements with no composite owner except for those elements with no composite owner in the imported file.
retainToolExtensions: Boolean[0..1] = false	Persistently retains the information from all XMI Extension elements after the import. See 7.1.2.4
matchUuids: Boolean[0..1] = false	Reconciles imported against existing elements using uuids. This is typically used for a 'round trip' where elements are exported, edited by an external tool, then re-imported. It is important that they are reconciled with the originals. See 7.1.2.2.

6.10.3.2 ExportOptions

This is a DataType to control export behavior through a number of properties:

6.10.3.2.1 Properties

XmiVersion: String[0..1] = 2.1	The version of XMI to use. There is no obligation to support any version except the default, 2.1.
clearAfter: Boolean[0..1] = false	fter successful completion, the export will clear the source extent (i.e., delete all elements). This is the equivalent of a 'move.'
includeUuids: Boolean[0..1] = false	Includes uuids in the export. This is the 'partner' to includeUuids option on import: see that for details.

retainToolExtensions: Boolean[0..1] = false	The exported file includes all information from previously imported and ratained XMI Extension elements. See 7.1.2.4.
toolExtensionToRestore: String[0..1]	Includes Extension information for the specified Tool identifier in the exported file. Furthermore it uses that Tool's identifiers as the xmi:uuids in the exported file.
nsURI: String [0..1]	The namespace to use in the exported file (to override what is in the source metamodel). May be used for example to export a L1 stream from a L3 model repository.

7. Changes or Extensions Required to Adopted OMG Specifications

7.1.1 MOF 2.0 Core formal/06-01-01

A further rule is necessary for the effective isID property: add the following to the Constraints section of 10.3 Property

- Only one member Attribute of a Class may have isID = true. Any others (e.g., those inherited) must be redefined: either made unavailable or redefined to change isID=false

7.1.2 MOF 2 XMI formal/05-09-01

7.1.2.1 Example of Basic URI Scheme

The Basic URI Scheme (see section 2.4.1.1) should be referenced and used as an example in the XMI specification.

There should be a new example 5 in section 4.10.2.2

5 Using the MOF 2 Facility Basic Encoding Scheme

The MOF2 Facility specification provides a means of encoding URIs to refer to elements in facilities: these may be realized through XMI files, database-backed repositories or other mechanisms. Hence it is usually not appropriate to make use of xmi:id values which are in general transient and limited in scope: rather names and unique ids are made use of. Full details are contained in that specification.

As an example here is a link to an activity called CalculateHoursWorked which is within ProcessModel within PayrollModels; PayrollModels is located via facility http://mof.adaptive.com:8083/ModelsFacility.

<activity
href="http://mof.adaptive.com:8083/ModelsFacility/PayrollModels?ProcessModel/MonthlyProcess/CalculateHoursWorked"/>

7.1.2.2 Reconciliation on Import

Add the following as a new section 4.15 in the XMI specification:

4.15 Import Reconciliation

The following are cases where an element in an imported XMI file will resolve to an existing element in the importer: only one of the following need apply

- both elements have uuids which are identical

- the XMI element has extenderID and extender which are identical to those associated with the element in the importer (see 2.11.7)
- both elements have identical values of a Property with isID=true
- both elements are in the same extent and would have identical values for the basic URI scheme

Should elements match as above then the element in the importer is updated as follows:

- If property P is explicitly included in the XMI file then the value of that property is updated to the value(s) from the XMI file. If multivalued then any existing values not in the new set are removed.
- If P is included but empty in the XMI file the property is unset; if mandatory it is instead set to its default value.
- If P is not explicitly included in the XMI file then any existing value in the importer is unchanged.

Should a matching element be referenced from the Differences element then the actions are carried out in order prior to the main import.

7.1.2.3 Clarify use of isId

Change the last bullet in section 4.11.2 as follows (this also fixes the typo where 2 occurrences of 'isProperty' should be 'idProperty'):

- If the id Property tag is set, the idName tag is ignored. The tagged property must have type which is a Datatype and it must also have isId = true in the metamodel. This means that only one idProperty tag may be specified for a class.

7.1.2.4 XMI Extensions

Add the following to the end of section 4.13 *Document Exchange With Multiple Tools* (prior to sub-heading for 4.13.1)

In order to allow the use of such schemes as outlined here, XMI Extensions must be persistently maintained by the importing tool.

7.1.3 MOF 2 Versioning formal/07-05-01

7.1.3.1 Align Workspace to Extent association

In order to allow a package merge of between Versioning:Workspace and facility::Workspace it is necessary to rename Workspace::versionedExtent to Workspace::extent in the Versioning spec.

This change should be made in the following places (in each case the word "versionedExtent" (when spelled with lower case 'v' only) is replaced with "extent"):

- Figure 5.3
- Section 5.6.2.1, first reference

7.1.4 MOF 2.0 IDL

This should be aligned to include the functionality and operations in this specification. However this will be done through the normal revision process rather than by the adoption of this submission.