Reusable Asset Specification

This OMG document replaces the submission document (ad/03-10-12) and the Draft Adopted specification (ptc/04-05-12). It is an OMG Final Adopted Specification and is currently in the finalization phase. Comments on the content of this document are welcomed, and should be directed to *issues@omg.org* by June 30, 2004.

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 29, 2004. If you are reading this after that date, please download the available specification from the OMG Specifications Catalog.

Reusable Asset Specification
Final Adopted Specification

ptc/04-06-06

Table of Contents

1 Scope

The scope of this Specification is a set of guidelines and recommendations about the structure, content, and descriptions of reusable software assets. We recognize that there are different categories of reusable software assets. The specification identifies some categories, or rather types or profiles and provides general guidelines on these profiles.

The Reusable Asset Specification (RAS) addresses the engineering elements of reuse. It attempts to reduce the friction associated with reuse transactions through consistent, standard packaging. This is much like the steering wheel, turn signals, pedals, and fuel gauge in a car: although they're slightly different across car models and makes, there's a familiarity among them that significantly reduces the costs of reuse.

2 Conformance

Editorial Comment: Needs to be completed.

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification.

Editorial Comment: Needs to be completed (or possibly eliminated).

4 Terms and Conventions

For the purposes of this specification, the following terms and conventions apply.

4.1 Document Conventions

The following conventions and terms are used in this document.

- <descriptor-group> element: a term with the < > delimiters represents an element in an XML schema.

- *attribute*: a bold-italic term is an attribute on an element.

- All node and attribute names are written in lower-case letters only.

- When multiple words are used for the name of a node or attribute, we use a hyphen between the words, as such 'artifact-type'; however, this will likely change as we bring RAS to be compliant with MOF 2 and XMI 2.

- Many of the images in this document show XML Schema and XML documents in the WebSphere Studio Application Development XML editor.

- Each UML model element is described with two sections, the UML Model for XML Schema section, with its respective XML Schema, and the UML Model for MOF 2.0 XMI section, with its respective MOF/XMI XML Schema.

UML Model for XML Schema	XML Schema
UML Model for MOF 2.0 XMI	XML Schema

4.1.1   XML Elements

XML elements are written inside of angled brackets, for example <element>.

4.2   UML Modeling Conventions

There are two RAS UML models described in this document, these models were produced using Rational Rose. One RAS UML model is used to translate into XML Schema and represents the RAS XML schemas and files that are used by various tool vendors today. The other RAS UML model is used to translate into MOF/XMI XML Schema. This model was organized to be translated by the Eclipse EMF converter.

The modeling conventions used in the models are described below starting with the RAS UML model for XML Schema.

4.2.1   RAS UML Model Conventions for XML Schema (the incumbent)

- **Class names**
  The class names begin with lower case and multiple words are separated with a hyphen ('-').

- **Association, IDs, containership**
  All associations are declared as by-value associations. This is intended to express that the 'contained' class will be a child element in the XML schema. As such, where persistent associations need to be preserved the owning class contains an ID attribute.

- **Association cardinality**
  The cardinality amongst classes is expressed using UML adornments with the style [lower range…upper range]. In the case of an infinite upper range the '*' adornment is used.

- **Attribute names**
  Following the same convention as class names, the names begin with lower case and multiple words are separated with a hyphen ('-').

- **Attribute types**
  The attribute type is declared using non-programming language specific adornments using lower case terms such as [string, int].

- **Attribute mandatory/optional**
  The attribute's mandatory/optional information is captured in the attribute's documentation window using values of

[required, optional].

- **Attribute visibility**
The attribute's visibility is declared as private by default, although these semantics do not translate directly into the XML schema.

4.2.2   RAS UML Model Conventions for MOF 2.0 XMI

- **Class names**
The class names begin with upper case and multiple words are identified with an upper case letter.

- **Class documentation**
Each class is defined in the class' documentation field.

- **Association, IDs, containership**
All associations which are intended to be parent-child relationships are modeled as by-value associations. If a class needs to maintain a reference to another class, this is handled with a uni-directional association, no by-value semantics are specified. This allows us to remove the ID attributes from the model. However, there are some places where an ID attribute exists. These are used for IDs that go beyond the boundary of the current asset manifest file.

- **Association role names**
All association role names are declared using singular terms.

- **Association cardinality**
The cardinality amongst classes is expressed using UML adornments with the style [lower range…upper range]. In the case of an infinite upper range the '*' adornment is used.

- **Attribute names**
The attribute names begin with lower case and multiple words are identified with an upper case letter.

- **Attribute types**
The attribute type is declared using non-programming language specific adornments using some terms that begin with upper case such as [String, int] and other terms that begin with lower case.

- **Attribute mandatory/optional**
The attribute's mandatory/optional information is captured in the attribute's stereotype information. Thus <<1..1>> on the attribute's stereotype information indicates that the attribute is required and has an upper bound of 1. This modeling convention was used to support the MOF/XMI translation tools.

- **Attribute visibility**
The attribute's visibility is declared as public for all attributes.

- **Attribute documentation**
Each attribute is defined in the attribute's documentation field.

5   Symbols

List of symbols/abbreviations.

Editorial Comment: Needs to be completed (or possibly eliminated).

---

6 Additional Information

6.1 Related and Dependent Standards

This specification depends on several other specifications which are listed below.

6.1.1 XML

The manifest document is an XML document. This specification was written with the Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation 6 published in October 2000. XML is a simple, very flexible text format derived from SGML (ISO 8879). This specification is managed by the W3C.

6.1.2 XML Schema

The authoritative description of the RAS manifest document structure is provided as an XML Schema. XML Schemas express shared vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents. XML Schema was approved as a W3C Recommendation on 2 May 2001. This specification is managed by the W3C.

6.1.3 MOF/XMI XML Schema

The OMG describes a MOF / XMI mapping which describes how to handle complex associations and UML models for interchange. Using this standard approach to creating the UML models describing the domain of reusable assets and translating to the XMI-based XML schema simplifies the effort. MOF and XMI are described more at http://www.omg.org/gettingstarted/overview.htm.

6.2 References

John Cheesman, UML Components, Addison-Wesley.

6.3 Acknowledgements

The following individuals are acknowledged for their contribution to RAS:

- Brent Carlson (LogicLibrary)
- Charles Stack (Flashline)
- Craeg Strong (Ariel Partners)
- Ed Bacon (Vanguard)
- Grady Booch (IBM)
- Grant Larsen (IBM)
- Ivar Jacobsen (Jaczone)
- Jim Conallen (IBM)
- Jim Green (Microsoft)
- Jimmy Kerekes (Telstra)
- John Cheesman (Irene 7)

- John Steele (Charles Schwab)

- Jun Ginbayashi (Fujitsu)

- Lance Delano (Microsoft)

- Lior Amar (OSTnet)

- Martin LeClerc (IBM)

- Kumar Vagaparty (Merrill Lynch)

- Pete Rivett (Adaptive)

- Sam Patterson (ComponentSource)

- Sridhar Iyengar (IBM)

- Wayne Wulfert (Caterpillar)

- Wojtek Kozaczynski (Microsoft)

6.3.1 Submitters

- Adaptive (http://www.adaptive.com)

- Blueprint Technologies (http://www.blueprinttech.com)

- ComponentSource (http://www.componentsource.com)

- Flashline (http://www.flashline.com)

- IBM (http://www.ibm.com)

- LogicLibrary (http://www.logiclibrary.com)

- OSTnet (http://www.ostnet.com)

6.3.2 Supporters

- ABB

- Aetna

- Borland Software

- Cap Gemini Ernst & Young

- Caterpillar

- Fujitsu

- IconMedialab

- Iocore-7n

- Jaczone

- Kantega

- Martin Griss Associates

- OSTnet

- Praxis Engineering Technologies

- RDA Corporation

- Telstra
- Unisys
- USPTO
- Volvo
- Xansa

6.3.3   Reviewers

- Alan Brown (IBM)
- Bran Selic (IBM)
- Davyd Norris (IBM)
- Daud Santosa (USPTO)
- Jim Rumbaugh (IBM)
- Kelli Houston (IBM)
- Magnus Christerson (IBM)
- Pete Eeles (IBM)
- Steve Brodsky (IBM)

7 Reusable Assets

7.1 Defined

Simply said, reusable assets provide a solution to a problem for a given context. The figure below illustrates a high-level description of reusable assets. The asset may have a variability point, which is a location in the asset that may have a value provided or customized by the asset consumer. The asset has rules for usage which are the instructions describing how the asset should be used.
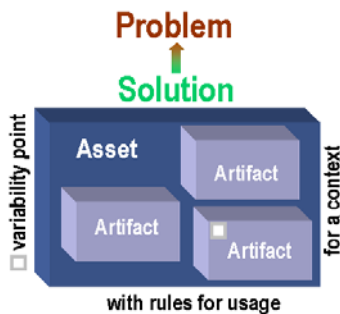
Figure 1 - General Asset Definition

Artifacts are any workproducts from the software development lifecycle, such as requirements documents, models, source code files, deployment descriptors, test cases or scripts, and so on. In general the term "artifact" is associated with a file. These terms are used interchangeably throughout this document.

7.2 Reusable Software Asset Types

The general asset definition given above is refined for various kinds of software assets. A specific kind of asset may specify the artifacts that must be in the asset and may declare a specific context, such as a development context or a runtime context for which the asset is relevant. There are three key dimensions that describe reusable assets: granularity, variability, and articulation.
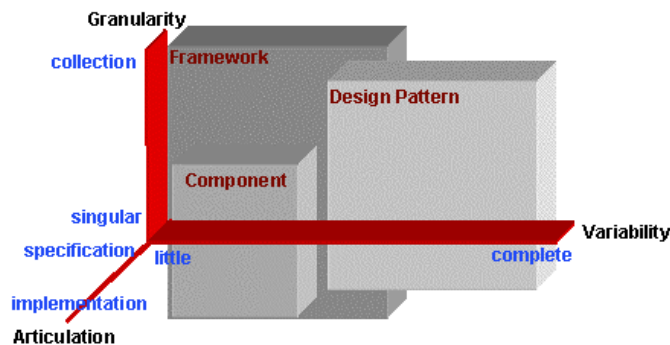
Figure 2 - Reusable Software Asset Types

7.2.1 Granularity

The granularity of an asset describes how many particular problems or solution alternatives a packaged asset addresses. The simplest assets only offer a singular solution to a single well-defined problem. As the granularity increases the asset addresses multiple problems, and/or may offer alternative solutions to those problems.

In general with the increase of granularity comes an increase in size and complexity of an asset.

7.2.2 Variability

The variability and visibility of an asset is another key property of an asset. At the one extreme an asset can be invariable, that is it cannot be altered in any significant way. This is often the case for assets that are component binaries. Assets at this end of the spectrum are sometimes called black-box assets, since their internals cannot be seen and are not modifiable.

At the other end of the spectrum are white-box assets. These assets are created with the expectation that asset consumers will edit and alter its implementation. White-box assets also typically include development artifacts such as requirements, models, build files, etc.

Two other variations in between are clear-box assets and gray-box assets. Clear-box assets expose implementation details (via models code fragments, or other documentation), however they cannot be modified. These details are exposed solely to help the consumer better understand the inner workings of the asset, so that the consumer can use the asset more efficiently. Gray-box assets expose and allow modification only to a subset of the asset's artifacts, usually through the parameters on the asset.

7.2.3 Articulation

The articulation dimension describes the degree of completeness of the artifacts in providing the solution. Assets whose artifacts specify a solution but do not provide the solution have a low degree of articulation. Whereas assets whose artifacts specify and implement a solution along with supporting documents such as requirements, use cases, testing artifacts, and so on, have a greater degree of articulation.

7.3 Asset Packaging

Every reusable asset must contain at a minimum one manifest file, which are described below, and at least one artifact to be considered a valid reusable asset. The manifest file is an XML document that validates against one of the known RAS XML Schemas (see Manifest Schema), and passes an additional set of semantic constraints (see Semantic Constraints) described in the profile document.

An asset package is the collection of artifact files plus a manifest. There are several asset packaging scenarios including:

- Packaged bundled as an archive file

- Packaged unbundled

  - Artifacts may remain in their place of origin

  - Artifacts may be moved to another location when "packaged"

7.3.1   Bundled As Single Archive File

This approach to packaging may be used in a team development environment. But it also works well with less formal asset-based development processes as well as single user environments.

For this packaging approach using the <u>Zip</u> compression algorithm all the files, including the manifest file can be combined into a single archive file, making distribution of the asset easier. Rational XDE is an example of a tool that produces and consumes zipped RAS archive files. This approach to asset packaging is illustrated in the figure below.
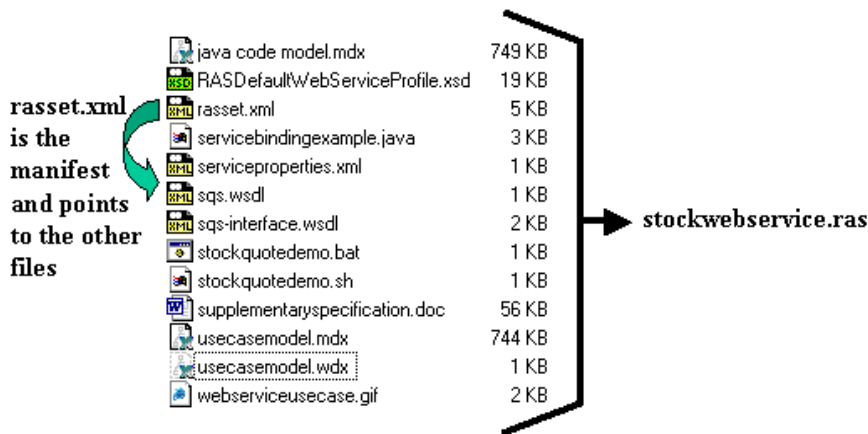
Figure 3 - Asset Packaging with Zip format

The directory in which a manifest file is located (on the filesystem or in an archive file) is considered the root context for all the asset's artifacts (files). All files referenced in the manifest file are referenced relative to the root context. When assets are packaged using the Zip format, as described above, all files referenced in the manifest must exist in the root context or in one of its sub directories. References to files from the manifest are relative to the root context.

By convention asset packages specifying a single asset should include a manifest file named "rasset.xml". However there is no restriction on the name of the manifest file, and it is up to any tool or the user to determine which files in the package are manifest files and which are artifacts of the asset.

There are also no restrictions on the inclusion of additional files in an asset's package. Additional files included in an asset package may exist for practical or pragmatic reasons necessary for any tooling or processes used. These files however should not be considered a part of the asset. All files that make up the asset and that are required to apply and use an asset must be referenced by exactly one <artifact> element in the <solution> element of the asset manifest. Files not referenced by an <artifact> element are considered to be not required by the asset, and it is perfectly legal for a tool to repackage the asset without the extra files and deliver the revised asset package and have it considered equivalent to the original asset package. There are two kinds of files that are exempt from this constraint. The first is the asset manifest file (i.e., rasset.xml) and the second is the RAS XML Schema file(s).

For aggregated assets or packages with more than one asset defined in them, the entire set of all <artifact> references for all manifest files is what determines which files are required to be kept in the package when transferring or replicating the asset.

7.3.2   Unbundled With Artifacts In Original Location

Developing artifacts in a team environment generally includes the use of version control systems. One approach to defining assets is to add the asset manifest file to the version control system and point to the artifacts in their original location. The RAS structure supports this style of asset "packaging".
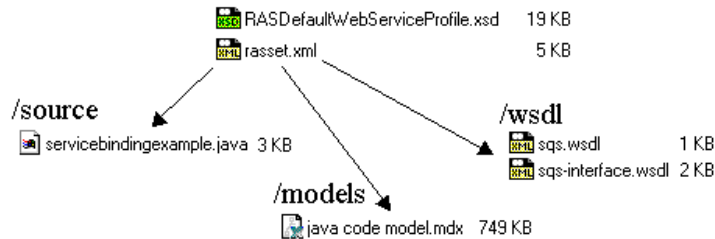
Figure 4 - Asset Manifest File Points To Artifacts In Original Location

7.3.3   Unbundled With Artifacts Moved To New Location

In many cases when artifacts are to be packaged within an asset, the artifacts require some modification to make them reusable. At this point the artifact may take on a new identity and be moved to a new location wherein it may be modified as necessary. Again, the RAS structure supports this scenario for asset "packaging".
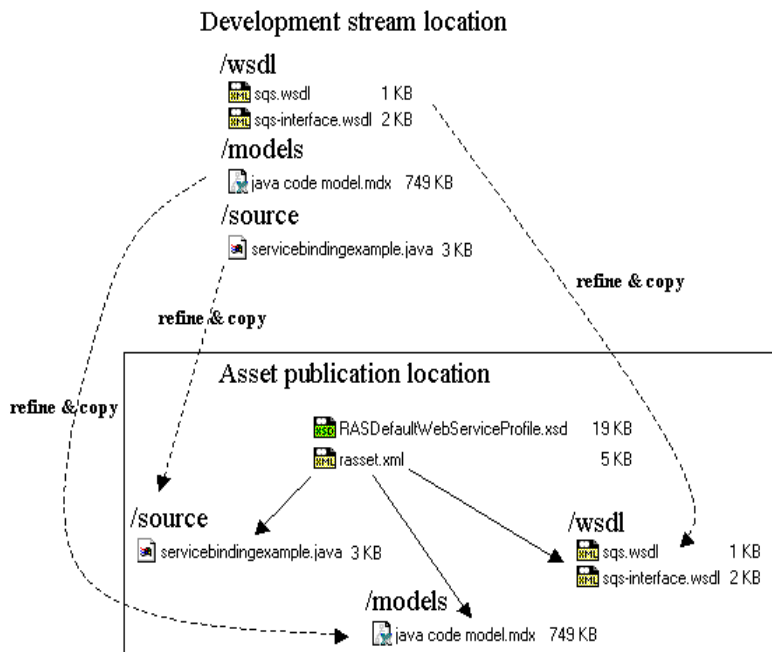
Figure 5 - Asset Manifest File Points To Artifacts In New Location

7.4 Core RAS 2.1

7.4.1 Core RAS and Profiles

RAS is described in two major categories, Core RAS and Profiles. Core RAS represents the fundamental elements of asset specification. Profiles describe extensions to those fundamental elements. A profile must not alter the definition or semantics of the nodes and elements defined in Core RAS.

The Core RAS is not instantiated therefore an asset must be of a particular profile. A profile may extend Core RAS or may extend another profile.

The image below illustrates the Core RAS and Profiles.
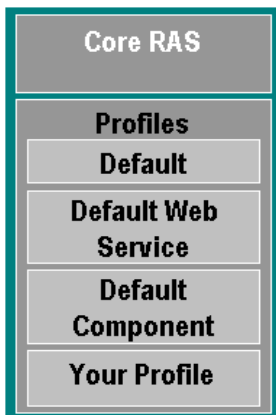
Figure 6 - Core RAS and Profiles

The image above shows the general relationship of the Core RAS and the profiles. However, the relationship is more accurately displayed in the image below. The Default Profile is a realization of the Core RAS. The Default Component Profile and the Default Web Service Profile derive from the Default Profile. The derivation information is captured in the profile history in each schema which is described later in this document.
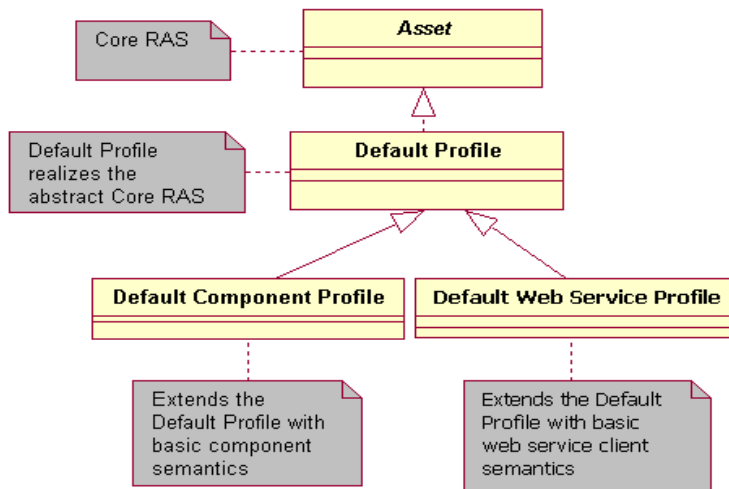
Figure 7 - RAS Profile Relationships

It is recognized that the Core RAS information specification may be improved for special circumstances. Using the same term and general goal as the UML extension mechanism a RAS Profile is a formal extension of the meta information structure. In general it is a way to add or augment information to the base (default) specification.

A RAS profile can be created to introduce tighter semantics and constraints. For example, a new profile may make current optional nodes to be required. But the constraints in the parent profiles cannot be removed. For instance, existing nodes cannot be made less constrained in the new profile than how they are defined in parent profiles.

Attributes on existing nodes can be added in new RAS profiles. However, the constraints on existing attributes cannot be reduced. For example, a new profile may make current optional attributes to be required. But the constraints in the parent profiles cannot be removed. Existing attributes cannot be made less constrained in the new profile than how they are defined in parent profiles.

Every manifest document must reference the XML schema document associated with the profile that can be used to validate the manifest document. The profile's schema document can be referenced with the xsi:schemaLocation as an attribute of the <asset> element. For example:

<asset xsi:chemaLocation="RAS_defaultprofile_ver2.1.xsd"

        name="My Asset" id="369BEA01-B4C2-4d47-99C8-6E44079207F1">

The actual filename may vary for the profile file, but the XML instance documents must reference the schema file. The schema file is expected to accompany the manifest document, however this is not required. The profile schema file may be referenced with a URL, and accessed through a network.

The image below identifies some major sections and elements of Core RAS. In the Asset section at the top of the image are some of the asset-level attributes. Core RAS defines four major sections to an asset including the Classification section, Solution section, Usage Section, and Related-Assets section.
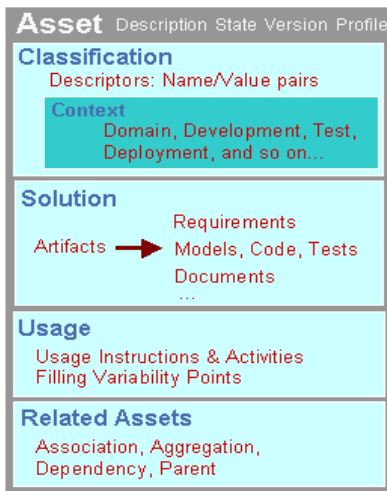
Figure 8 - Major Sections of Core RAS

The image above illustrates the general structural elements of Core RAS. An asset is specified by these various sections which are contained in the asset's meta data, as shown in the image below.
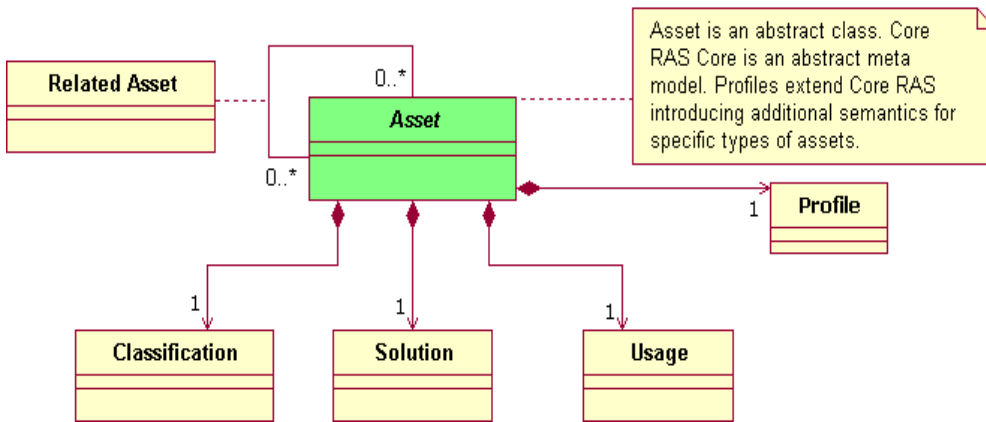
Figure 9 - Core RAS Domain Model - Major Sections

The collection of discrete artifacts in an asset can be overwhelming even for moderately sized assets. The RAS helps by specifying how the artifacts are organized and which pieces of meta-documentation of the asset (the information describing the asset) are required. It structures the asset into sections, as shown in the figure below. These sections (which, returning to our car analogy, are like the steering wheel, turn signals, and so on) include:

- The Classification section, listing a set of descriptors for classifying the asset as well as a description of the context(s) for which the asset is relevant.

- The Solution section, describing the artifacts of the asset.

- The Usage section, containing the rules for installing, customizing, and using the asset.

- The Related Assets section, describing this asset's relationship to other assets.

While this is a general representation of assets, there is certainly more required to specify certain kinds of assets such as web services, patterns, components, and frameworks. RAS can be extended through profiles. Several groups have started working on such profiles such as a Component profile and a Web Service profile.

These profiles preserve and extend the core description of RAS given above.

7.4.2   Core RAS Model and XML Schema Overview

This section of the specification explains the structure of the manifest document. From the UML model is derived an XML Schema document, which is the authoritative description for the manifest document. The XML Schema is not sufficient to describe completely a valid RAS manifest document. Additional semantic constraints are summarized later in this document (see Semantic Constraints). This section outlines both the XML document elements and structure as well as key semantic constraints associated with each element.

This document presents each of the asset elements using the Rose model as the medium. For each class representing an asset element, the XML Schema element is used. For instance, for the Asset model element, the <asset> XML schema element is also used to describe the Asset.

Multiple tool vendors are currently using the RAS XML schema files included with this document. As such this document describes both the incumbent XML schema files and the newly formed MOF/XMI schema expressions of RAS.

The UML model below articulates the key classes comprising the asset specification. This model is at the level from which an XML schema could potentially be generated. The aggregation relationships between the classes declare the element owners and containers. The association relationships describe asset element associations wherein an id is generally needed to persist the relationship.
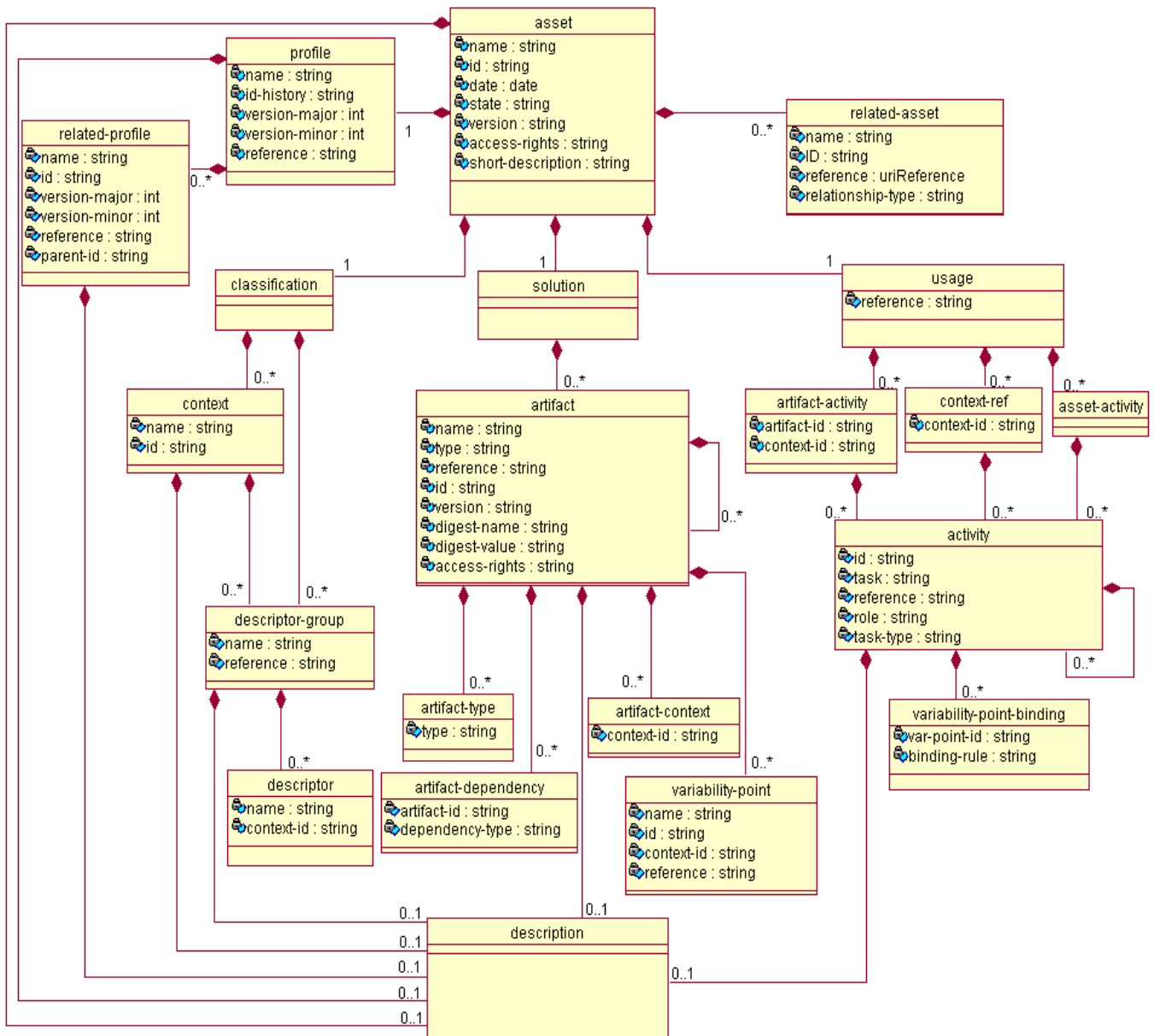
Figure 10 - Core RAS UML Model for XML Schema

Figure 11 - Core RAS UML Model for MOF 2.0 XMI

This Core RAS UML model is translated into an XML schema representation as illustrated in the image below. The major sections from the RAS UML models are represented with a brief description of each element.
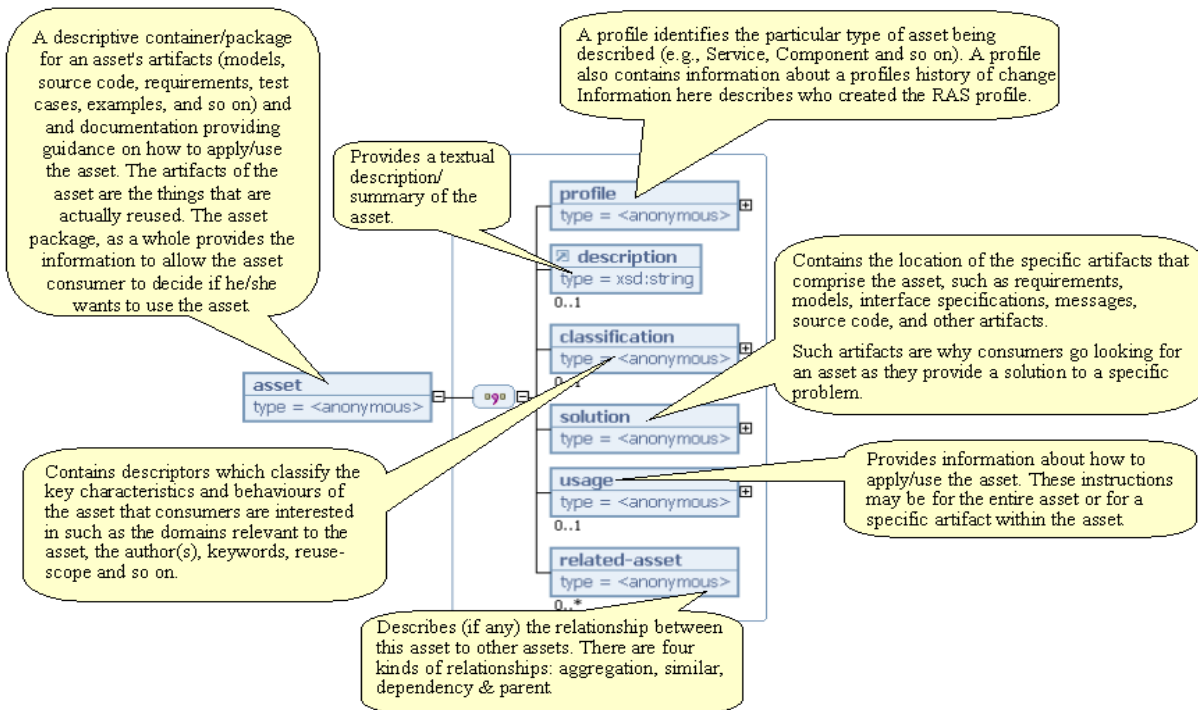
Figure 12 - RAS Default Profile XML Schema Overview

7.4.3   RAS Compliance

An asset is RAS compliant if all of the following conditions are true:

The asset enforces all the <XREF>semantic constraints described for Core RAS in this document.

7.4.4   Required Classes

The semantic constraints section describes what elements must have some values for RAS compliance to be achieved. The purpose for this is to support a spectrum of reuse formalities. Some organizations are prepared for informal reuse and lightweight packaging costs, whereas other organizations may be on the other end of the spectrum.

A RAS profile could be created to introduce tighter semantics and more required elements; but the existing required elements cannot be made optional in a new RAS profile, see *Constraint 16*.

The required elements are listed below:

- Asset
- Profile
- Solution

Although the Solution class is required and the associated Artifact class is optional, *Constraint 2* states that there must be at least one Artifact with a ***name*** and a ***reference*** to be RAS compliant. The <artifact> element is optional to ease future profiles that might add more specific nodes in the solution section (such as requirements, design, implementation and so on) and may want to make them required. A key rule for creating new profiles is to not alter the constraints of parent profiles.

7.4.5   Required Attributes

Very few of the attributes are required. The semantic constraints section describes what additional attributes must have some elements for RAS compliance to be achieved. The purpose for this is to support a spectrum of reuse formalities. Some organizations are prepared for informal reuse and lightweight packaging costs, whereas other organizations may be on the other end of the spectrum.

A RAS profile could be created to introduce tighter semantics and more required attributes; but the existing required attributes cannot be made optional in a new RAS profile, see *Constraint 16*.

The **required attributes** are listed in the table below; many of these attributes reside on optional classes, meaning the XML schema node for the class is not required for packaging an asset.

Table 1 - Core RAS::UML Model for XML Schema Required Attributes

Core RAS UML Model for XML Schema				
Required Class	**Required Attribute**		**Optional Class**	**Required Attribute**
asset	name		related-profile	name
asset	id		related-profile	id
profile	name		related-profile	version-major
profile	id-history		related-profile	version-minor
profile	version-major		context	name
profile	version-minor		context	id
			descriptor	name
			artifact-context	context-id
			artifact-dependency	artifact-id
			variability-point	name
			variability-point	id
			artifact-type	type
			artifact-activity	artifact-id
			context-ref	context-id
			activity	id
			activity	task
			variability-point-binding	variability-point-id
			variability-point-binding	binding-rule
			related-asset	name
			related-asset	relationship-type

There are fewer required attributes in the MOF/XMI XML Schema because of the id support in XMI and the fact that relationships can be modeled and supported as part of the schema itself.

Table 2 - Core RAS::UML Model for MOF 2.0 XMI Required Attributes

Core RAS UML Model for MOF 2.0 XMI				
Required Class	**Required Attribute**		**Optional Class**	**Required Attribute**
Asset	name		RelatedProfile	name
Asset	id**		RelatedProfile	id***
Profile	name		RelatedProfile	versionMajor
Profile	idHistory		RelatedProfile	versionMinor
Profile	versionMajor		Context	name
Profile	versionMinor		Descriptor	name
			VariabilityPoint	name
			ArtifactType	type
			Activity	task
			VariabilityPoint-Binding	bindingRule
			relatedAsset	name
			relatedAsset	relationshipType

** This attribute is not formally specified in the model because XMI provides id support by default; these ids in XMI are optional and therefore this table specifies constraints on the id that it is required.

*** This id attribute *DOES* reside in the model and is the profile id for a profile which is an ancestor to the current profile and which is not the <profile> id from the current asset's manifest.

7.4.6   Asset

Every RAS manifest document begins with a single Asset instance. This Asset instance defines the identity of the reusable software asset (see Asset Identity section). This Asset instance contains two required attributes; *name* and *id*. For the XMI XML schema the *id* does not appear in the model as it relies on the XMI generator to produce it.

The *name* identifies the asset in a few words and is intended for human consumption, whereas the *id* attribute is expected to contain a globally unique identifier and is used by tooling to distinguish assets.

An asset's *name* and *short description* are typically the first pieces of information that potential consumers see when searching asset repositories. An asset's *name* should reflect the general solution strategy of the asset and optionally the problem that it addresses. The *short description* should be suitable for use in a line item where multiple asset names and short descriptions are displayed to a potential consumer.

The *date* attribute contains a valid date using the default XML format (YYYY-MM-DD). The *date* indicates the date that the asset is ready to be used by asset consumers.

The *state* attribute indicates the state that the asset is currently in. This is intended primarily for asset certification workflows as an asset is undergoing reviews in preparation to be published in a repository.

The asset's *version* attribute can be any string and is used to compare two assets with the same *id* attribute.

The asset's *access rights* attribute can be any string which describes the permissions of asset consumers for interacting with the asset such as viewing or using.

The Asset class has two required associations, Profile and Solution and four optional associations Description, Classification, Usage, and RelatedAsset. These classes are discussed throughout this document.

Table 3 - Core RAS::Asset Class

Asset	
UML Model for XML Schema	**XML Schema**
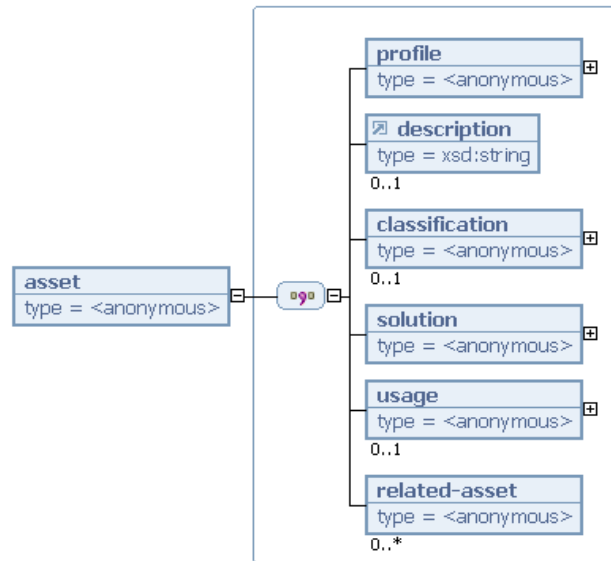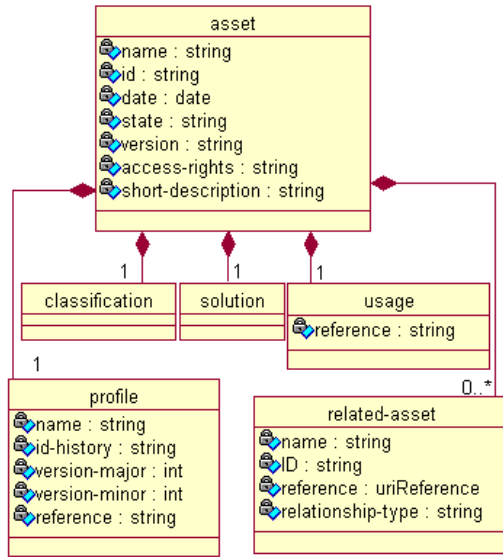
- **Required**: true
- **Document global**: false
- **Unbounded**: false

UML Model for MOF 2.0 XMI	**XML Schema**
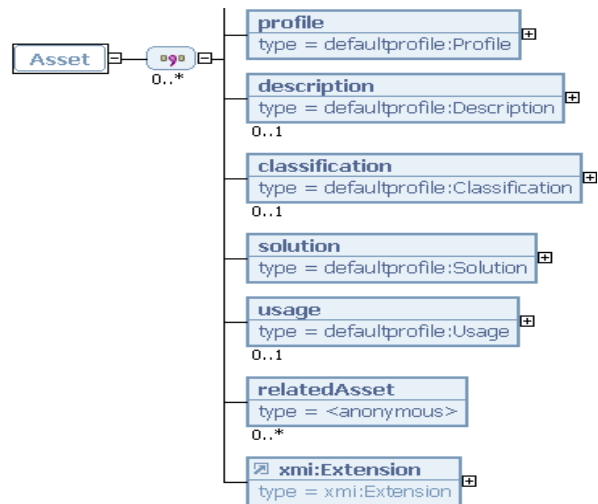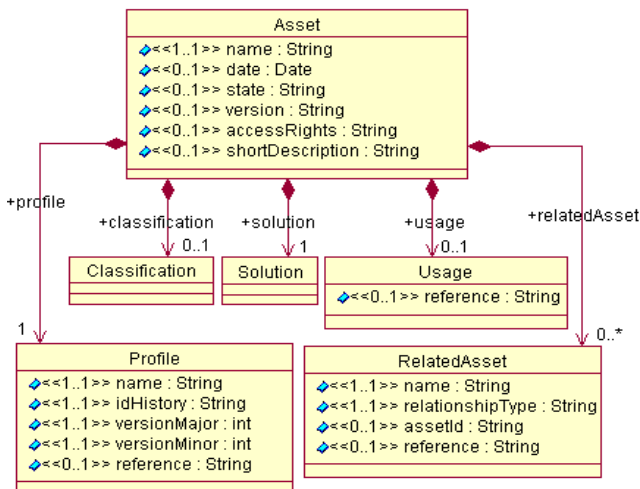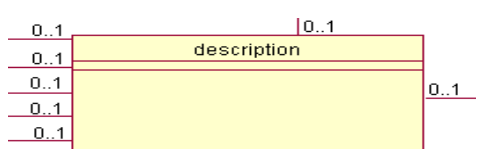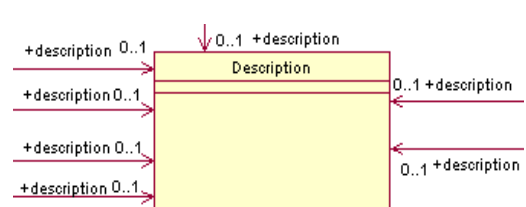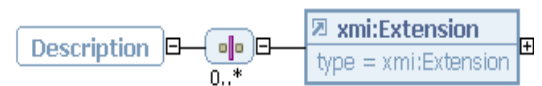
7.4.7 Description

The Description class is a simple container for a human readable description of the asset. This description is expected to be about one or two paragraphs in length, however there is no restriction on size specified by this document. It describes in some detail the problem that the asset addresses and its main solution strategies.

It is possible for the content of the Description element to be formatted with HTML. The Description is global in the XML Schema and is referenced in multiple places.

This class does not have any attributes but supports mixed text..

Table 4 - Core RAS::Description Class

7.4.8 Profile

An asset is defined by one Profile; a Profile describes the asset's type. The Profile can have different versions and should declare its lineage or ancestry from other profiles. The RelatedProfile captures information on the Profile's lineage.

The Profile class in the XML schema includes information about the format of the manifest itself. It identifies exactly which version of this specification and which RAS profile should be used to validate the manifest document for compliance. A Profile defines the structure and semantics of an asset's manifest document. Every RAS manifest document must identify the Profile that can be used to validate it. Every Profile is derived from another Profile with the one exception being the original Core Profile, which was defined by the first version of the RAS and for which there is no XML Schema produced. Profiles can extend directly from Core RAS or from any other profile such as the Default Profile for version 2.1. These derived Profiles can only add elements and attributes to the manifest's XML Schema, and/

or associate new semantics to existing elements. They cannot remove elements or attributes from the XML Schema. In general derived Profiles are more restrictive. This attempts to make it easier for tools to gracefully handle assets created with profiles defined after the tooling was created.

Each Profile specifies a human readable *name* that reflects the purpose or scope of the Profile. The authoritative identifier of a profile is its *id*. A profile's *id* can be any sequence of characters but must not contain a double colon (::). The examples in this document use Microsoft style GUIDs[1] , however the specification only requires the ids to be unique within the expected scope of reuse, and not to include a double colon.

The *id-history* is a composite key that is made up of the Profile *id* followed by the Profile ids of all the Profiles that it is derived from. A Profile is derived from exactly one parent Profile with the notable exception of the first and original Core profile introduced with the RAS 1.0 specification.

As an example the following is the *id-history* for the RAS Default profile version 2.1:

F1C842AD-CE85-4261-ACA7-178C457018A1::31E5BFBF-B16E-4253-8037-98D70D07F35F

It indicates that the profile identified by: "F1C842AD-CE85-4261-ACA7-178C457018A1" is derived from the profile identified by "31E5BFBF-B16E-4253-8037-98D70D07F35F", which is the Core profile defined in the first version of the RAS specification.

If a new Profile is defined, a new *id* is be generated and is pre-pended to the *id-history* of the Profile that it derived from. For example, using GUIDs as ids it might be:

F8C49799-25C9-4312-B798-D5D2E1FBC656::F1C842AD-CE85-4261-ACA7-178C457018A1::31E5BFBF-B16E-4253-8037-98D70D07F35F

This new Profile defines a new set of elements, attributes and semantics that extend those already defined by all of the other profiles in the id-history.

The *version-major* and *version-minor* attributes help identify the Profile version, and in particular helps distinguish it from previous Profiles with the same name. These attributes are integers. Often these two values are combined together with a period, and form what appears to be a floating-point number. For example a version major of 2, and version minor of 1 might be written as version 2.1.

Changes in the version major and version minor values should be made when a profile is updated and when its name remains the same. This would be the case when a profile is updated but its target purpose or scope, and hence name remains the same.

The *reference* attribute is an optional attribute that references an external document that contains more information about the profile. This document should explain the new elements, attributes and semantics of the profile used. The *reference* attribute can also contain a URL reference that points to a resource outside of the root context (i.e. http://www.myorg.org/ profiles/newProfile.html).

The Profile class has two associations, one with Description, which used to capture human readable comments that describe the Profile, and one with RelatedProfile, which provides human readable information about each of the Profiles in the *id-history*.

---

1. Specifically GUIDs encoded using style D. For more information on the Format Provider Specifier values (N, D, B, and P) refer to the .NET Framework Class Library documentation for the Guid.ToString(String, IFormatProvider) function.

Table 5 - Core RAS::Profile Class

Profile

UML Model for XML Schema	XML Schema
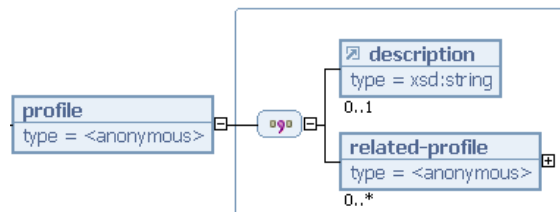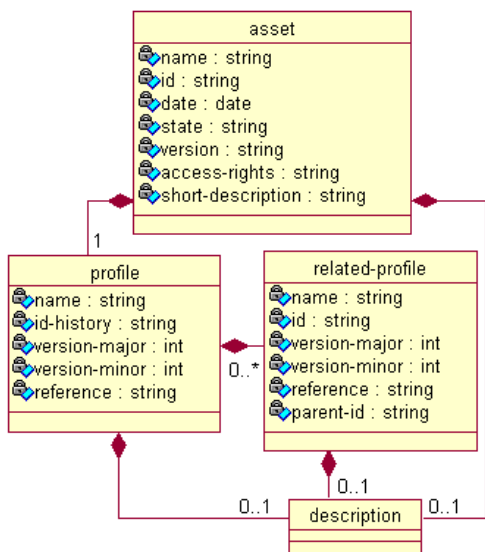
- *Required*: true
- *Document global*: false
- *Unbounded*: false

UML Model for MOF 2.0 XMI	XML Schema
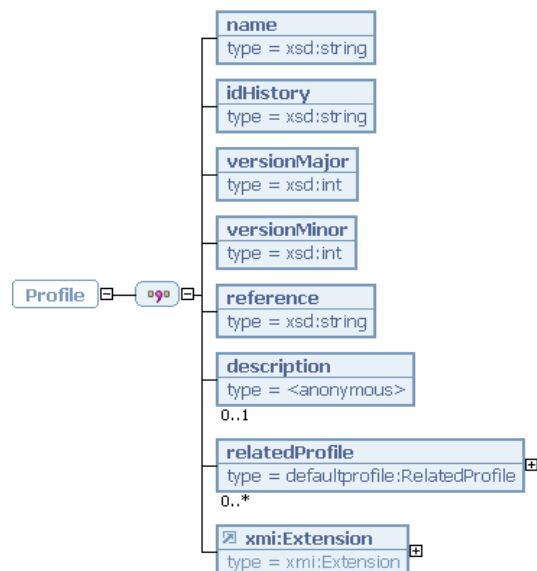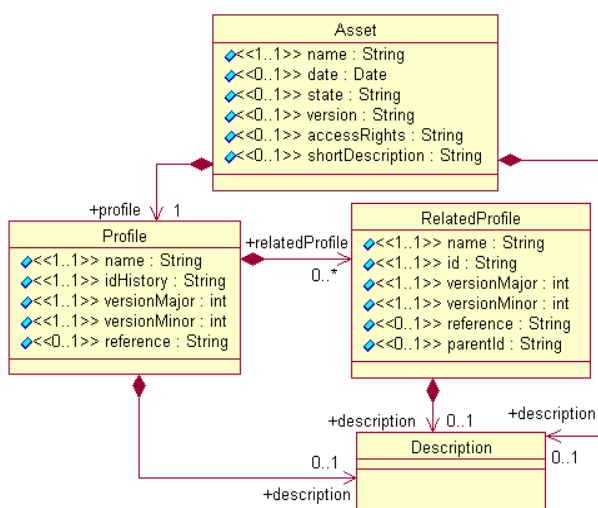
7.4.8.1 RelatedProfile

This node captures the history of the profile by describing the profile's genealogy. This element includes many of the same attributes found in the <profile> element, however there is no *idHistory* attribute.

This node has the attributes *name*, *id*, *version-major*, *version-minor*, *reference*, and *parent-id*. The *name* contains the profile name. The *id* contains the profile's id, which should appear in the *id-history* attribute of the <profile> element.

The *version-major*, and *version-minor* attributes contain the ancestor profile's version information. The *reference* contains a pointer to a document, which describe the profile in more detail. The *parent-id* describes the profile's ancestor from which it was derived.

Table 6 - Core RAS::RelatedProfileClass
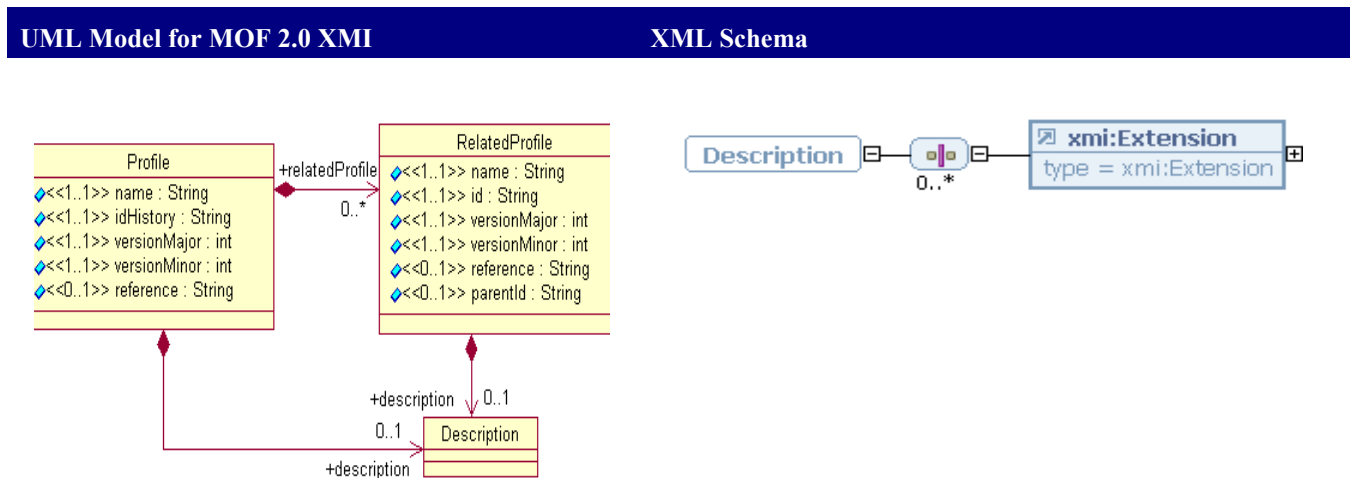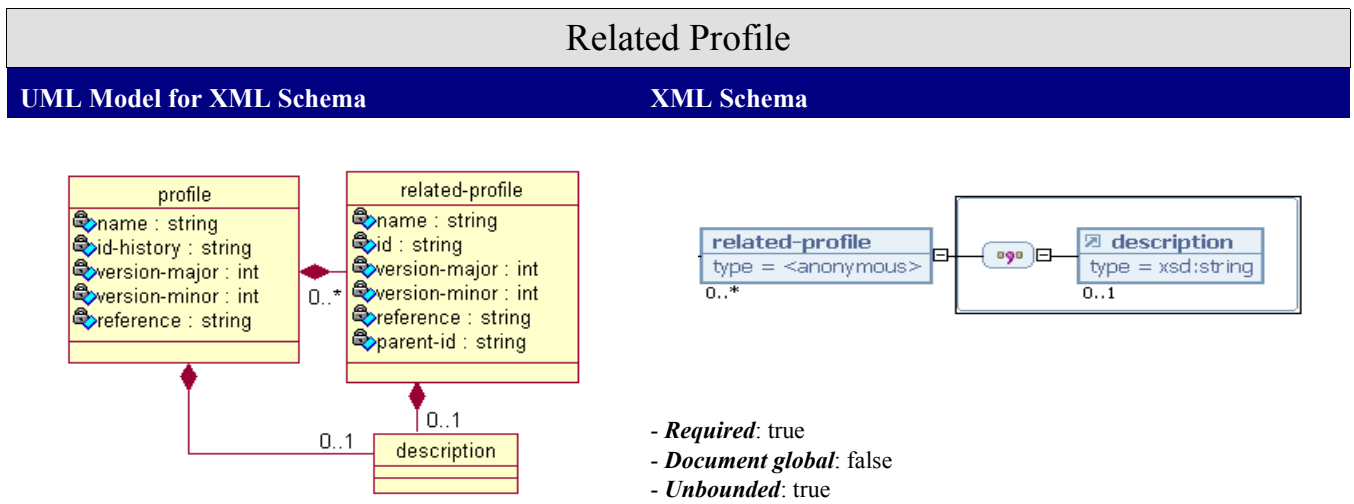
7.4.9   Classification

An asset is classified from one or more perspectives. This generally causes challenges for those searching for assets if they are looking for an asset from a perspective which is different than the one in which it was packaged.

Tool vendors should consider supporting multiple ontologies whereby an asset may be classified and discovered. The classification section supports simple name / value pairs and supports pointing to external schemas for classifying assets.

The <classification> element is simply a container for all the elements of the manifest that classifies the asset. The <classification> element does not define any attributes. There are two child elements; <context> and <descriptor-group>. These elements are optional however; there should be at least one descriptor group element, which contains the main descriptors for the asset.

Table 7 - Core RAS::Classification Class
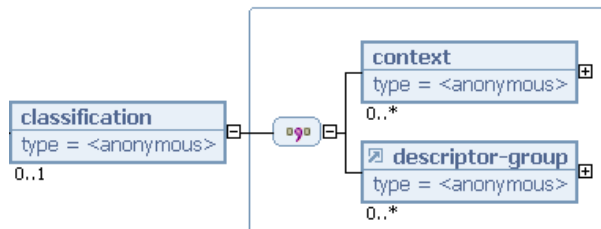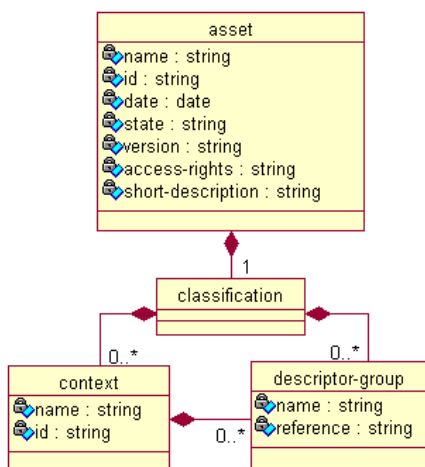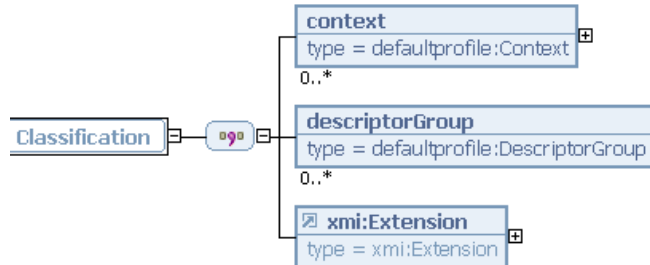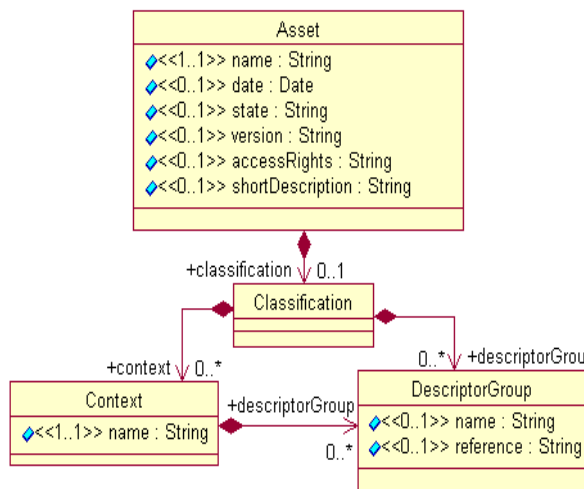
7.4.9.1 Context

A context defines a conceptual frame, which helps explain the meaning of other elements in the asset. A <context> element defines a *name* attribute, which is required, and an *id* attribute. The id is used as a reference by other elements in the manifest. A <context> element's optional description is captured by a <description> child element. A <context> element also may include <descriptor-group> child elements that act to help define this context with descriptors (see <descriptor>)..

Table 8 - Core RAS::Context Class
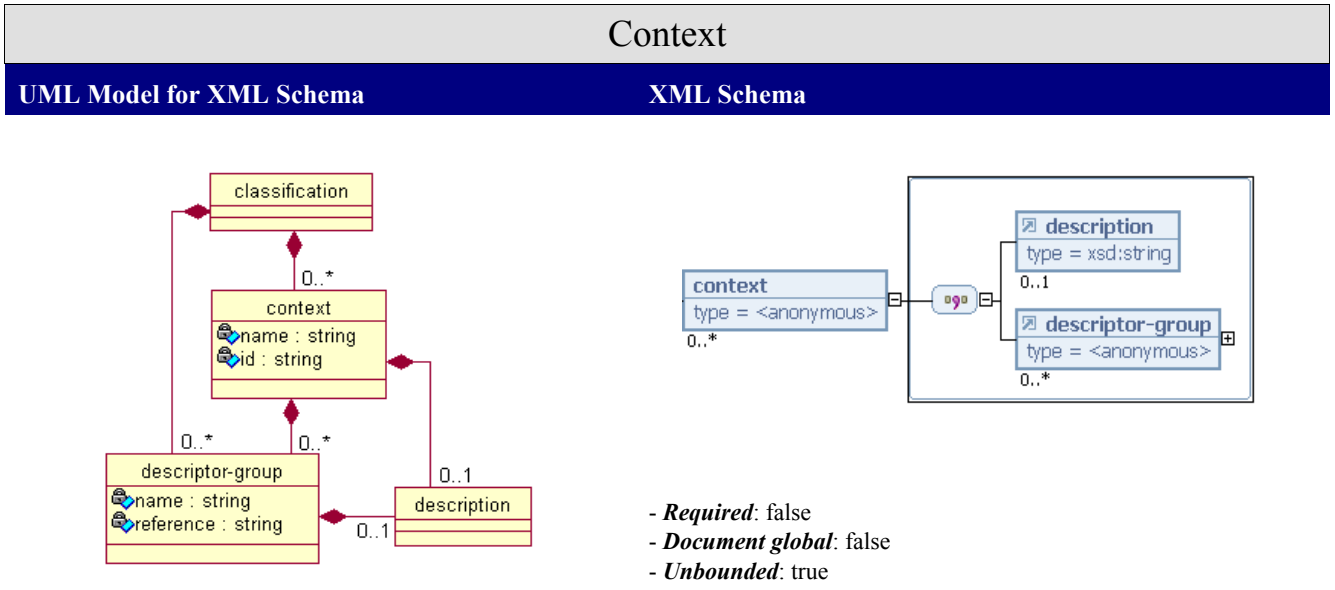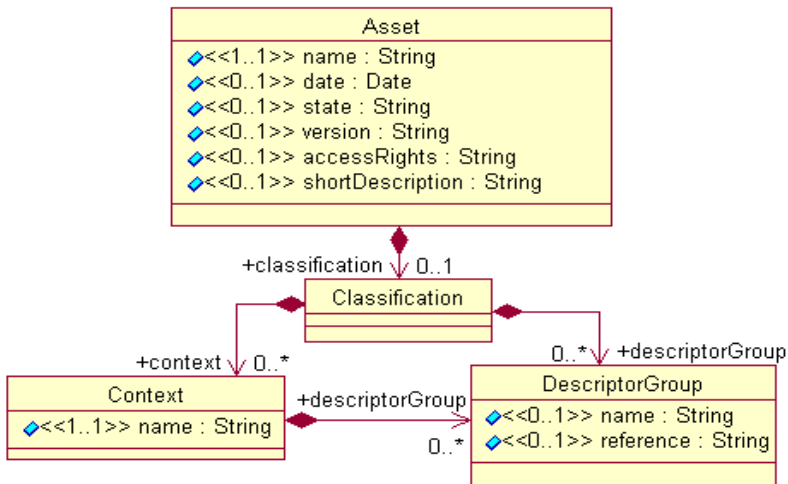
Table 8 (continued)

XML Schema
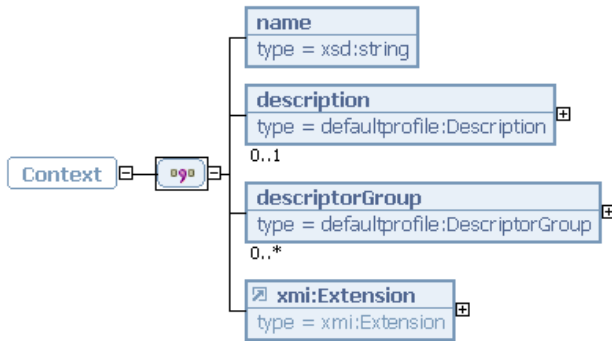
There can be many contexts defined in a manifest. An <artifact> may declare its relevance to more than one context. Some sample categories of contexts are described in the table below. RAS does not declare what contexts to use; therefore the sample contexts below are merely illustrative.

Table 9 - Context Categories

Context Categories	Context Description and Example
Core	Artifacts associated with this context represent things that are essential to the asset. Without the artifacts or activities associated with this context the asset cannot be successfully applied to a target application.
Business	Artifacts associated with this context are relevant to a specific business context. Example: Insurance Example: Financial Services
Development	Artifacts associated with this context are required for the development of the asset. This context is usually included in white box assets where it is intended for some of the artifacts of the asset to be modified. Artifacts associated with this context might include build scripts and tools, models, and specification documents. Example: J2EE 1.3 Example: WebSphere Studio Application Developer 5.1
Documentation	Artifacts associated with this context are intended to document and explain the asset. They are not necessarily required to apply it.

Table 9 - Context Categories

Runtime	Artifacts associated with this context are required for the runtime execution of the asset. Example: WebSphere Application Server 5.1
Test	Artifacts associated with this context make up the unit test infrastructure. They can be scripts, sample data or test plans. These elements are not expected to be applied directly to the target application.

7.4.9.2   DescriptorGroup

A descriptor group is simply a container for a related group of descriptors. It has optional *name* and *reference* attributes. The *name* attribute describes the group of descriptors. The *reference* attribute is used to point to an external document that provides additional information about the group.

RAS does not describe all possible classification schemas for all possible industries and asset types. Therefore, the *reference* attribute may point to another classification schema or ontology.

A <description> child element can be used to provide a description of the group.

Table 10 - Core RAS::DescriptorGroup Class
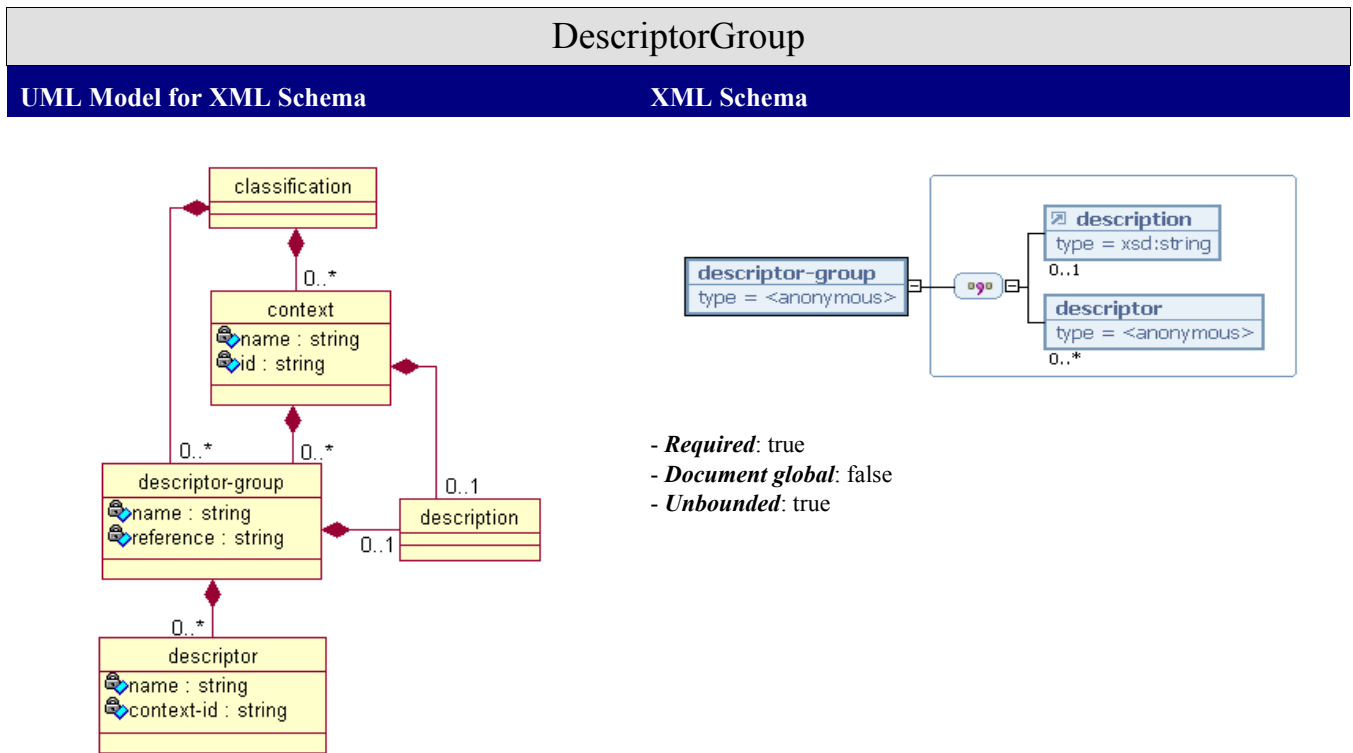
Table 10 - DescriptorGroup (continued)

UML Model for MOF 2.0 XMI	XML Schema
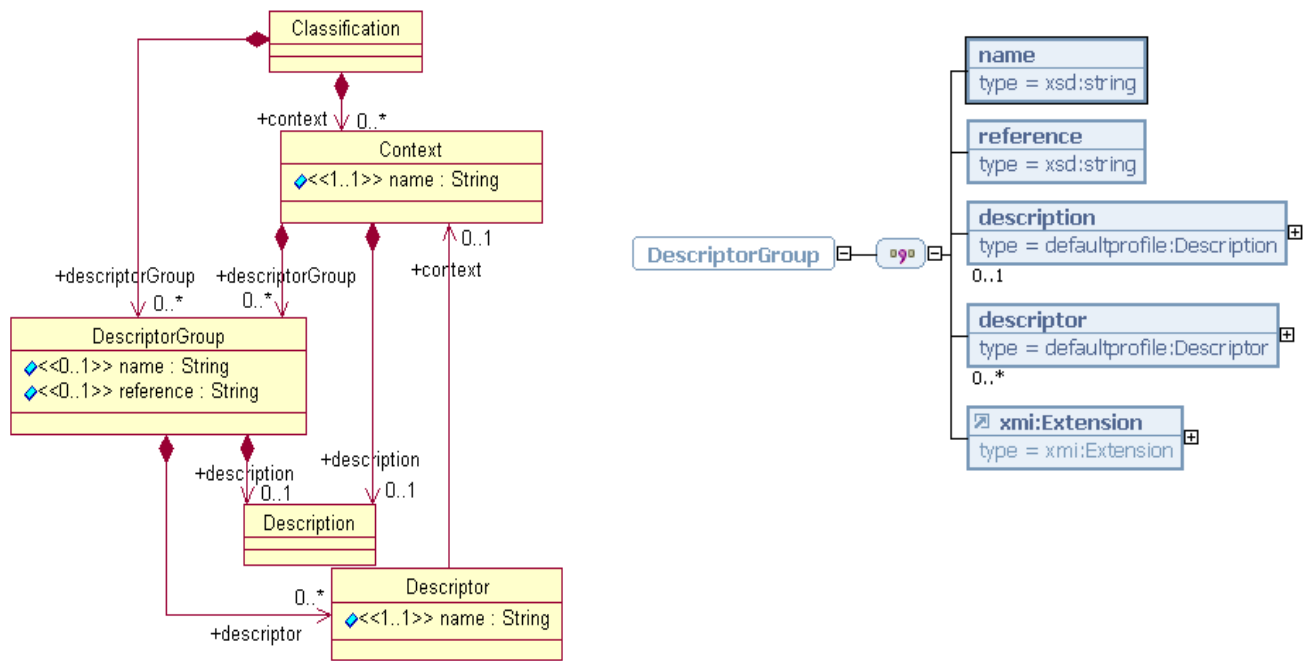
7.4.9.3 Descriptor

A <descriptor> is a simple key/value pair that emphasizes certain qualities and characteristics of the asset. Often these are searchable and may be used for asset discovery and evaluation. The key is specified by the *name* attribute, and is typically a word or two. The value is captured as the elements content in mixed text.

A <descriptor> may be associated with a specific <context> through the context association. This means that the key and value should be thought of in the specified <context>. The value of the context association must reference an existing id of a <context> element.

.

Table 11 - Core RAS::Descriptor Class

Descriptor	
UML Model for XML Schema	**XML Schema**
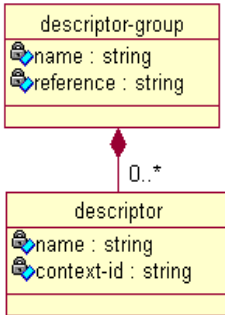
- *Required*: true
- *Document global*: false
- *Unbounded*: true

UML Model for MOF 2.0 XMI	**XML Schema**
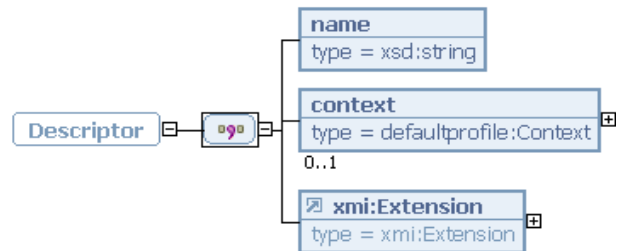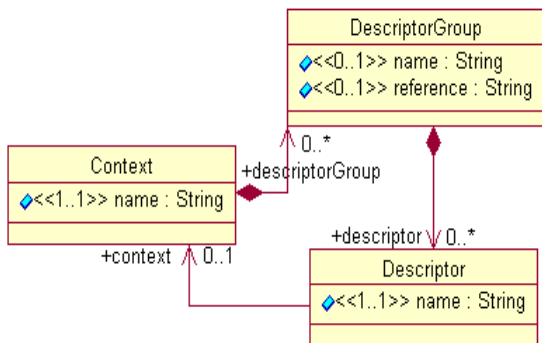
7.4.10 Solution

An asset provides a solution, which is found in a collection of artifacts. An artifact may contain another artifact or may have a relationship to another artifact. An artifact may be relevant to a particular context such as a development or a runtime context. An artifact may have a point of customization known as a variability point.
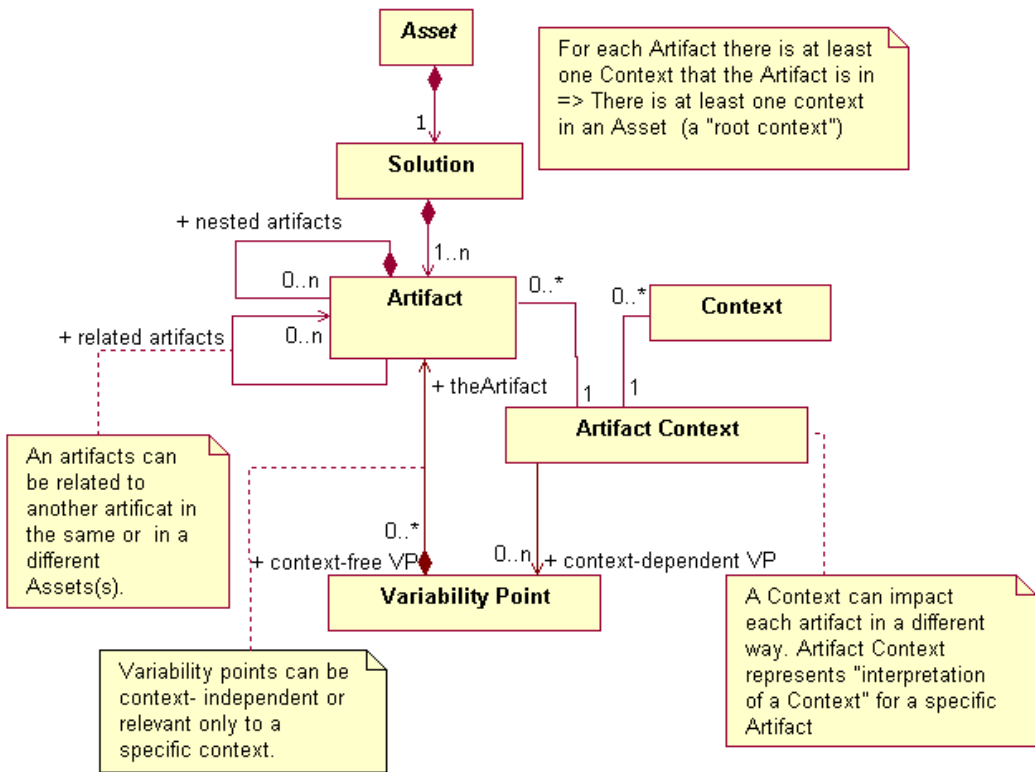
Figure 13 - Solution Section Domain Model

The <solution> element in a manifest is a simple container for all the artifact references of the asset. It is a required element and specifies no attributes. The <solution> element specifies only <artifact> child elements.

Table 12 - Core RAS::Solution Class

Solution	
UML Model for XML Schema	**XML Schema**
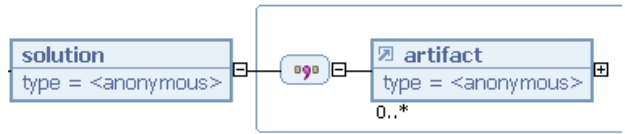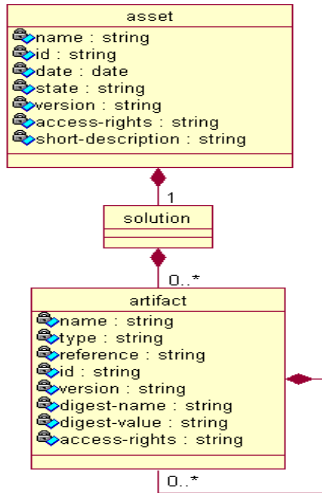
- *Required*: true
- *Document global*: false
- *Unbounded*: false

UML Model for MOF 2.0 XMI	**XML Schema**
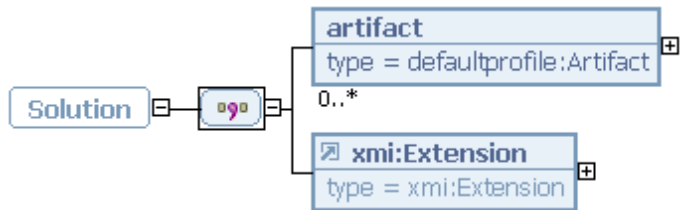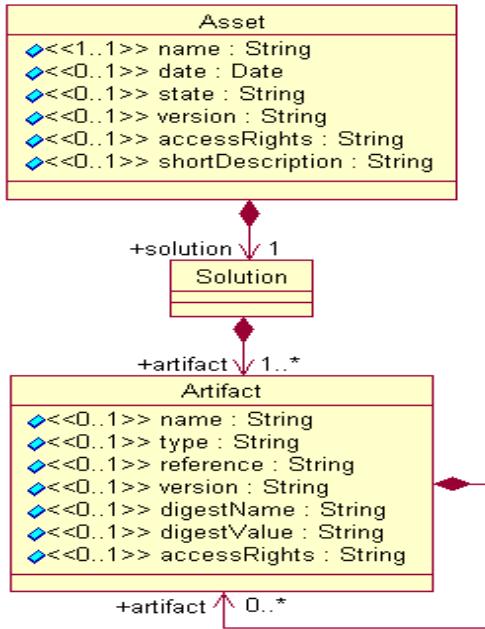
7.4.10.1 Artifact

An artifact is a work product that can be created, stored and manipulated by asset producers/consumers and by tools. An artifact is either an actual file located in the asset's package, or represents a logical entity that contains at least one child artifact that is an actual file. An <artifact> element must specify minimally a **name** or a **reference** attribute. The **name** is required for artifacts that represent logical entities. The **reference** is required for artifacts that specify actual file or work products that are part of the asset's packaging.

The **name**, **type**, and **reference** attributes are optional. This was done to allow large numbers of artifacts to be added by tooling but for which each specific name may not be known. In this scenario the **name** may have become the filename, but that would be redundant with the **reference** attribute. The **reference** attribute remains optional to allow <artifact> elements to contain other <artifact> elements and to reference anything on the filesystem or elsewhere.

An <artifact> element can specify an **id** that can be referenced by other elements in the manifest. This **id** must be unique within the scope of all the artifacts in the manifest document. An optional **version** attribute is used to identify an artifact's version.

With ever-increasing needs for security an artifact may be encrypted with a specific algorithm. The **digest-name** and **digest-value** attributes contain the name and value from such encryption activities.

A specific artifact may have its own permission and usage rights associated with it. The **access-rights** attribute captures artifact-level permission information. RAS does not specify the format of this permission information.

The <artifact> element may contain a number of child elements; <artifact-type>, <artifact-context>, <artifact-dependency>, <variability-point> as well as specify child artifacts.

Artifacts that represent actual files can be of any type. That is they can be any type of file (binary, plain text, etc.). An artifact referenced in a manifest can optionally declare a primary type, which is captured in the **type** attribute. The primary type can affect how tools process the artifact. In addition to a primary type an artifact can specify any number of secondary types. Each secondary type is specified with a <artifact-type> child element (see <artifact-type>).

In practice the primary types tend to map to file extensions. For instance if we had a file with the name web.xml its primary type is XML and its secondary type may be J2EE Web Configuration.

The primary type list maps file extensions to type names. There may be many file extensions to the same type name. It is also possible to have many type names to the same file extension. In this case the tooling should provide a way for the user to reconcile. For instance, you may have a file named usecases.doc. The .doc extension may map to a "Microsoft Word" type and it may map to a "WordPerfect" type.

Primary Types

A sample list of these primary types, taken from the file RASPrimaryArtifactTypes.xml is below.

```
<artifact id="dothtml" type="Microsoft Word HTML Template"/>
<artifact id="dox" type="Visual Basic User Document Binary File"/>
<artifact id="dqy" type="Microsoft Excel ODBC Query files"/>
<artifact id="drv" type="Device driver"/>
<artifact id="dsm" type="DSM File"/>
<artifact id="dsn" type="Microsoft OLE DB Enumerator for ODBC Drivers"/>
<artifact id="dsp" type="Project File"/>
<artifact id="dsr" type="Visual Basic Designer Module"/>
<artifact id="dsw" type="Project Workspace"/>
<artifact id="dsx" type="Visual Basic Designer Binary File"/>
<artifact id="dtd" type="Document Type Definition"/>
```

```
<artifact id="dun" type="Dialup Networking File"/
<artifact id="dv" type="DV"/>
<artifact id="DVD" type="DVD"/>
<artifact id="ecs" type="Exchange Server Content Source"/>
<artifact id="elm" type="Microsoft Office Themes File"/>
<artifact id="eml" type="Internet E-Mail Message"/>
<artifact id="ent" type="External Entity"/>
<artifact id="enx" type="Rational XDE Unit"/>
<artifact id="exc" type="Text"/>
<artifact id="mdx" type="XDE Model"/>
<artifact id="ifx" type="Rational XDE Unit"/>
<artifact id="inx" type="Rational XDE Unit"/>
```

This list is dynamic and should be used by tool builders to provide the proper processing of artifacts.

The <artifact-context> element allows multiple contexts to be associated with the artifact. Each of these elements defines a single *context*.

The <artifact-dependency> element identifies a dependency on another artifact in the asset. Each element specifies an *artifact-id* attribute, which must contain a value that matches the id value of another artifact in the manifest. This value must not reference itself. An optional attribute, *dependency-type*, is used to describe the type of dependency. There are several kinds of artifact dependencies such as a compile-time dependency or a runtime dependency, and so on.

A <variability-point> is a conceptual spot in an artifact that is expected to be altered by the asset consumer. It describes where and what in the artifact can be modified. Each <variability point> specifies a name, which describes it and an identifier, which is used to reference it from other elements in the manifest. A variability point may be associated with a context and its optional context association must therefore specify a valid id of a <context> element in the document. A further explanation of the <variability-point> can be captured in an external document pointed to by the *reference* attribute.

An artifact may specify child artifacts. A child artifact uses the same <artifact> element. If the artifact is a logical artifact then it must specify a *name* and have at least one descendent artifact that is an actual file reference.

Table 13 - Core RAS::Artifact Class

Artifact	
UML Model for XML Schema	**XML Schema**
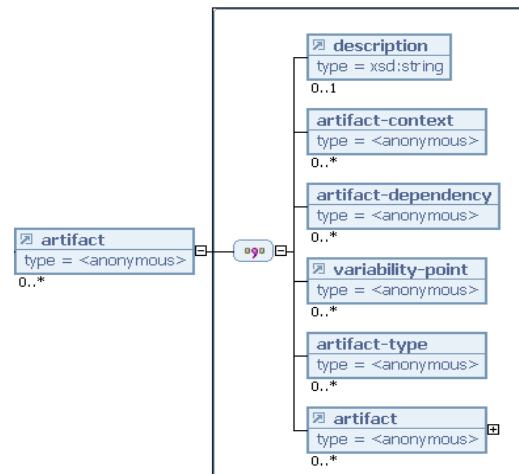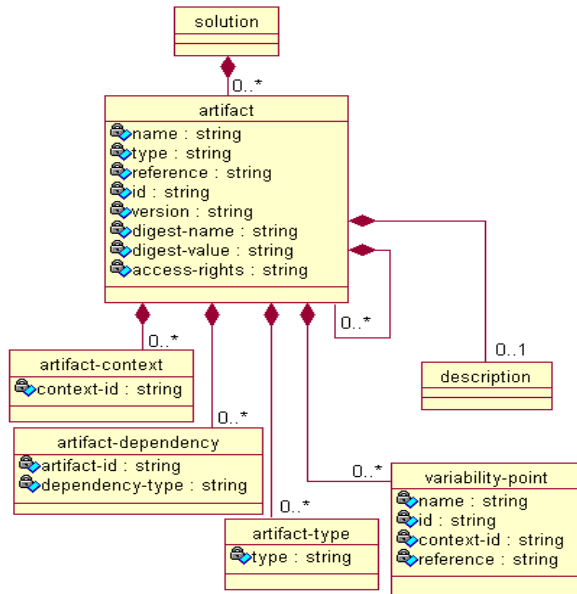
- *Required*: schema states false, constraints require at least one
- *Document global*: true
- *Unbounded*: true

Table 13 - (continued)

UML Model for MOF 2.0 XMI	XML Schema
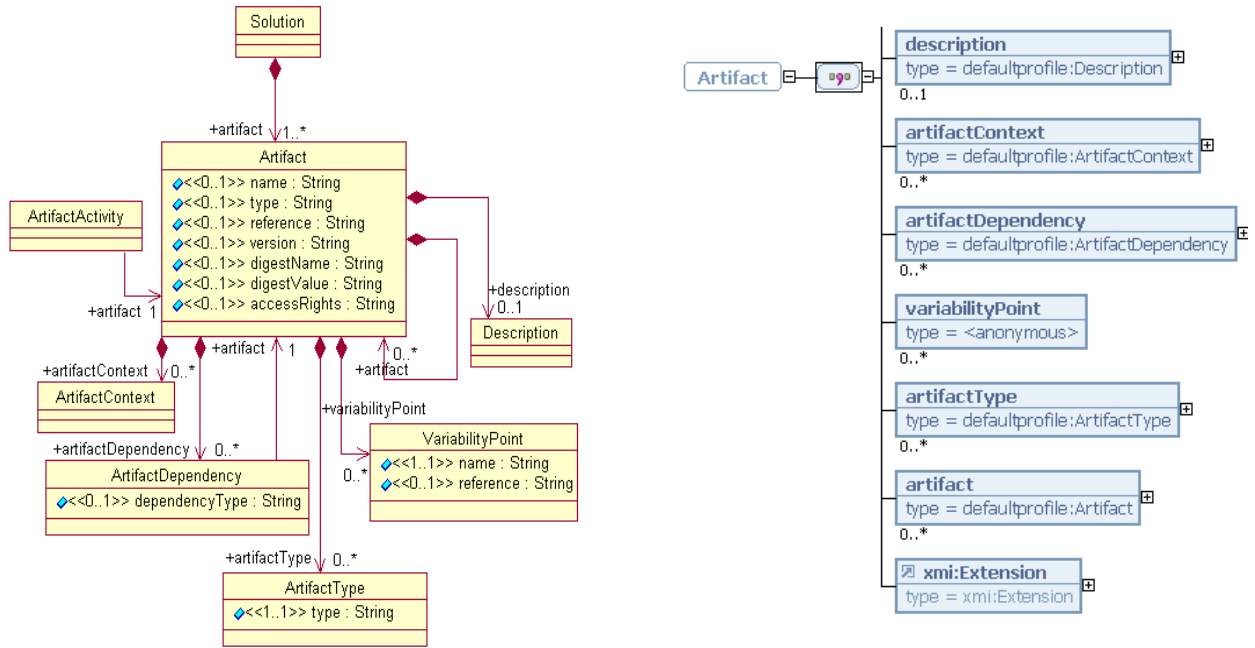
7.4.10.2 ArtifactContext

An <artifact-context> element associates a context to an artifact. See <artifact> element description..

Table 14 - Core RAS::ArtifactContext Class

ArtifactContext	
UML Model for XML Schema	**XML Schema**
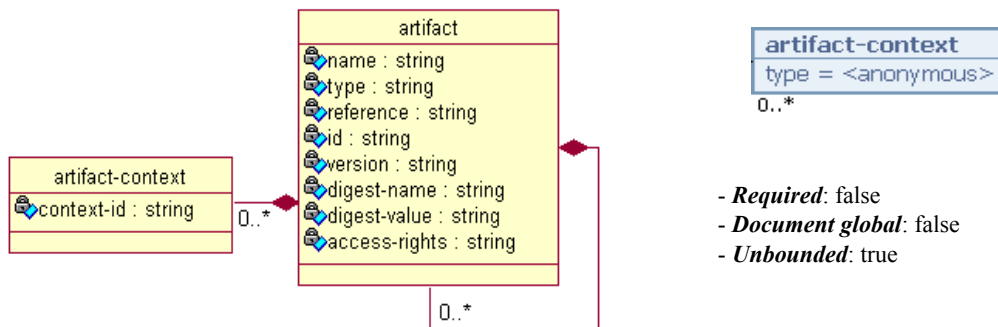
- *Required*: false
- *Document global*: false
- *Unbounded*: true

Table 14 - (continued)

UML Model for MOF 2.0 XMI	XML Schema
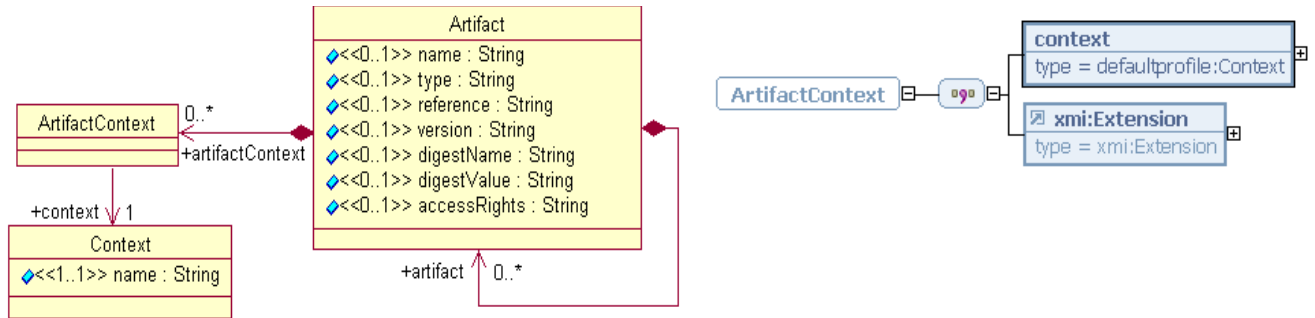
7.4.10.3 ArtifactDependency

An ArtifactDependency identifies a dependent Artifact. The dependent Artifact must be another Artifact defined in the manifest. See the Artifact description. This is a child element to the Artifact element. The *dependencyType* attribute describes artifact dependencies such as design time dependency or a compile time or runtime.

Table 15 - Core RAS::ArtifactDependency Class

ArtifactDependency	
UML Model for XML Schema	**XML Schema**

- *Required*: false
- *Document global*: false
- *Unbounded*: true

Table 15 - (continued)

UML Model for MOF 2.0 XMI	XML Schema
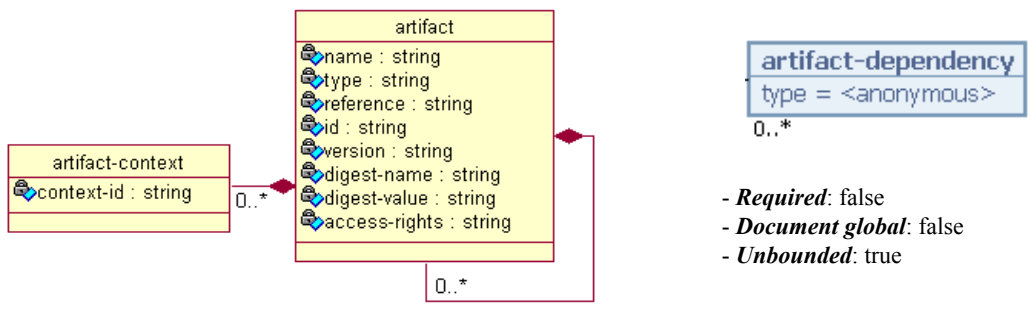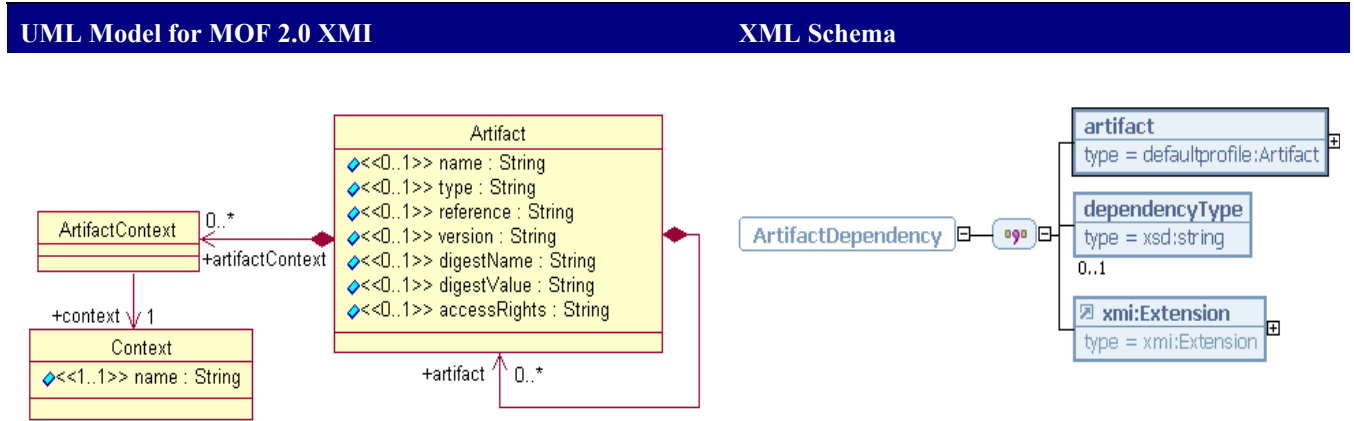
7.4.10.4 VariabilityPoint

Each <variability-point> identifies a location in the artifact that is expected to be modified when applied. The element requires a *name* and *id* attribute. The name should be descriptive of the nature of the customization that will be applied to the artifact. The *id* is referenced by other elements in the manifest (see <variability-point-binding> element description). A <variability-point> can be optionally associated with a context through the ***context-id*** attribute. The <variability-point> element's free-form text is used to capture a fuller description of the activity, which should be represented in plain text. The optional *reference* attribute points to an external document that could further explain the variability point.

Table 16 - Core RAS::VariabilityPoint Class

VariabilityPoint	
UML Model for XML Schema	**XML Schema**

- ***Required***: false
- ***Document global***: true
- ***Multi-line text***
- ***Unbounded***: true

Table 16 - (continued)

UML Model for MOF 2.0 XMI	XML Schema
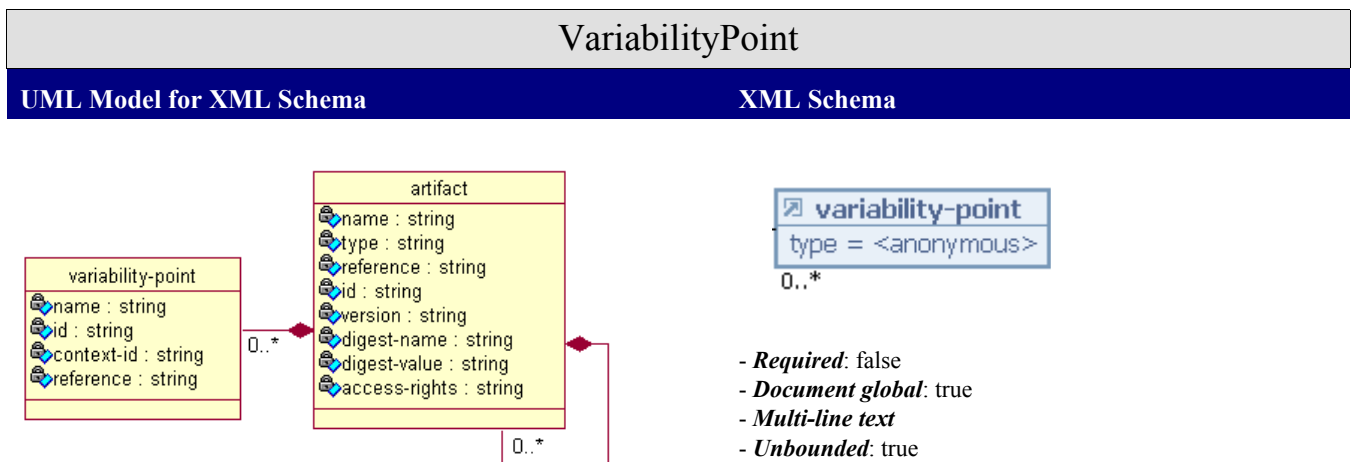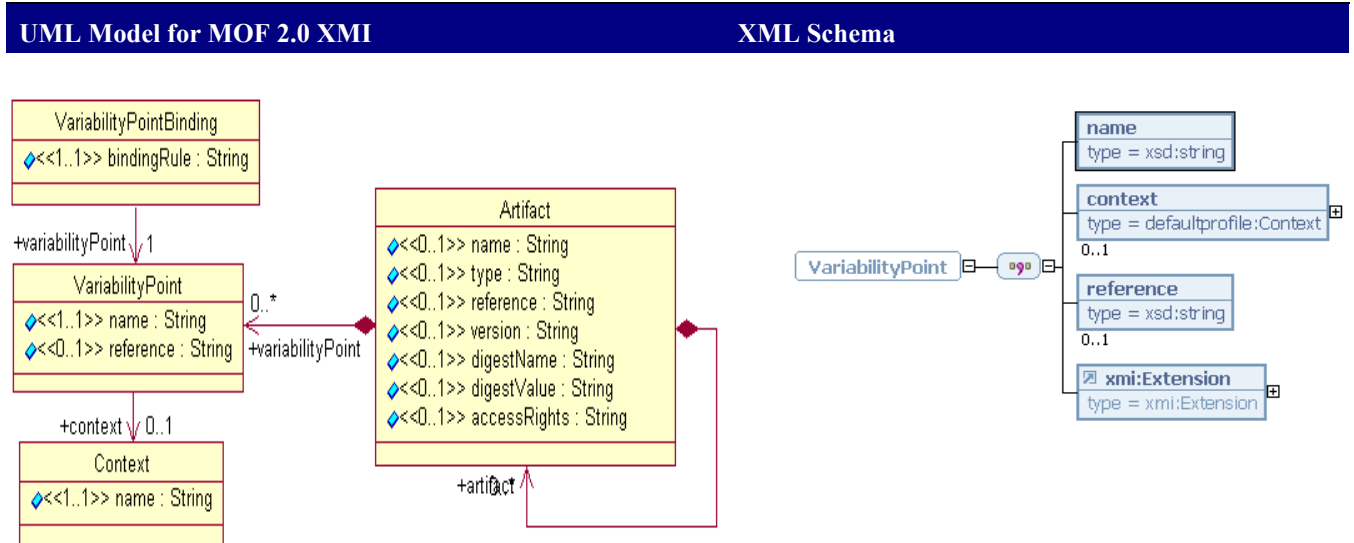
7.4.10.5 ArtifactType

Multiple types may describe an <artifact>. There are two artifact types we describe here, the primary type and the secondary type. Based on the values of the primary type the tooling should perform the main actions that will occur to the artifact when the asset is reused/imported/browsed. Whereas the secondary type is mainly for description purposes and/or secondary actions that could happen when the asset is being reused/imported/browsed.

The <artifact> element *type* attribute is used to represent the primary type. There can be only one primary type per <artifact>. Whereas there can be many secondary types per <artifact>. The <artifact-type> element with the *type* attribute is used to describe the secondary type.

Secondary Types

If the wrong primary type is applied then the tooling may not perform any special processing for that type when the asset is imported. Tool vendors are required to do processing on the primary type list (see Constraint 12) but there is no special processing required on the secondary type list.

The secondary type list includes the primary list and adds types that are mainly for description purposes. A sample list of these secondary types, taken from the file `RASSecondaryArtifactTypes.xml` is below.

```
<artifact id="usecase" type="Use Case"/>
<artifact id="testcase" type="Test Case"/>
<artifact id="reqmodel" type="Analysis Model"/>
<artifact id="designmodel" type="Design Model"/>
<artifact id="implmodel" type="Implementation Model"/>
<artifact id="testmodel" type="Test Model"/>
<artifact id="busncnptmodel" type="Business Concept Model"/>
<artifact id="usecasemodel" type="Use Case Model"/>
<artifact id="busntypemodel" type="Business Type Model"/>
```

```xml
<artifact id="intfacespecmodel" type="Interface Spec Model"/>
<artifact id="websvcintermodel" type="Web Service Interactions Model"/>
<artifact id="analysisset " type="Analysis Artifact Set"/>
<artifact id="designset " type="Design Artifact Set"/>
<artifact id="implset " type="Implementation Artifact Set"/>
<artifact id="testset " type="Test Artifact Set"/>
<artifact id="implset " type="Implementation Artifact Set"/>
<artifact id="testset " type="Test Artifact Set"/>
<artifact id="intfacespecdiag" type="Interface Spec Diagram"/>
<artifact id="usecasediag" type="Use Case Diagram"/>
<artifact id="compinterdiag" type="Component Interaction Diagram"/>
```

This list is dynamic and should be used by tool builders to provide the proper processing of artifacts.

Table 17 - Core RAS::ArtifactType Class
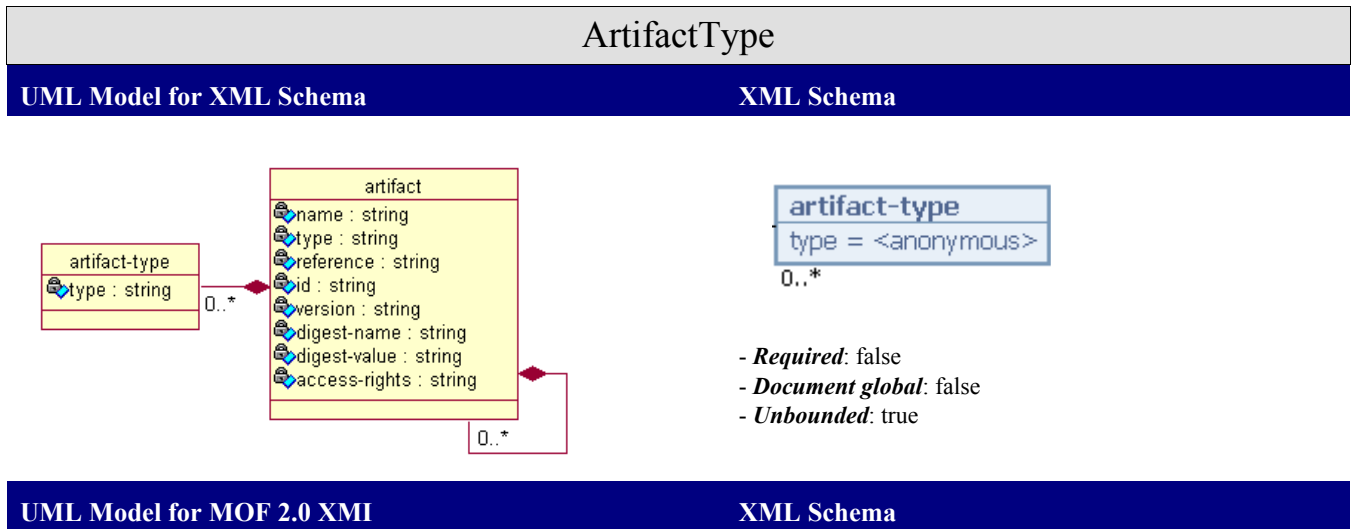
7.4.11 Usage

The usage section describes the activities to be performed for applying or using the asset. This section is described with a lightweight activity or workflow model. There are several forms for conducting activities on an asset. Some activities are for the asset in general whereas other activities are for a specific artifact within the asset, and still other activities may be relevant to a particular context. The model below goes further to describe that an artifact in a particular context may have a variability point that is relevant.
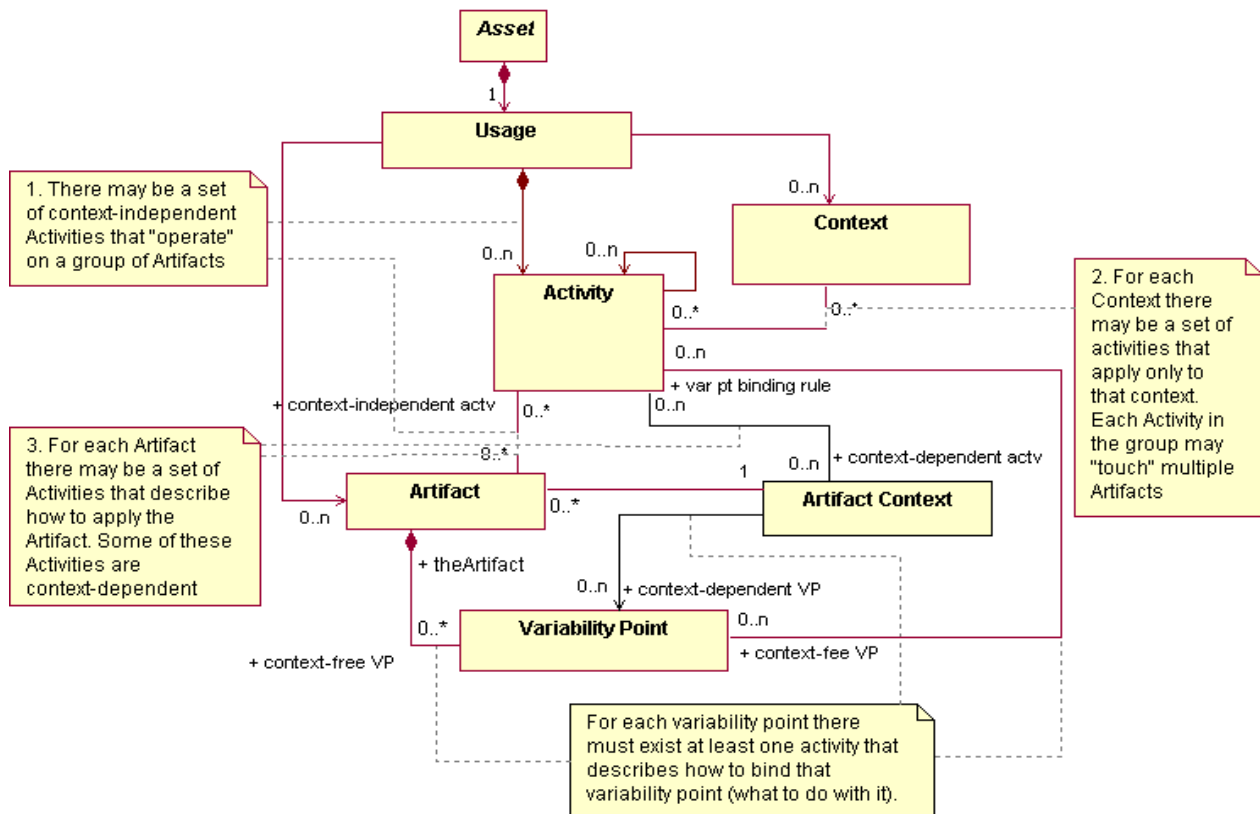
Figure 14 - Usage Section Domain Model

The UML schema model below shows the realization of these asset usage domain concepts outlined above.

The <usage> element is a container element to specify process or usage guidance. The usage element defines only one attribute, *reference*. The *reference* element points to an external document that may clarify the usage section as a whole, or summarize all of the usage activities of the asset.

There are three types of child elements, each of which contains a set of activities that are to be followed when applying this asset to a target application. There may be multiple instances of each of these container child elements, meaning that there may be multiple <artifact-activity>, <context-ref>, and <asset-activity> child elements.

Table 18 - Core RAS::Usage Class

Usage	
UML Model for XML Schema	**XML Schema**
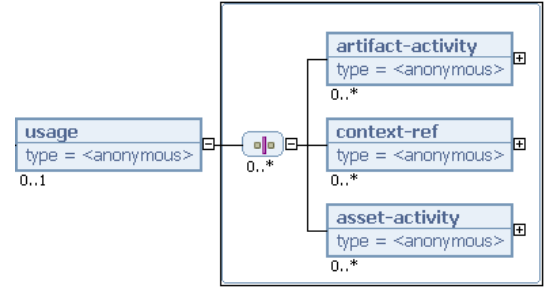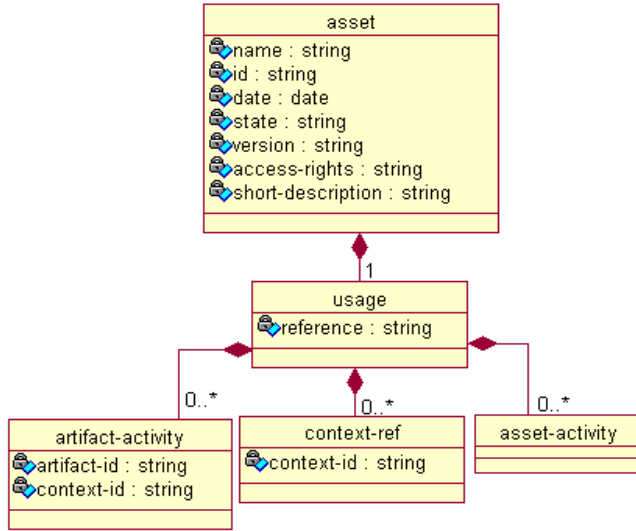
- *Required*: false
- *Document global*: false
- *Unbounded*: false

UML Model for MOF 2.0 XMI	**XML Schema**
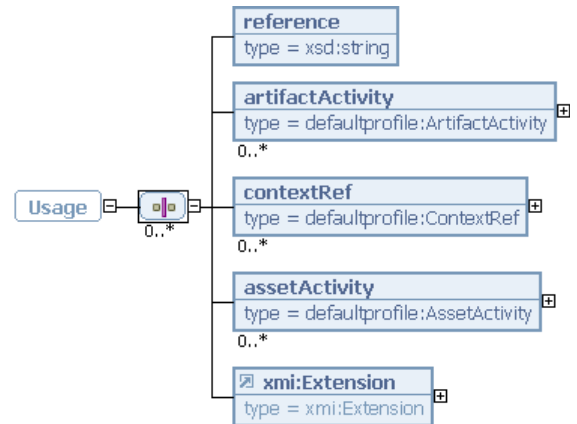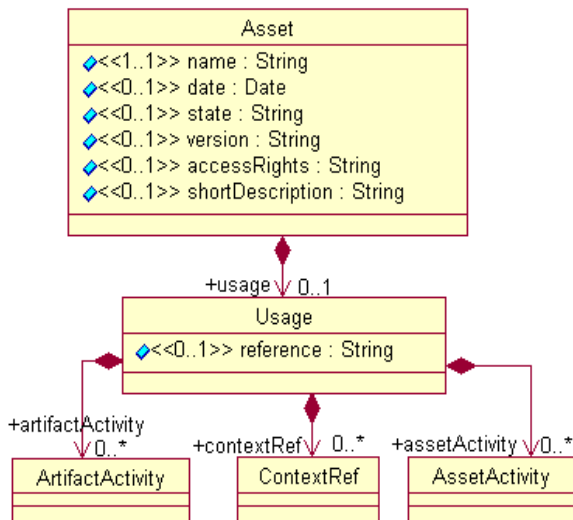
7.4.11.1 ArtifactActivity

The <artifact-activity> element is a container for activities associated with a specific artifact. This element specifies two attributes: *artifact-id* and *context-id*. The *artifact-id* attribute is required and must specify an id associated with an artifact specified in the manifest document. There may be many activities specified for any given artifact. The activities may also be optionally associated with a context. If specified the *context-id* attribute must match an id in a <context> element in the manifest document.

An <artifact-activity> should contain at least one <activity> element, which specifies a concrete activity that the user or tooling should execute when applying the asset to a target application. See <artifact> element description..

Table 19 - Core RAS::ArtifactActivity Class
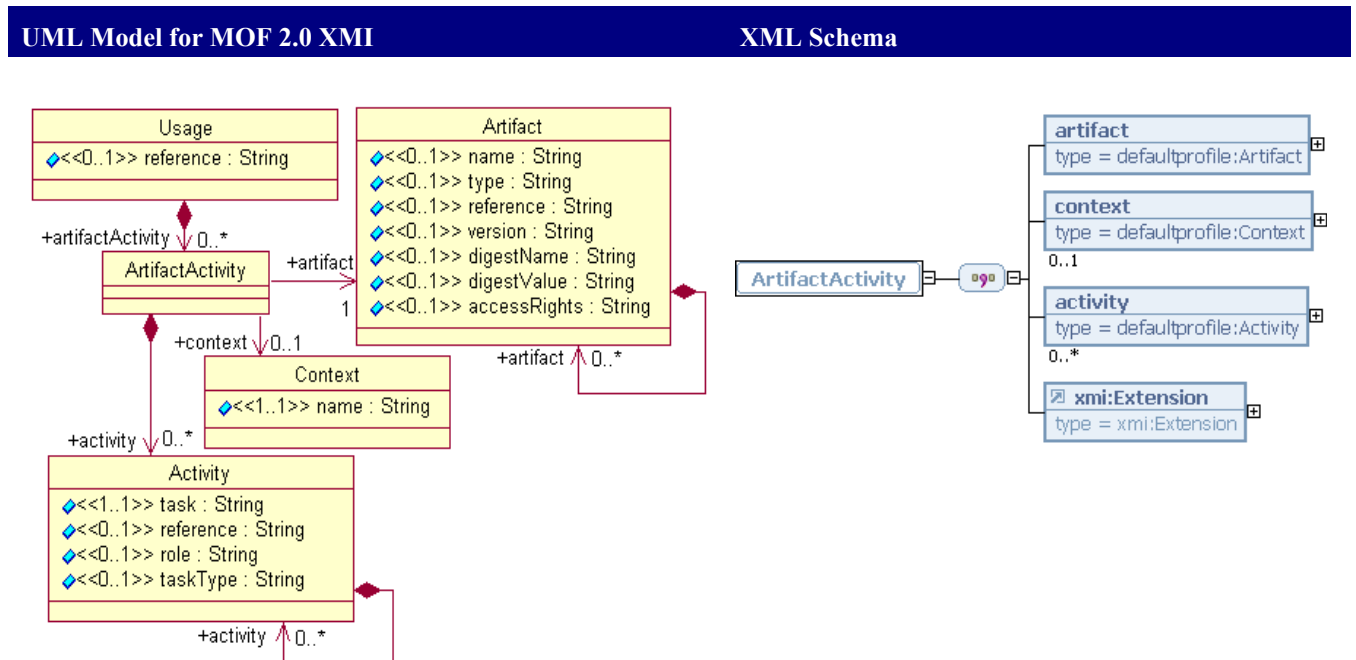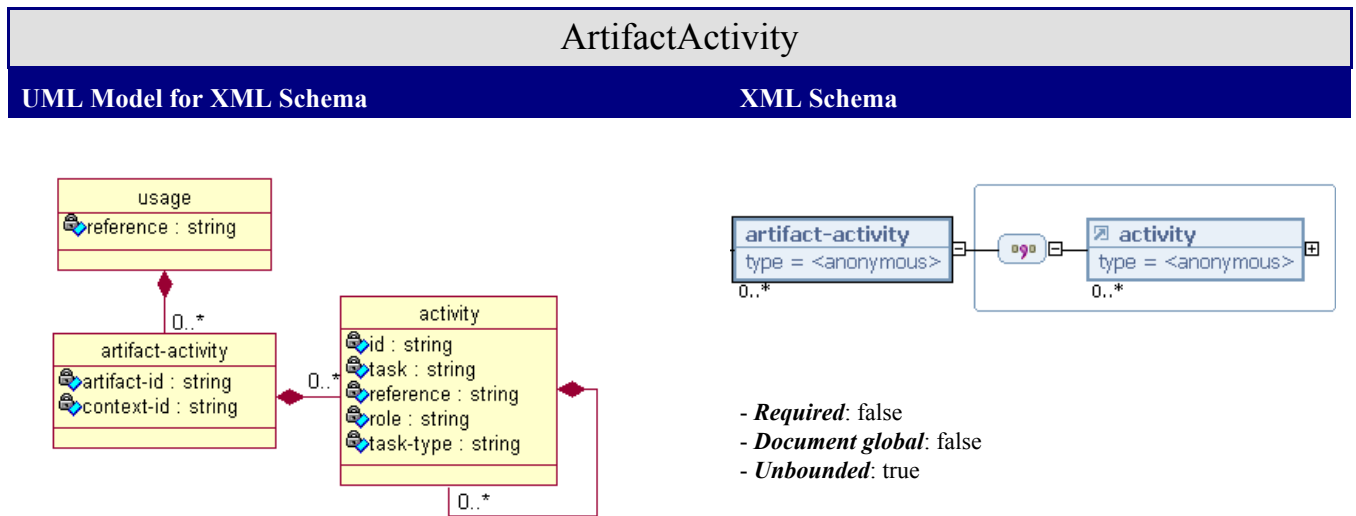
7.4.11.2 ContextRef

The <context-ref> element is a container of activities associated with a specific context. The required *context-id* attribute specifies a context defined elsewhere in the manifest document.

A <context-ref> should contain at least one <activity> element, which specifies a concrete activity that the user or tooling should execute when applying the asset. See <artifact> element description..

Table 20 - Core RAS::ContextRef Class
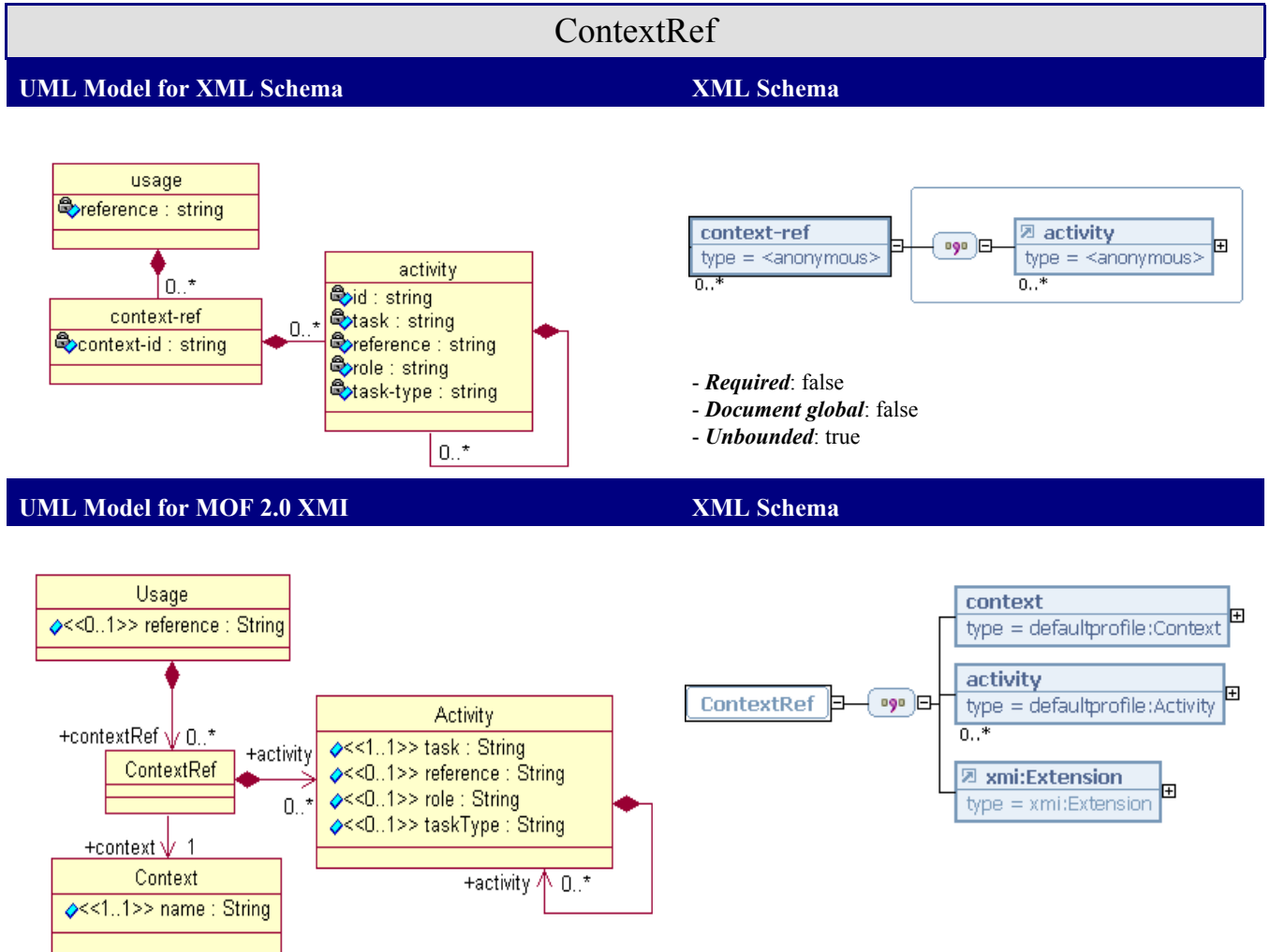
7.4.11.3 AssetActivity

An <asset-activity> element is a container of activities that are associated with the asset as a whole. All of the activities specified in an asset activity are expected to be executed as a group, although not necessarily all at the same time. This element does not define any attributes.

Table 21 - Core RAS::AssetActivity Class

AssetActivity	
UML Model for XML Schema	**XML Schema**

- *Required*: false
- *Document global*: false
- *Unbounded*: true

UML Model for MOF 2.0 XMI	**XML Schema**
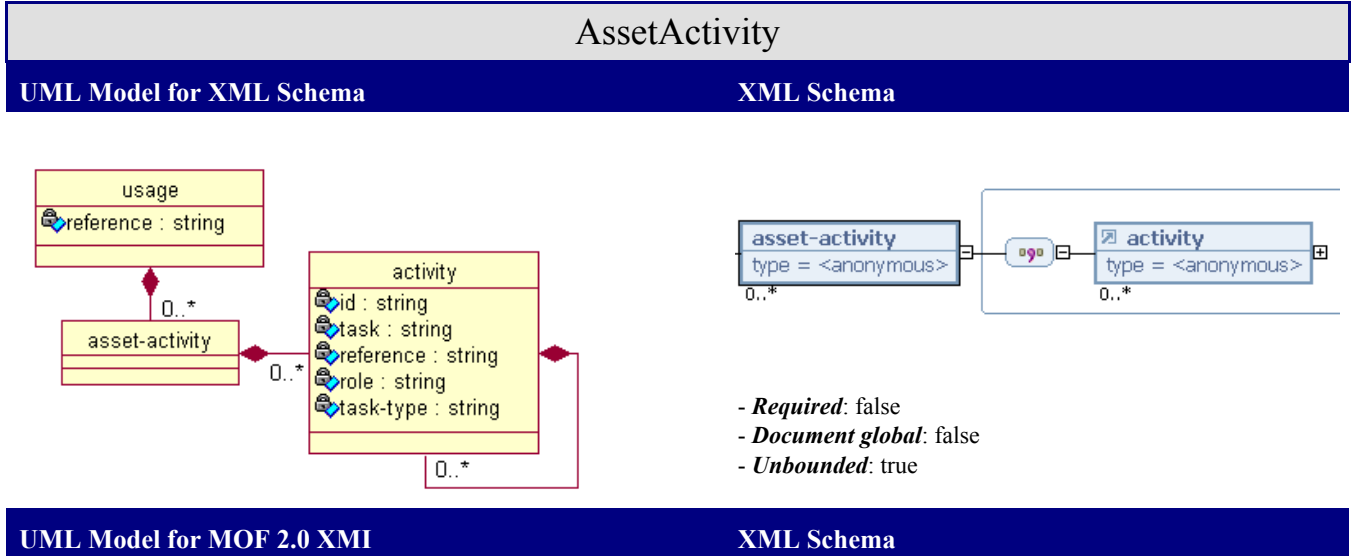
7.4.11.4 Activity

An activity is something that either the consumer or the tooling processing the asset does when applying the asset. An <activity> element specifies two required attributes: *id* and *task*. The *id* attribute should contain a unique identifier across all activities in the manifest file. Although the id attribute is not referenced by other elements in the manifest document, it is included to support tool processing. The *task* attribute is a short description or keyword that represents the activity's purpose or goal. A fuller description of the activity can be found in the <description> child element, or in the external document pointed to by the optional *reference* attribute. The *reference* attribute may also point to a file that contains executable code or scripts that can be used by the tooling.

Some activities may be relevant to certain asset consumer roles, therefore the *role* attribute declares for which, if any, asset consumer role the activity is relevant. The *task-type* attribute is used to categorize the type of activity. This attribute may be used by the tooling and explain how to execute this activity. Some task types may suggest that the tool load and execute a script or run an executable, while others might just indicate that activity should be referenced in the consumer's To Do list.

In addition to the <description> element, an <activity> element may specify child <activity> elements and <variability-point-binding> elements. A <variability-point-binding> is a reference to a defined variability point of an artifact, and specifies a binding rule. The binding rule is a short description describing the relationship between the variability point and the activity..

Table 22 - Core RAS::Activity Class

- *Required*: false
- *Document global*: true
- *Unbounded*: true
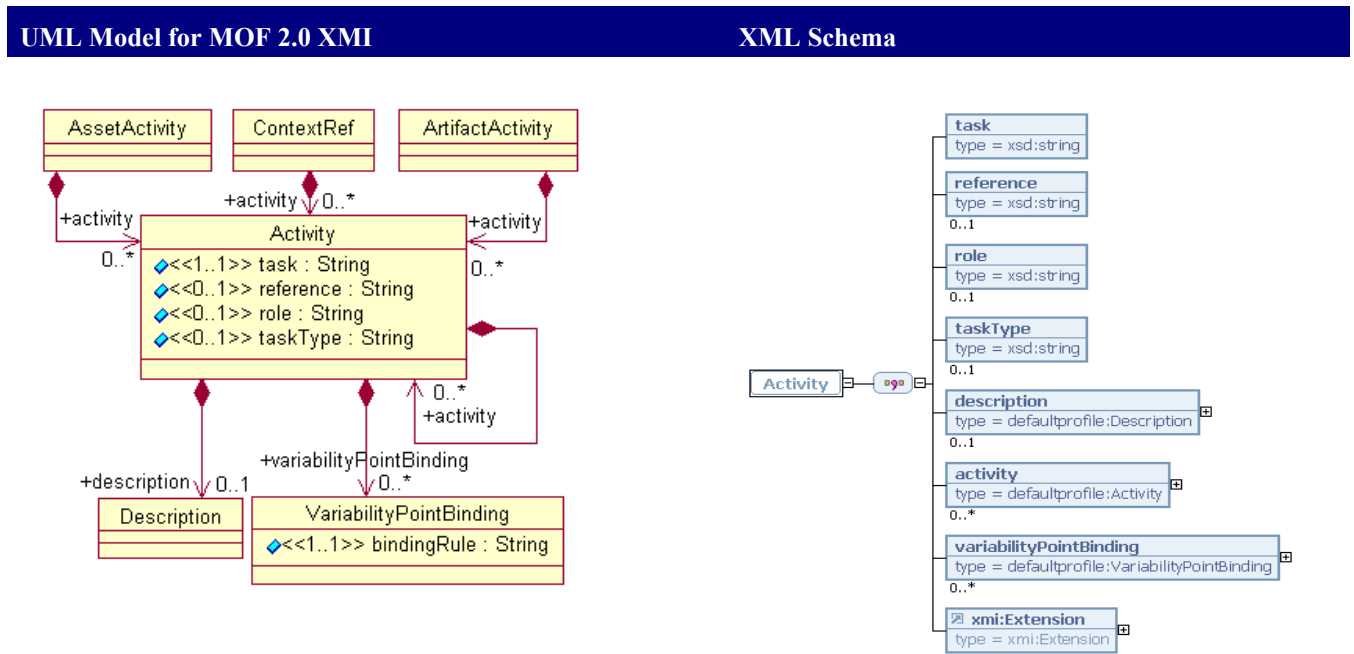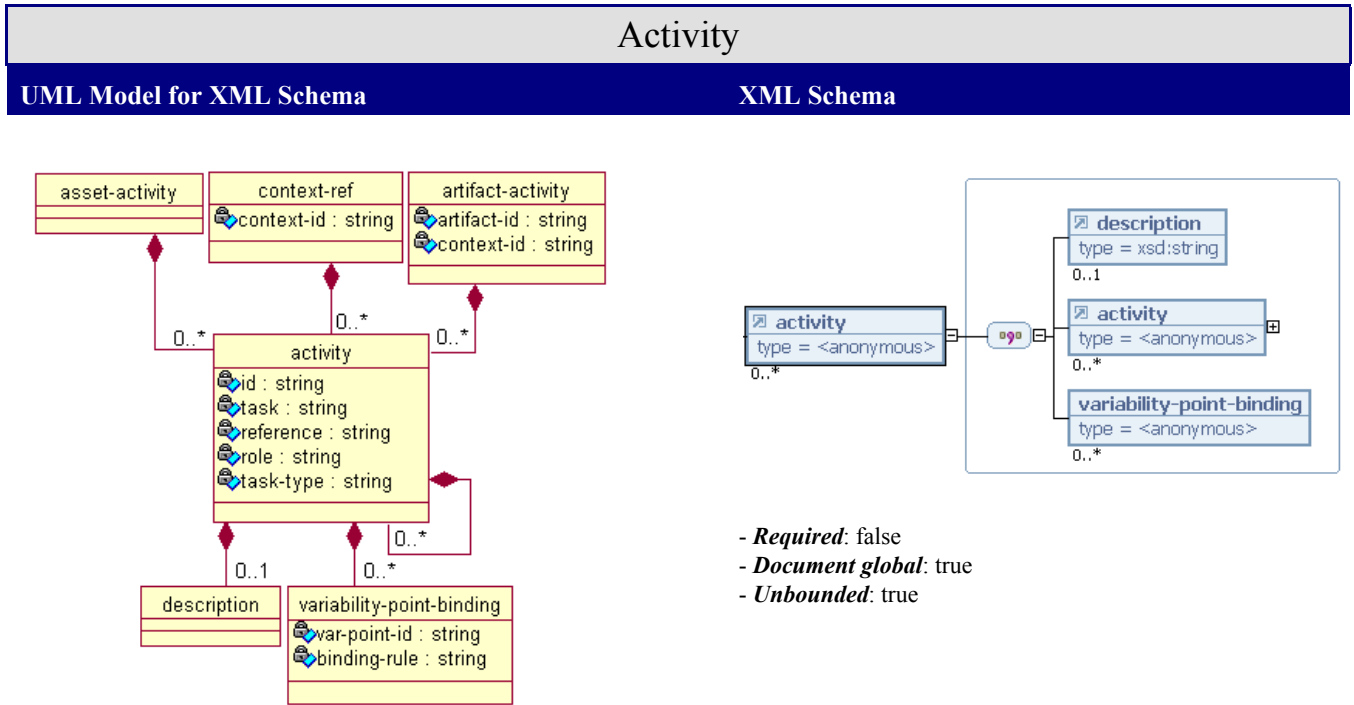
7.4.11.5 VariabilityPointBinding

The ***binding-rule*** attribute is a short description of the nature of the variability point. This attribute provides additional rules and direction that the asset consumer should following when conducting the <activity> on the <variability-point>.

Consider the following partial-grammar example.

<solution>

    <artifact> ***name***: Design Model, ***id***: 100, ***reference***: model/designmodel.mdx

        <variability-point> ***name***: Design Model::User Account Management::Use Case_Create New User Account , ***id***: 1

<usage>

    <artifact-activity> ***artifact-id***: 100

        <activity> ***id***: 5, ***task***: Specify the alternate flows

            <variability-point-binding> ***variability-point-id***: 1, ***binding-rule***: Provide a description of invalid database connection flow.

            <variability-point-binding> ***variability-point-id***: 1, ***binding-rule***: Do not create new relationships to existing packages.

In this example the <variability-point> identifies the location within the <artifact> where the customization occurs. The <activity> identifies what the asset consumer is expected to do and the <variability-point-binding> describes any rules, constraints, and additional guidance for the asset consumer to perform the customization.

Table 23 - Core RAS::VariabilityPointBinding Class

Table 23 (continued)

UML Model for MOF 2.0 XMI	XML Schema
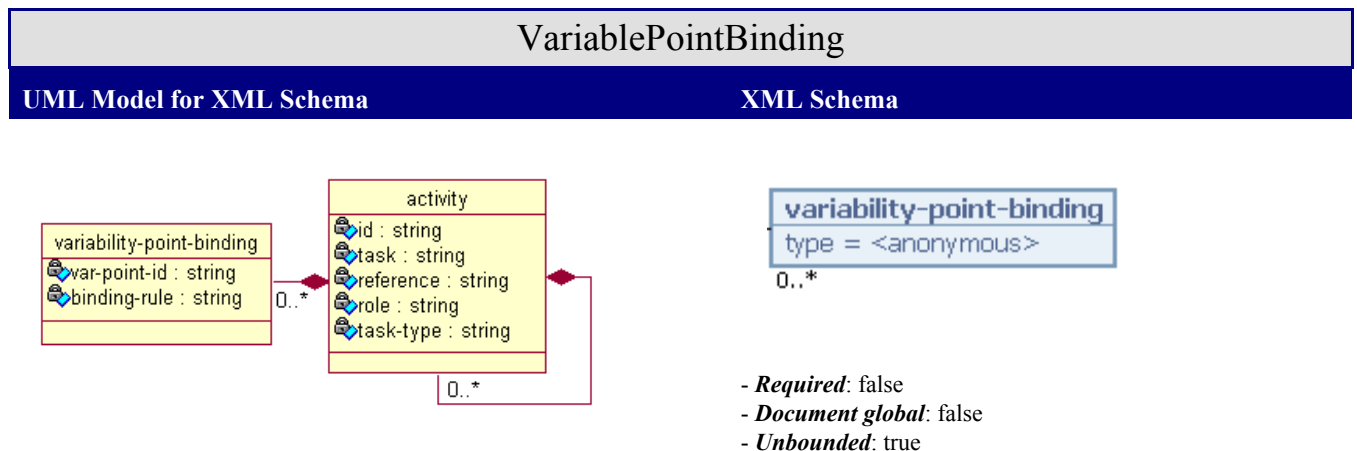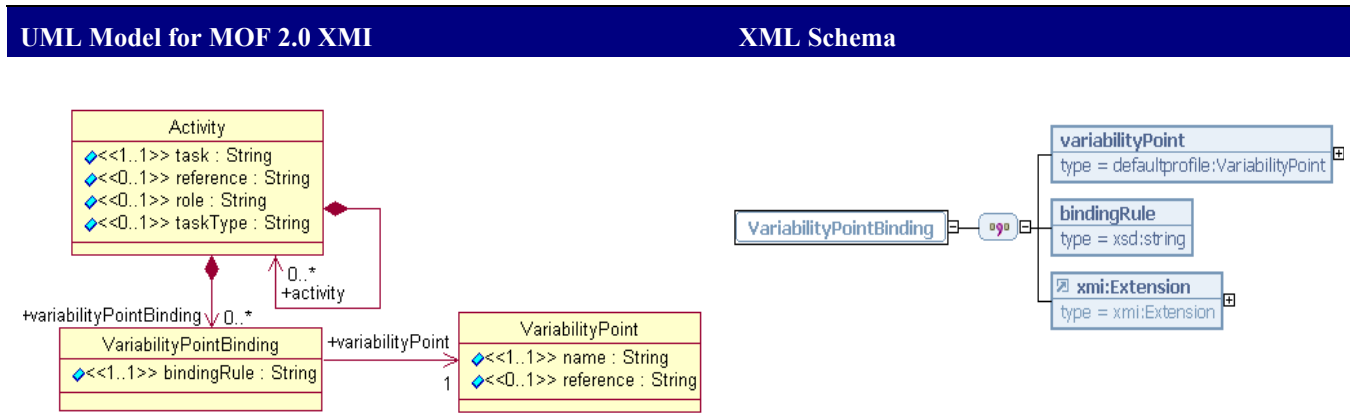
7.4.12 RelatedAsset

Assets rarely exist in isolation; generally there is a relationship to another asset. However, this relationship may not be exposed in the asset's packaging. To the contrary we recommend that this information is included when packaging an asset as the context it describes can help to reduce the reuse costs.

These general related-asset principles are refined in the UML schema model below.

An asset may specify an arbitrary number of related assets. Each related asset is specified with a <related-asset> element, which is a child element of the <asset> element. A related asset may be asset outside the scope of the current asset.

The <related-asset> element is necessary to scale asset reuse to larger-grained or coarse-grained assets wherein a family of assets or asset assemblies can be defined and reused at that level.

The *name* attribute contains the name of the related asset, such as Credit Card web service.

The *relationship-type* attribute may contain any value. However, for certain types of relationships there are reserved values that should be used. These relationships and their reserved values are described below:

- *aggregation*: this indicates that the current asset 'contains' the related asset, this containment may be by value or by reference.
- *similar*: this indicates that the other asset has characteristics which are similar to the current asset.
- *dependency*: this indicates that the current asset references or relies on the services or artifacts of the related asset.
- *parent*: this indicates that the current asset is contained or owned by the related asset.

Asset packages that contain multiple assets can use aggregation and parent relationship types to help structure the contained assets.

The *asset-id* attribute contains the asset id from the related asset's manifest document.

The *reference* attribute contains a location of the related asset, such as: http://companyintranet/RASRepositoryService/ RASRepositoryService.asmx, *repository logical path*: webservices/creditcardservice.ras, and so on. This attribute may also contain reference to a document, which describes the related asset.

Table 24 - Core RAS::RelatedAsset Class
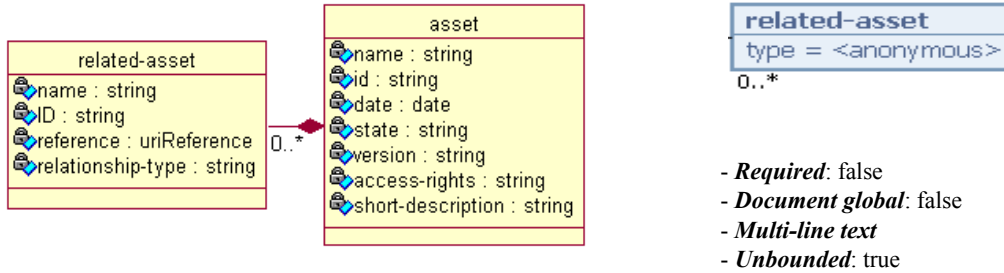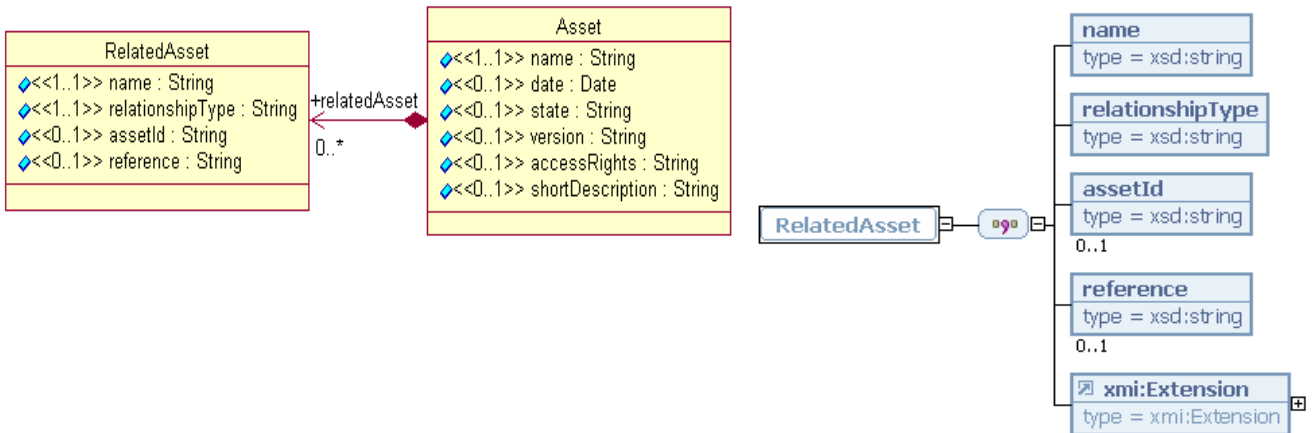
7.4.13 Asset Identity

A reusable asset's identity is tied to directly its manifest. A completed manifest defines an asset by associating a name as well as other meta information to a collection of files that provide a solution to a recurring software development problem.

An asset's version is captured as a string, and therefore can be anything. However it is recommended that a consistent numbering system be used to express the version. Any system can be used (incremental, date, etc.). Regardless of the mechanism used, it should be easy for the consumer to compare two different version strings for the same asset name/id and easily determine which is the more recent, and which is the oldest. Using internal project names like "Chicago" or "Phoenix" are not good version identifiers since it is not clear which is the most recent version.

An asset's id remains constant with each new asset version. The id is a unique identifier, such as a globally unique identifier (GUID). This specification however does not dictate how an asset id is constructed, and therefore there is no absolute guarantee that any two different assets will not have the same id, and it is therefore advised not to rely on ids being unique.

Subsequent versions of an asset may change its name, short description or any other piece of meta information except for the asset id. When the asset id is changed it is considered a completely new asset. It is suggested that when an asset evolves to the point where it is assigned a new identity (i.e. new id value), that the originating asset be referenced in the <related-asset> section of the manifest.

7.4.14 Core RAS Semantic Constraints

There are several constraints that must be enforced as one element of achieving <u>RAS compliance</u>.

Semantic constraints are rules for the manifest's content that are not expressible with standard XML Schemas. The following constraints combined with the XML Schema fully define a valid RAS manifest file and are intended for the XML Schema and may not apply to the MOF/XMI XML Schema. Additional semantic constraints may be defined by profiles.

Constraint 1: The manifest file must validate against the XML Schema associated with the profile and must be referenced by the manifest file.

Constraint 2: An asset must have within the solution element at least one artifact element with a reference attribute that is non-empty and a non-empty ***name*** attribute.

Constraint 3: A file in an asset must be associated with at most one <artifact> element.

Constraint 4: The ***context-id*** attribute in the <artifact-context>, <descriptor>, <artifact-dependency>, <variability-point>, <context-ref> and <artifact-activity> element must specify an ***id*** from a context element found in the same manifest document.

Constraint 5: The ***artifact-id*** attribute in the <artifact-activity>, and <artifact-dependency> elements must specify an id from an <artifact> element found in the same manifest document.

Constraint 6: The ***variability-point-id*** attribute of the <variability-point-binding> element must specify an ***id*** from a <variability-point> element found in the same manifest document.

Constraint 7: If the ***asset-id*** attribute of the <related-asset> element is used it must specify the ***id*** attribute of the <asset> element in separate manifest document.

Constraint 8: The <related-asset> element ***relationship-type*** attribute may contain any values. However, for certain types of relationships there are reserved values that should be used. These relationships are described below:

- *aggregation*: this indicates that the current asset 'contains' the related asset
- *similar*: this indicates that the other asset has characteristics which are similar to the current asset
- *dependency*: this indicates that the current asset references or relies on the services or artifacts of the related asset
- *parent*: this indicates that the current asset is contained or owned by the related asset

*Constraint 9*: The ***id-history*** attribute in the <profile> element must contain a concatenated value of profile ids illustrating the ancestry of the profile. The ids must be delimited with two successive colons.

In this example the concatenated ids are Microsoft GUIDs, using style D[1] . This profile does not constrain the actual value types that may be used for ids. Rather, the ids must be unique within the intended reuse scope of the asset profile.

> If we are looking at a profile with the following id-history value "a::b::c", then the id for the current profile is "a", its parent id is "b", and its grandparent is "c".

Constraint 10: A manifest file cannot reference itself in an <artifact> element. This would cause confusion between meta information and information in an asset.

Constraint 11: The **artifact-id** attribute on an <artifact-dependency> element and <artifact-activity> element must use an **id** from an <artifact> element in the same document.

Constraint 12: The **type** attribute value on an <artifact> element must use a <u>primary type</u> value. <u>Secondary type</u> values must be handled through <artifact-type> element. The primary and secondary type lists are dynamic. Tool vendors should provide a mechanism to manage these lists.

Constraint 13: Tool vendors must provide processing for at least one primary **artifact** type. This means that tool vendors must recognize the value in the **type** attribute and process it appropriately within the context of their own tooling. The other primary types may be generically handled. Whereas tooling vendor support for all secondary **artifact** types are considered optional.

Constraint 14: Each <artifact> element is part of a <context> element known as the asset's root context. However, this context is implied and does not need to be captured for each <artifact>.

Constraint 15: A RAS profile can be created to introduce tighter semantics and constraints. For example, a new profile may make current optional elements to be required. But the constraints in the parent profiles cannot be removed. For instance, existing elements cannot be made less constrained in the new profile than how they are defined in parent profiles.

Constraint 16: Attributes on existing nodes can be added in new RAS profiles. However, the constraints on existing attributes cannot be reduced. For example, a new profile may make current optional attributes to be required. But the constraints in the parent profiles cannot be removed. Existing attributes cannot be made less constrained in the new profile than how they are defined in parent profiles.

7.5   Default Profile 2.1

This version of the Default profile is a realization of the Core RAS. We maintain the Core RAS and the Default profile, as separate entities due partly because the Core RAS may migrate over time and its realization in the Default profile may not be synchronized from a timing perspective. Also, these are separate entities because the realization of the Core RAS may require some implementation details that the Core RAS does not specify. Customized profiles should extend from the Default profile or perhaps one of the other profiles in this document such as the Default Web Service profile or the Default Component profile.

---

1. For more information on the Format Provider Specifier values (N, D, B, and P) refer to the .NET Framework Class Library documentation for the Guid.ToString(String, IFormatProvider) function.

7.5.1 Default Profile History

The XML Schema for the Default profile has an <xsd:annotation> element. This annotation contains the profile history of this schema. This uses <xsd:appinfo> to describe the profile history which means the it can be machine readable. Note: this information does not appear in a manifest document (e.g., rasset.xml) but rather resides in the schema file.

The Default profile history is shown below, as taken from the XML schema file; again, this information only resides in the XML schema file. Therefore tool vendors need to open the XML schema file, retrieve this information, and populate the <profile> element and children elements in the manifest document (e.g., rasset.xml) as necessary.

<xsd:appinfo xmlns:rasprofile="http://www.rational.com/ras/rasdefaultprofile2_0" source="profile-history">

      <rasprofile:profile name="Default" id="F1C842AD-CE85-4261-ACA7-178C457018A1::31E5BFBF-B16E-4253-8037-98D70D07F35F" version-major="2" version-minor="01" parent="F1C842AD-CE85-4261-ACA7-178C457018A1">

         <rasprofile:description>This is the second major version of the default profile. This profile can be accepted by XDE release 2 and later.</rasprofile:description>

         <rasprofile:related-profile name="Core" id="F1C842AD-CE85-4261-ACA7-178C457018A1" version-major="1" version-minor="0" parent="">The original base of Core RAS.</rasprofile:related-profile>

      </rasprofile:profile>

</xsd:appinfo>

7.5.2 New Element Summary

Other than the updated profile history, as described above, the Default profile reflects the Core RAS as described in the sections above. The Default profile is expressed in an XML Schema file that accompanies this document.

7.5.3 Required Elements

This profile uses all required elements as specified by the Core RAS and adds no new elements.

7.5.4 Required Attributes

This profile uses all required attributes as specified by the Core RAS and adds no new attributes.

7.5.5 Semantic Constraints

There are no additional semantic constraints on this profile. The semantic constraints of Core RAS apply to this profile.

7.5.6 RAS Compliance

An asset based on this profile is RAS compliant if all of the following conditions are true:

The RAS Compliance of the Core RAS is preserved.

7.6 Default Component Profile 1.1

This profile leverages many principles and concepts described in John Cheesman's book UML Components. Specifically this profile can support a collection of models and diagrams to describe the component as outlined on page 41 in Cheesman's book.

7.6.1 Default Component Profile History

This profile derives from the RAS Default Profile, version 2.1.

The XML Schema for the Default Component profile has an <xsd:annotation> element. This annotation contains the profile history of this schema. This uses <xsd:appinfo> to describe the profile history which means the it can be machine readable. Note: this information does not appear in a manifest document (e.g., rasset.xml) but rather resides in the schema file.

The Default Component profile history is shown below, as taken from the XML schema file; again, this information only resides in the XML schema file. Therefore tool vendors need to open the XML schema file, retrieve this information, and populate the <profile> element and children elements in the manifest document (e.g., rasset.xml) as necessary.

<xsd:appinfo xmlns:rasprofile="http://www.rational.com/ras/rascomponentprofile1_11" source="profile-history">

    <rasprofile:profile name="Default Component" id="F1C842AD-CE85-4261-ACA7-178C457018A1::31E5BFBF-B16E-4253-8037-98D70D07F35F::1025A790-78D4-4f57-94CE-E65B23275FCD" version-major="1" version-minor="11" parent="31E5BFBF-B16E-4253-8037-98D70D07F35F">

        <rasprofile:description>This is the first major version of the default component profile. This profile can be accepted by XDE release 2 and later.</rasprofile:description>

        <rasprofile:related-profile name="Default" id="31E5BFBF-B16E-4253-8037-98D70D07F35F" version-major="2" version-minor="01" parent="F1C842AD-CE85-4261-ACA7-178C457018A1">This is the second major version of the default profile. This profile can be accepted by XDE release 2 and later.</rasprofile:related-profile>

        <rasprofile:related-profile name="Core" id="F1C842AD-CE85-4261-ACA7-178C457018A1" version-major="1" version-minor="0" parent="">The original base of Core RAS.</rasprofile:related-profile>

    </rasprofile:profile>

</xsd:appinfo>

7.6.2 Required Classes

The semantic constraints section describes the rules for certain elements and should be reviewed. In addition to the required elements in the Default profile, the Default Component profile adds the following required class:

- Operation

7.6.3 Required Attributes

In addition to the required attributes in the Default profile, the Default Component profile adds the following required attributes:

Table 25 - Default Component Profile::UML Model for XML SchemaRequired Attributes

Default Component Profile UML Model for XML Schema				
Required Class	**Required Attribute**		**Optional Class**	**Required Attribute**
operation	name		association-role	name
operation	initiates-transaction		association-role	type
			attribute	name
			attribute	type
			condition	description
			condition	type
			diagram-dependency	diagram-id
			interface-spec	name
			model-dependency	model-id
			parameter	direction
			parameter	name
			parameter	type

Table 26 - Default Component Profile::UML Model for MOF 2.0 XMI Required Attributes

Default Component Profile UML Model for MOF 2.0 XMI				
Required Class	**Required Attribute**		**Optional Class**	**Required Attribute**
Operation	name		AssociationRole	name
Operation	initiatesTransaction		AssociationRole	type
			Attribute	name
			Attribute	type
			Condition	description
			Condition	type
			InterfaceSpec	name
			Parameter	direction
			Parameter	name
			Parameter	type

** This attribute is not formally specified in the model because XMI provides id support by default; these ids in XMI are optional and therefore this table specifies constraints on the id that it is required.

*** This id attribute *DOES* reside in the model and is the profile id for a profile which is an ancestor to the current profile and which is not the <profile> id from the current asset's manifest.

7.6.4   RAS Compliance

An asset based on this profile is RAS compliant if all of the following conditions are true:

1.   The <u>RAS Compliance</u> of the Default profile, version 2.1 is preserved.

2.   The <u>constraints</u> of the Default profile, version 2.1 is preserved.

7.6.5   Solution

Only the new classes for this profile are outlined here. For information on other elements refer to this profile's ancestry, namely the Default profile. The Solution section has four new elements now including Requirements, Design, Implementation, and Test. These sections organize special kinds of Artifacts that improve browsing and navigation of the asset and also specify some required WSDL elements.

The models in this section only show those elements that are new or unique to this profile. The Solution section is the class that is extended for this profile and therefore the UML models illustrate elements from that section.
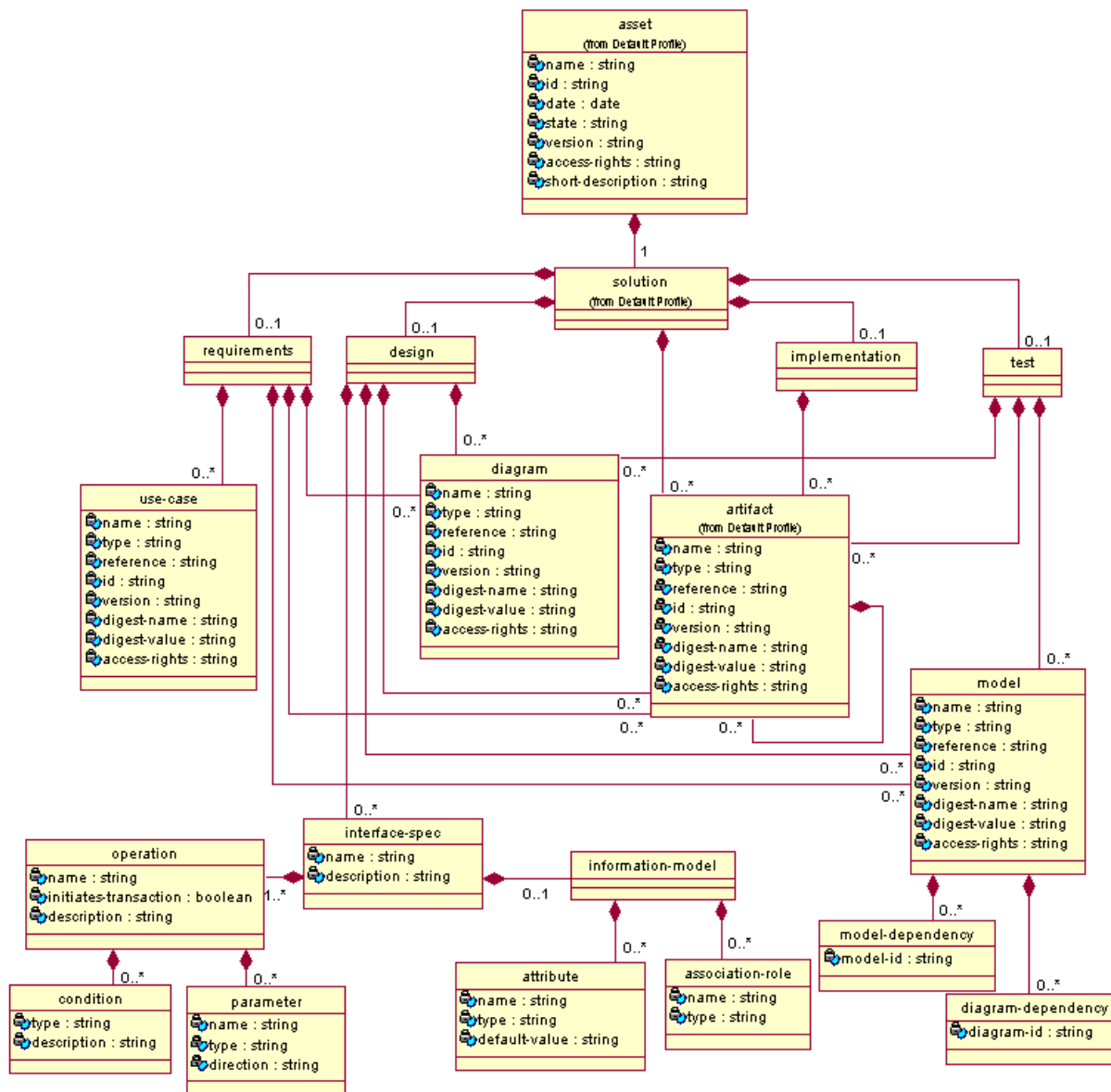
Figure 15 - Default Component Profile UML Model - for XML Schema

Figure 16 - Default Component Profile UML Model - for MOF/XMI XML Schema

7.6.5.1 Requirements

This is a new class, having no attributes, but which has association with several classes including Model, Diagram, UseCase, and Artifact. The models, diagrams, artifacts, and so on within this element are intended to describe the requirements that the component proposes to fulfill. The model, diagram, and artifact nodes are global in the XML schema..

Table 27 - Default Component Profile::Requirements Class

- *Required*: false
- *Document global*: false
- *Unbounded*: false
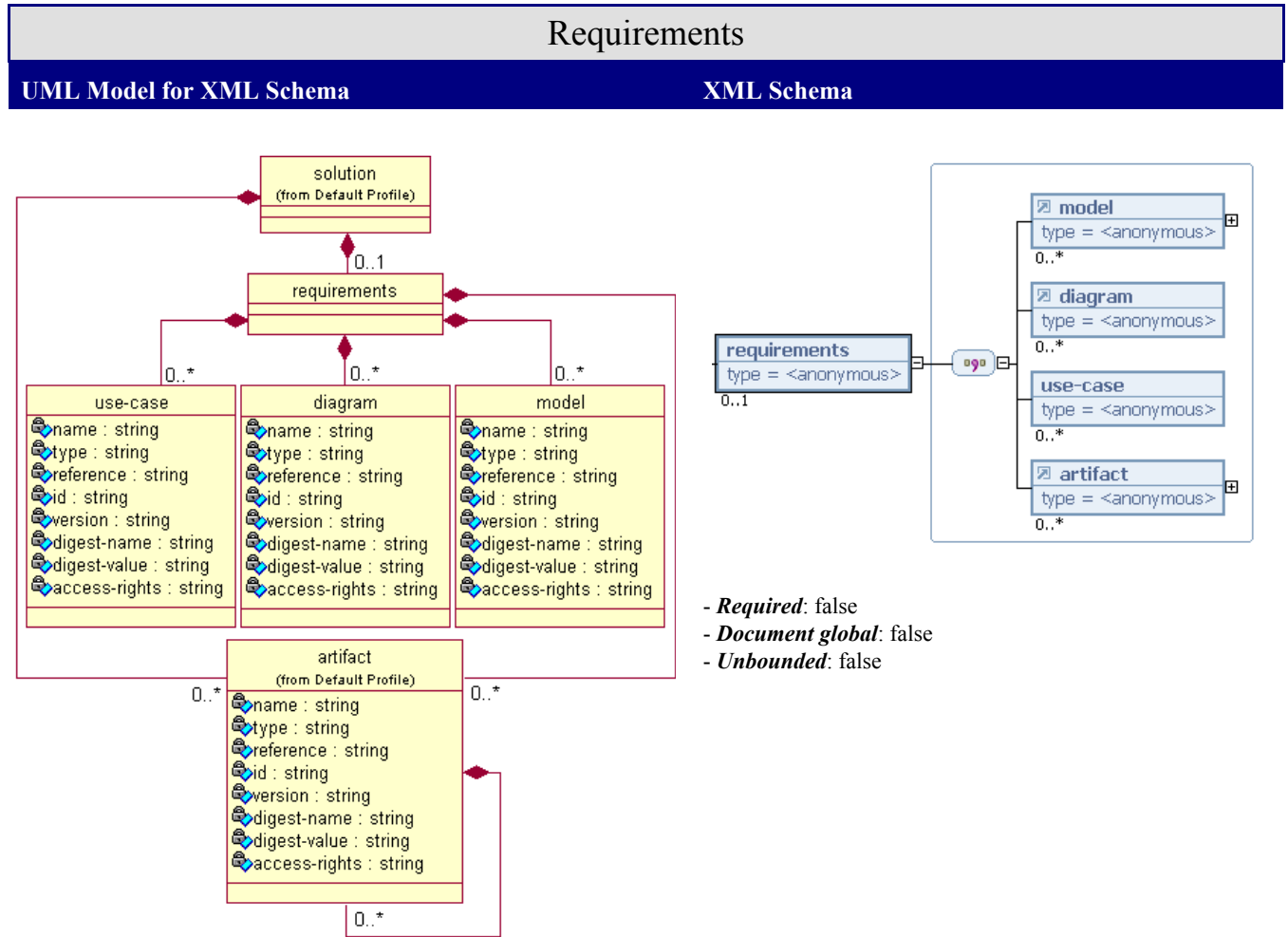
Table 27 (continued)

UML Model for MOF 2.0 XMI	XML Schema
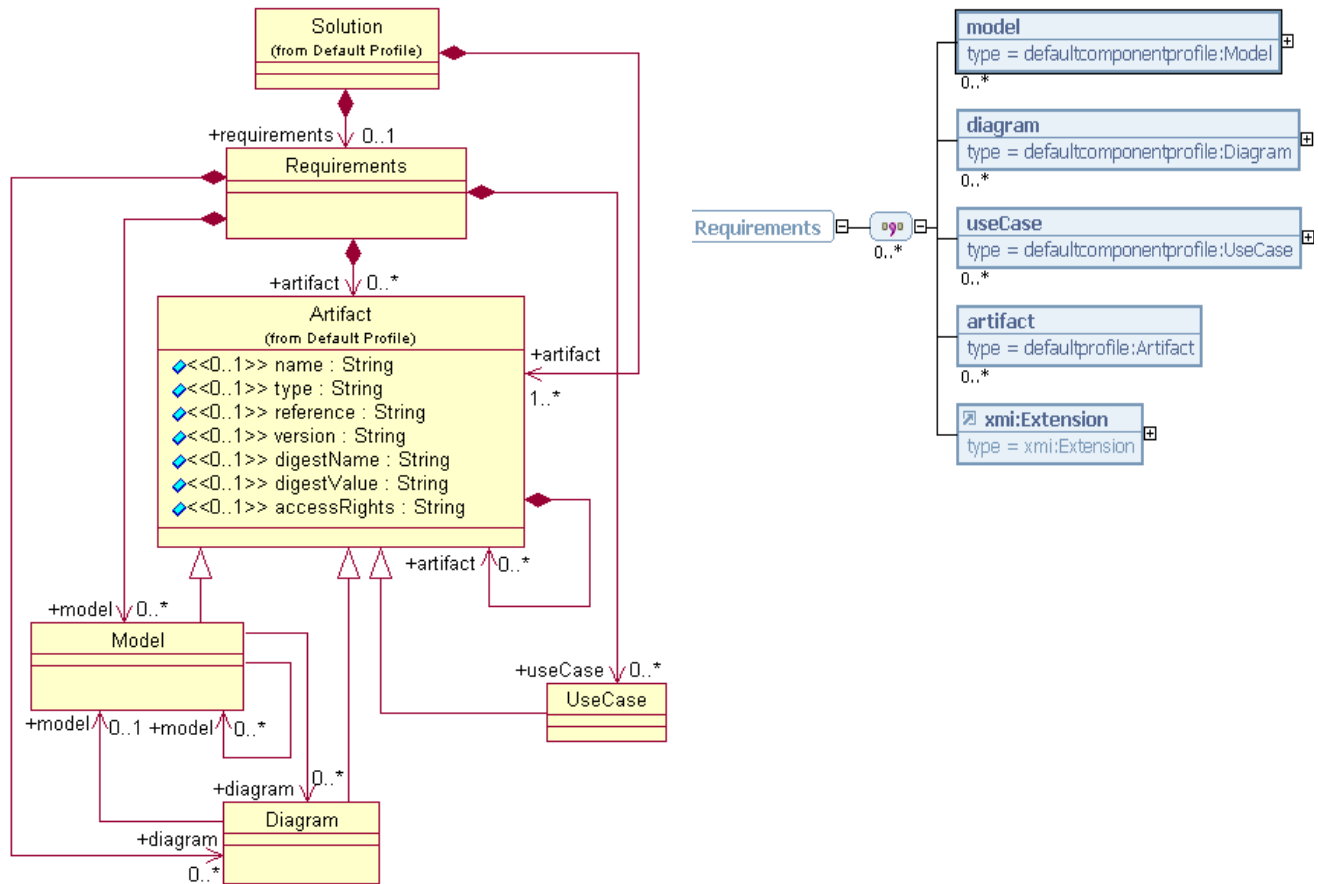
7.6.5.2 Model

This class represents the model for specifying the requirements the component proposes to fulfill. There may be multiple models such as the Business Concept Model and the Use Case Model, see UML Components, page 41.

The attributes are the same as the Artifact attributes; but are constrained to reference models for describing the requirements for the component.

Table 28 - Default Component Profile::Model Class

Model
UML Model for XML Schema / **XML Schema**
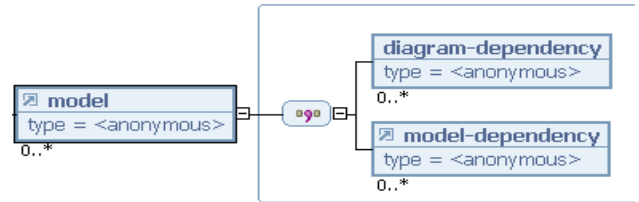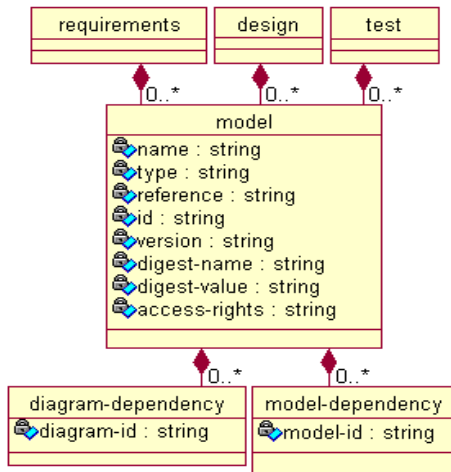
- *Required*: false
- *Document global*: true
- *Unbounded*: true
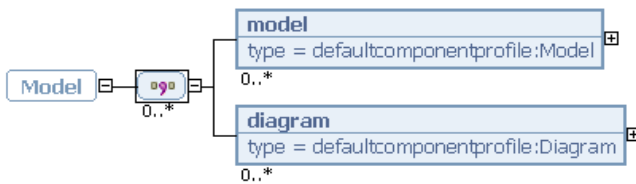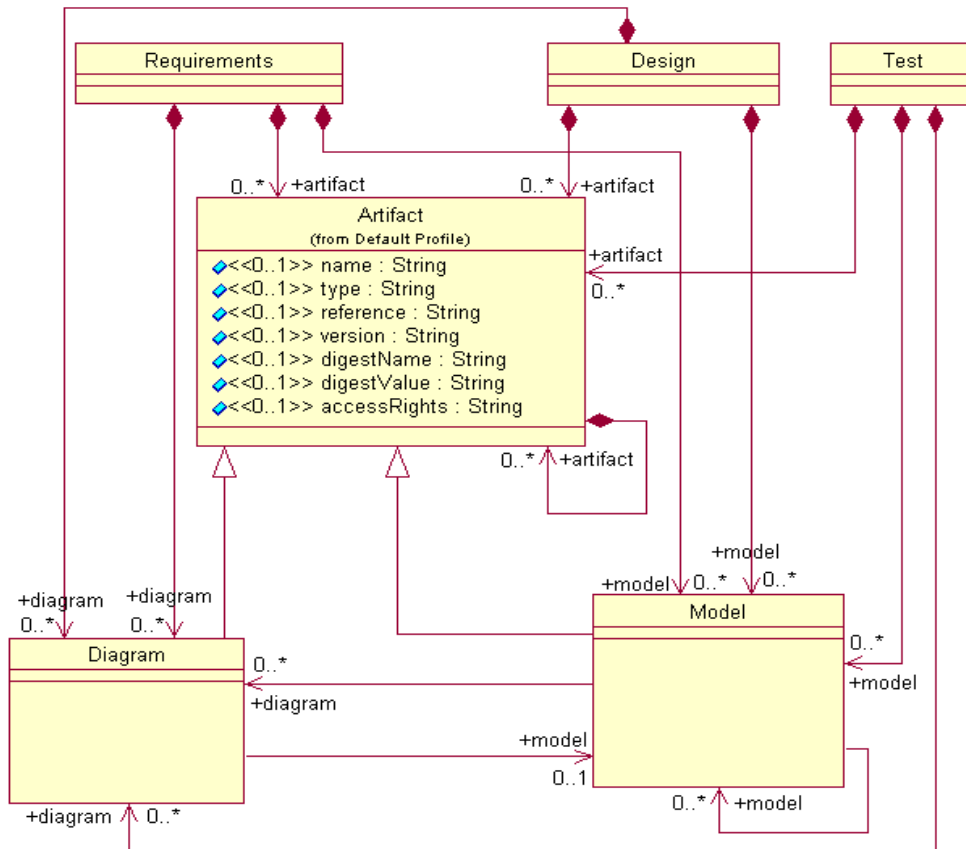
7.6.5.3 DiagramDependency

This class creates a relationship between models and diagrams. This is to help the asset consumer understand all the diagrams for a particular model during asset browsing and evaluation.

The diagram-id attribute should reference a Diagram in the manifest document.

Note that this class is not needed in the MOF / XMI XML schema due to XMI relationships, rather it is handled through the Diagram and Model classes.

Table 29 - Default Component Profile::DiagramDependency Class

7.6.5.4 ModelDependency

This element establishes relationships amongst Models. The asset consumer can be guided through a series of models to understand the component.

The *model-id* attribute on this element contains the id value from a Model in the same manifest file.

Note that this class is not needed in the MOF / XMI XML schema due to XMI relationships, rather it is handled through the Model class.

.

Table 30 - Default Component Profile::ModelDependency Class

ModelDependency	
UML Model for XML Schema	**XML Schema**
	  - *Required*: false - *Document global*: true - *Unbounded*: true
UML Model for MOF 2.0 XMI	**XML Schema**

7.6.5.5 Diagram

A model may have multiple diagrams. For each of the Requirements, Design, and Test classes, the Diagram class identifies the relevant diagrams such as the Business Concept Model diagram and the Use Case diagram, see UML Components, page 41.

The attributes are the same as the Artifact class; but are constrained to reference diagrams for describing the requirements for the component.

Table 31 - Default Component Profile::Diagram Class

Diagram

UML Model for XML Schema — **XML Schema**

- *Required*: false
- *Document global*: true
- *Unbounded*: true

UML Model for MOF 2.0 XMI — **XML Schema**
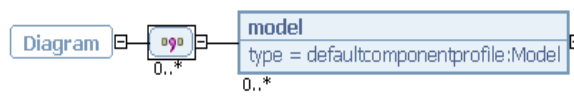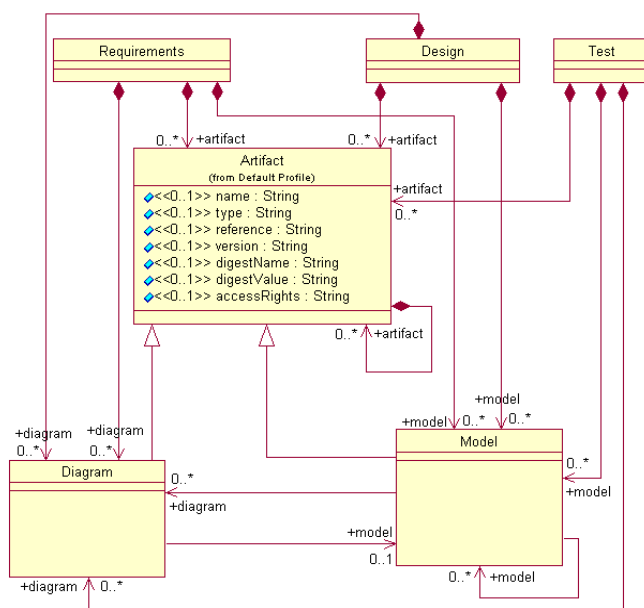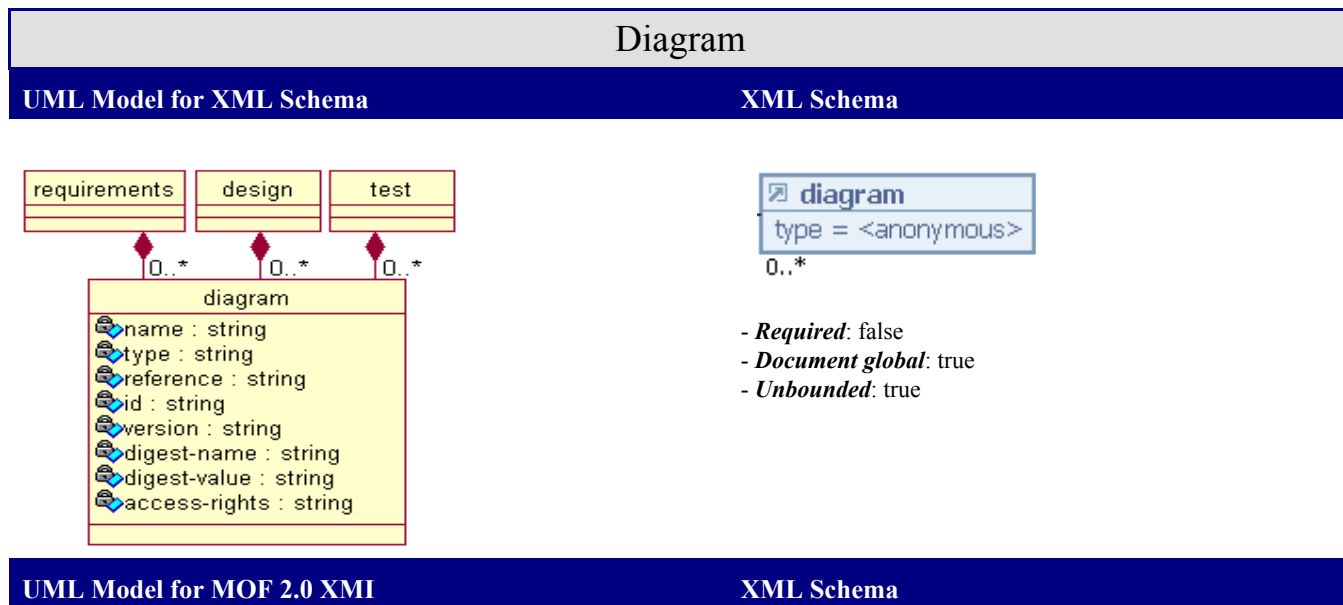
7.6.5.6 UseCase

The component may fulfill one or more use case. This element points to a use case description.

The attributes are the same as the Artifact attributes; but are constrained to reference use case documents for the component.

Table 32 - Default Component Profile::UseCase Class

7.6.5.7 Design

The Design class has no attributes, but has several associations including Model, Diagram, InterfaceSpec, and Artifact. The models, diagrams, artifacts, and so on within this element are intended to describe the design elements that are necessary for the asset consumer to use the component.

Table 33 - Default Component Profile::Design Class

Design	
UML Model for XML Schema	**XML Schema**
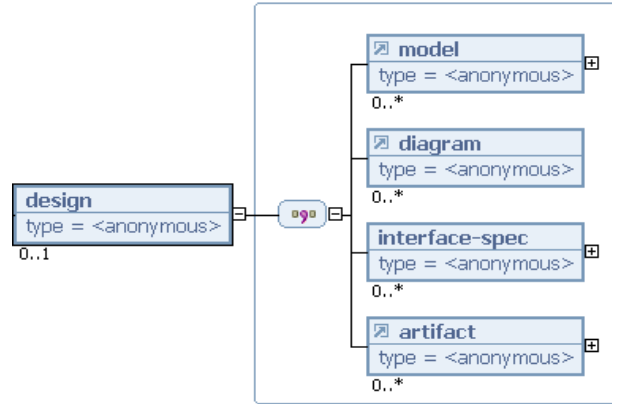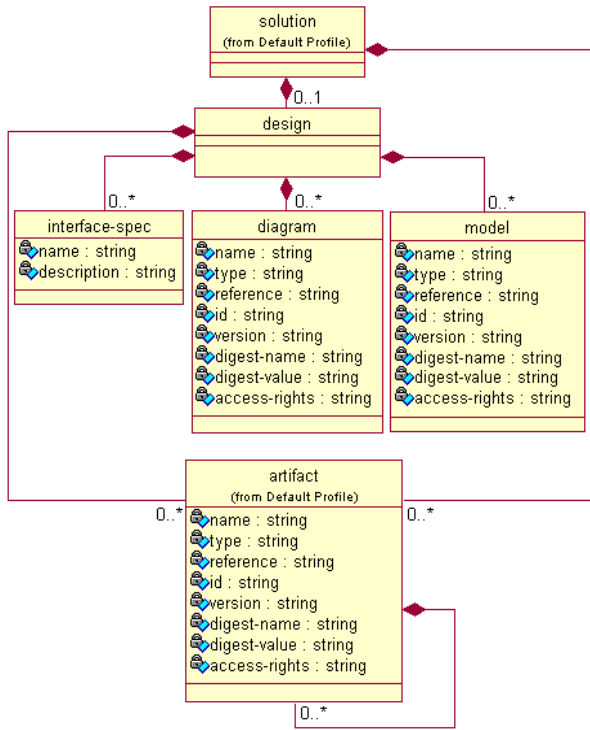
- *Required*: false
- *Document global*: false
- *Unbounded*: false

7.6.5.8   InterfaceSpec

The InterfaceSpec class describes an interface of the component. There may be multiple interfaces defined on the component, so multiple instances of this class may be necessary. The *name* attribute is the user-consumable name of the interface. The *description* is a human consumable short description of the interface. This class has associations with Operation and InformationModel. If you create an InterfaceSpec instance you must create one or more Operations.

.

Table 34 - Default Component Profile::InterfaceSpec Class

InterfaceSpec
UML Model for XML Schema   **XML Schema**

- *Required*: false
- *Document global*: false
- *Unbounded*: true

UML Model for MOF 2.0 XMI   **XML Schema**
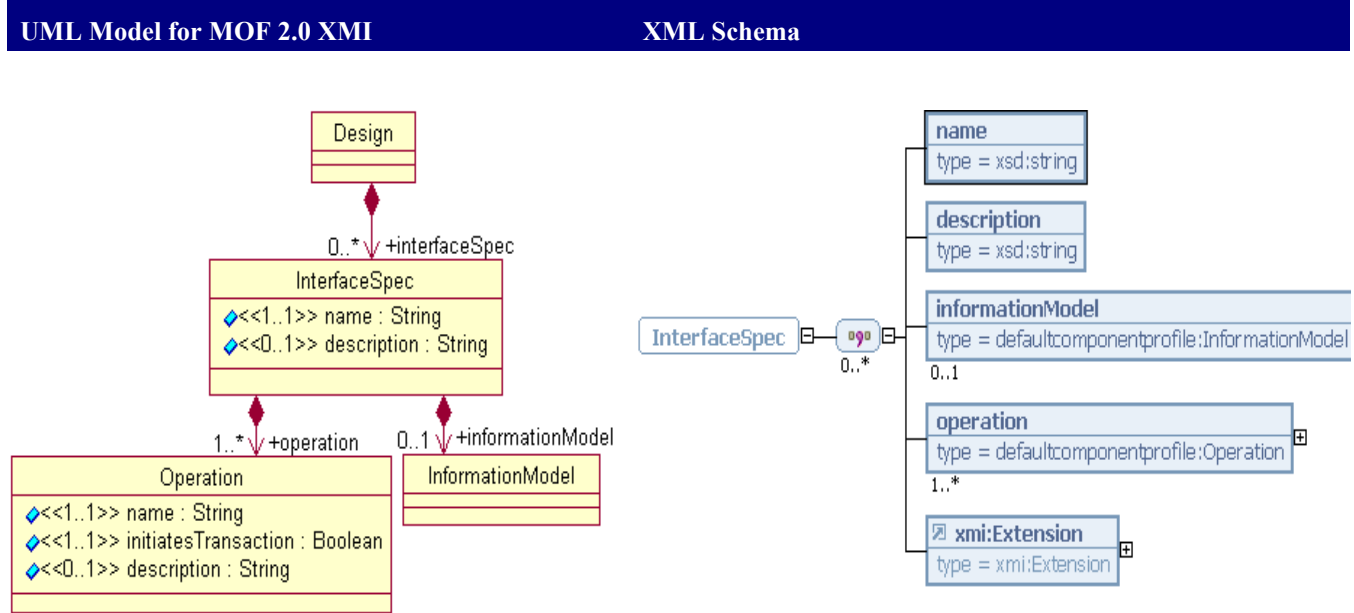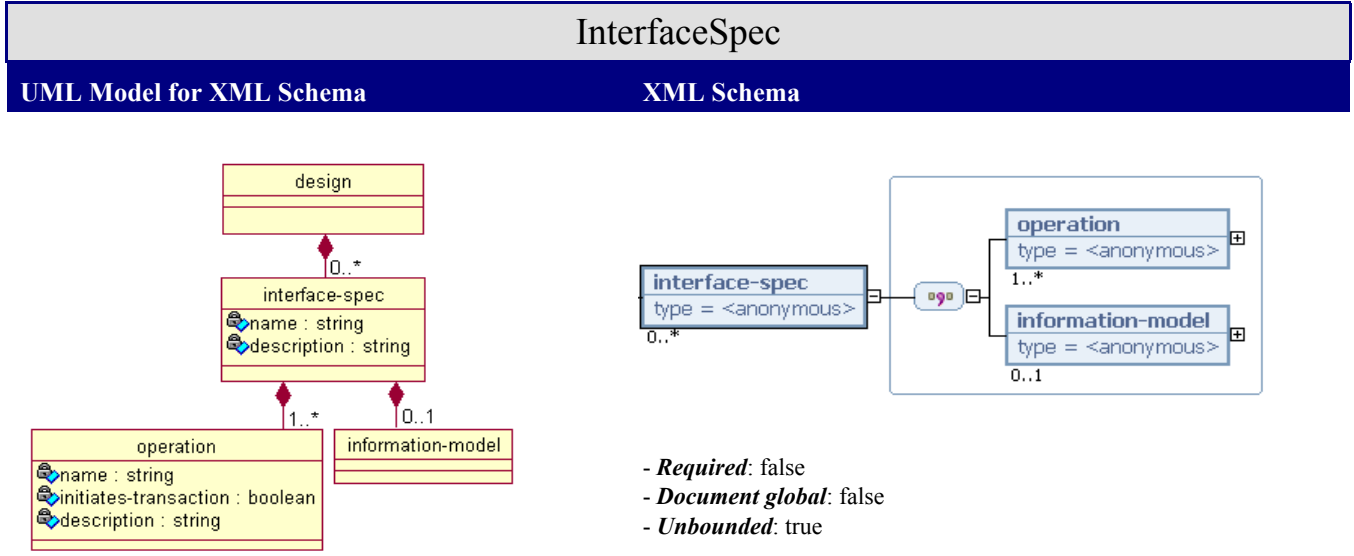
7.6.5.9 Operation

The Operation class describes one interface operation and has association with two classes, Condition and Parameter. These classes provide sufficient information in the asset packaging to let asset consumers and tools reason on the nature of the interface.

The Operation class has three attributes, *name*, *initiates-transaction*, and *description*. The *name* is the operation name. The *initiates-transaction* declares (i.e., boolean) if the Operation starts a transaction. The *description* provides for a brief abstract on the Operation.

Table 35 - Default Component Profile::Operation Class

Operation	
UML Model for XML Schema	**XML Schema**
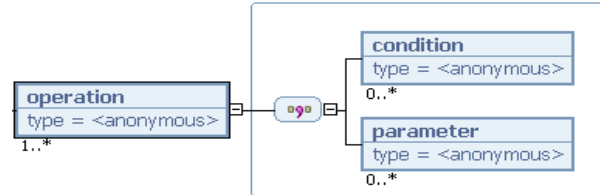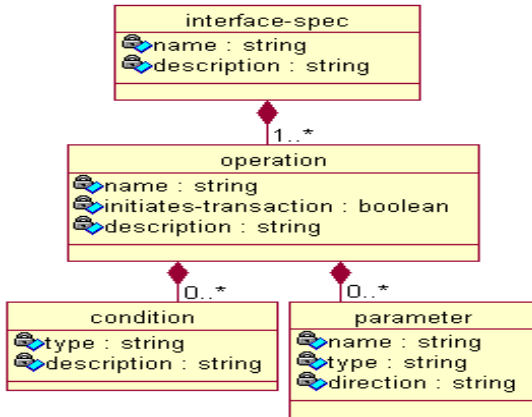
- *Required*: true
- *Document global*: false
- *Unbounded*: true

UML Model for MOF 2.0 XMI	**XML Schema**
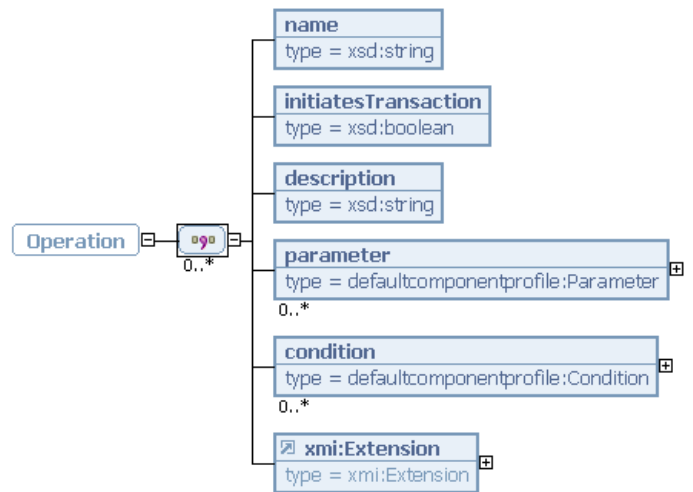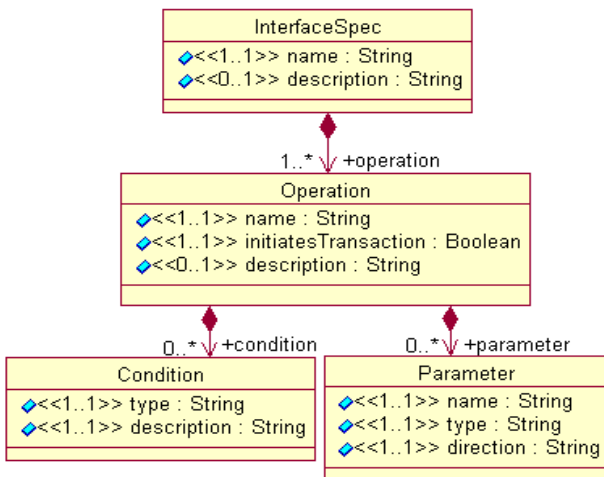
7.6.5.10 Condition

The Condition class captures the pre-, post- and other conditions of the Operation. It has two attributes *type* and *description* that declare the kind of the condition and explains the condition, respectively.

.

Table 36 - Default Component Profile::Condition Class

Condition	
UML Model for XML Schema	**XML Schema**

- *Required*: false
- *Document global*: false
- *Unbounded*: true

UML Model for MOF 2.0 XMI	**XML Schema**
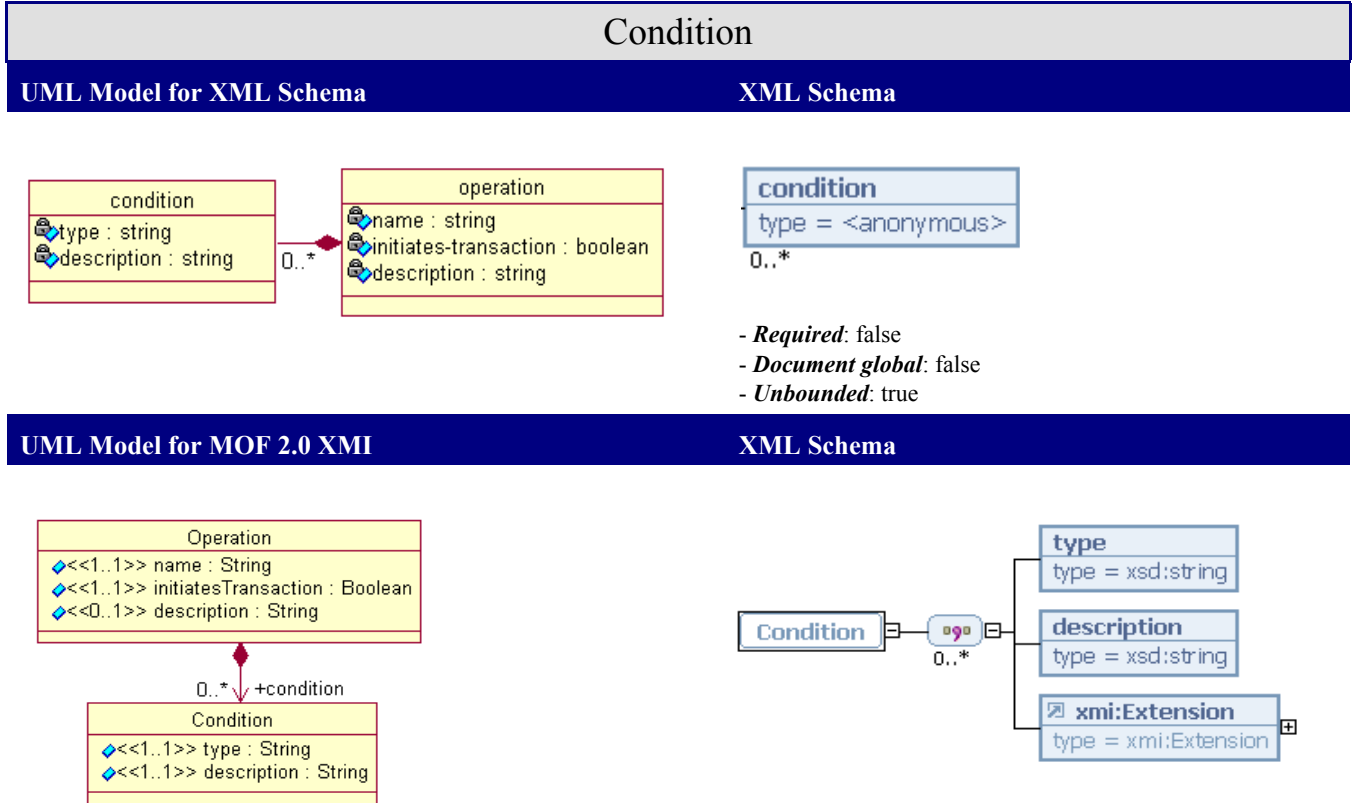
7.6.5.11 Parameter

The Parameter class describes the parameters on the Operation using the attributes, *name*, *type*, and *direction*. The *name* attribute is the name of the parameter. The *type* attribute describes the parameter's type. The *direction* attribute describes whether the parameter is input to the operation, or output, or both, and so on.

Table 37 - Default Component Profile::Parameter Class

Parameter	
UML Model for XML Schema	**XML Schema**
	- *Required*: false
	- *Document global*: false
	- *Unbounded*: true
UML Model for MOF 2.0 XMI	**XML Schema**
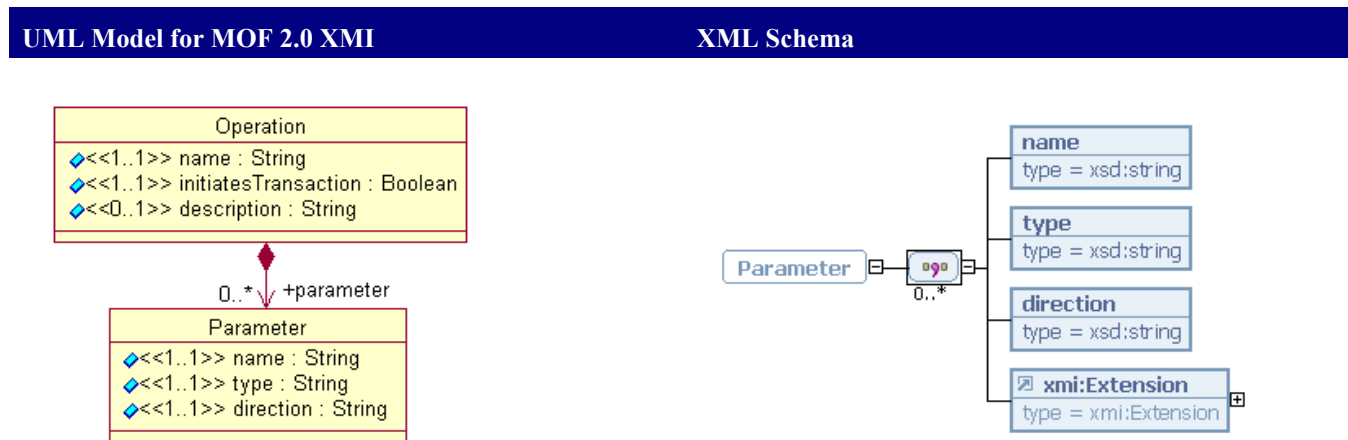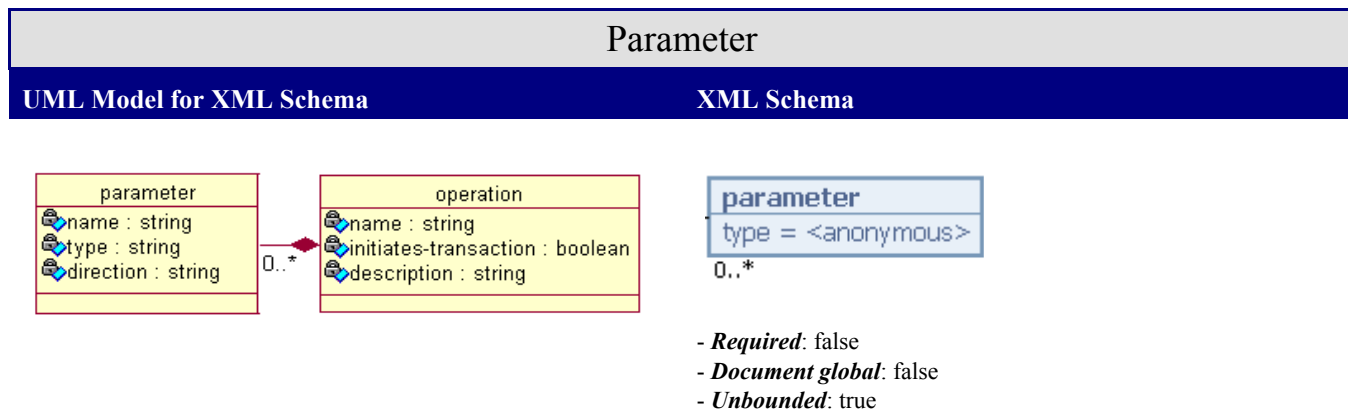
7.6.5.12 InformationModel

The InformationModel class describes the information or state that is retained between invocations of the operations on the interface. This class has two associations, Attribute and AssociationRole.

Table 38 - Default Component Profile::InformationModel Class

InformationModel	
UML Model for XML Schema	**XML Schema**

- *Required*: false
- *Document global*: false
- *Unbounded*: false

UML Model for MOF 2.0 XMI	**XML Schema**
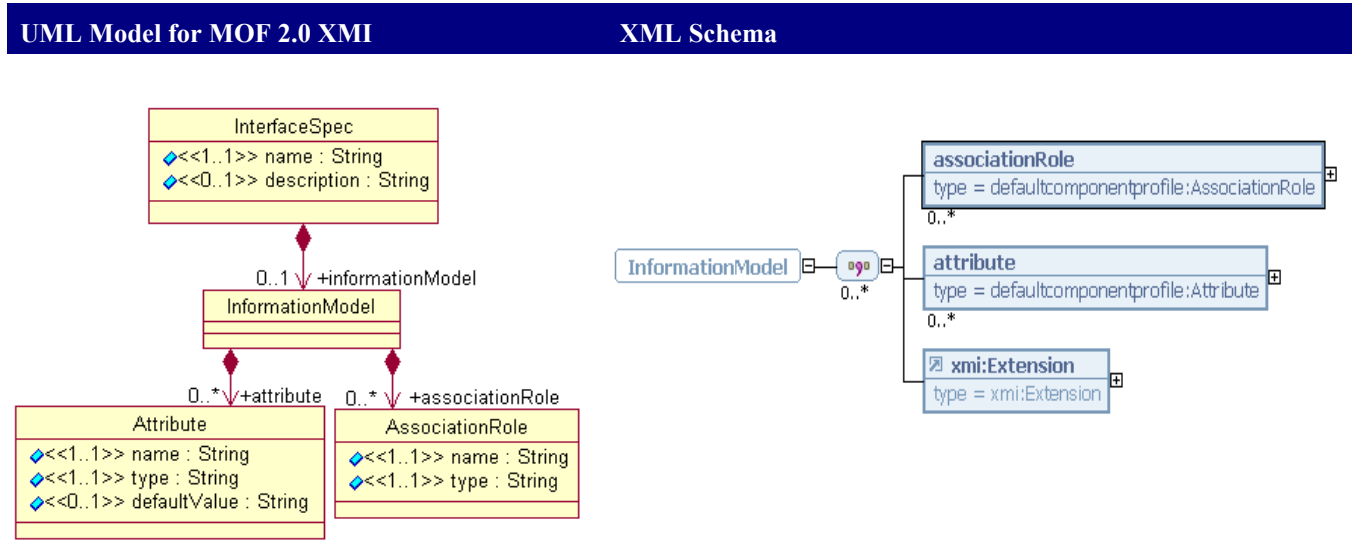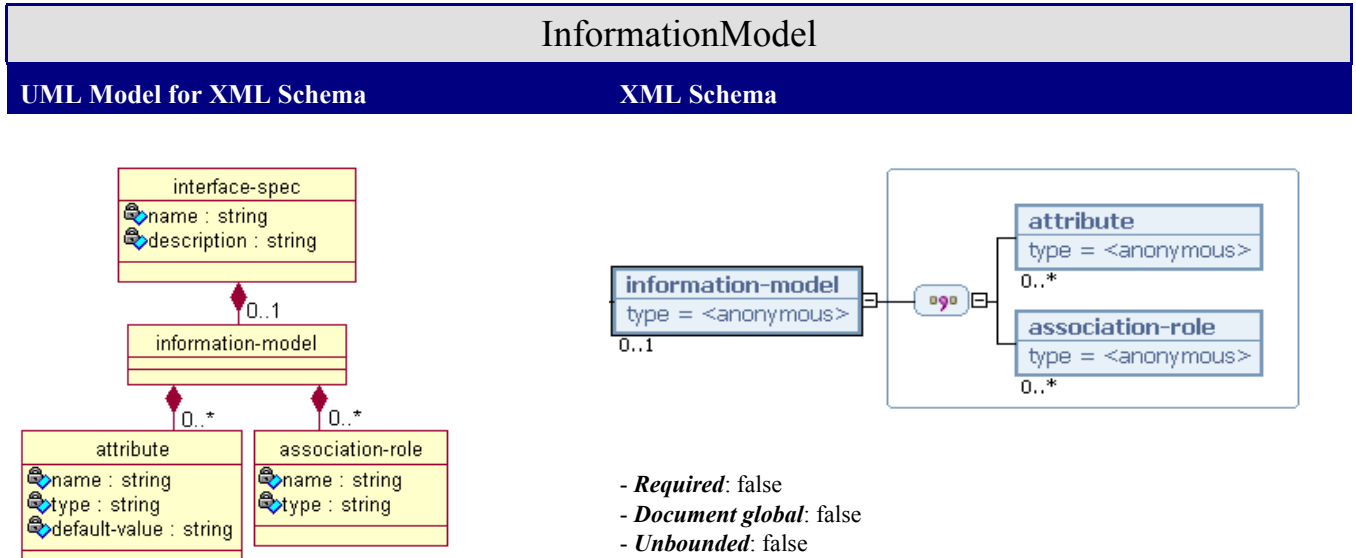
7.6.5.13 Attribute

The Attribute class captures the *name*, *type*, and *default-value* attributes of the state that is retained between invocations of operations on the interface. The *name* attribute is the name of the Parameter. The *type* attribute is the Parameter attribute's type. And the *default-value* attribute is the Parameter attribute's default value.
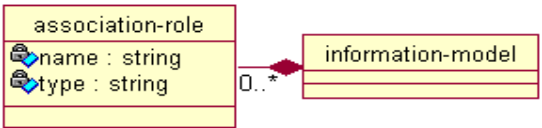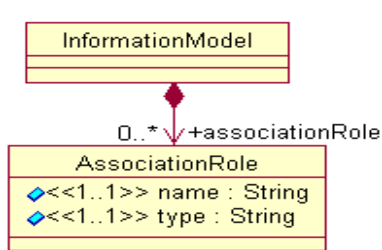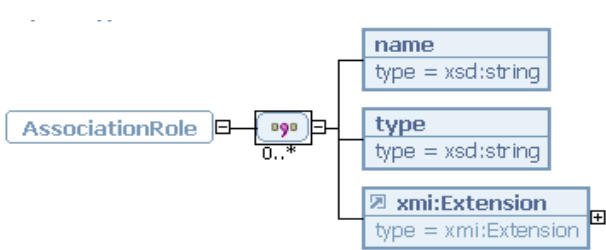
Table 39 - Default Component Profile::Attribute Class

Attribute

| UML Model for XML Schema | XML Schema |

- *Required*: false
- *Document global*: false
- *Unbounded*: true

| UML Model for MOF 2.0 XMI | XML Schema |
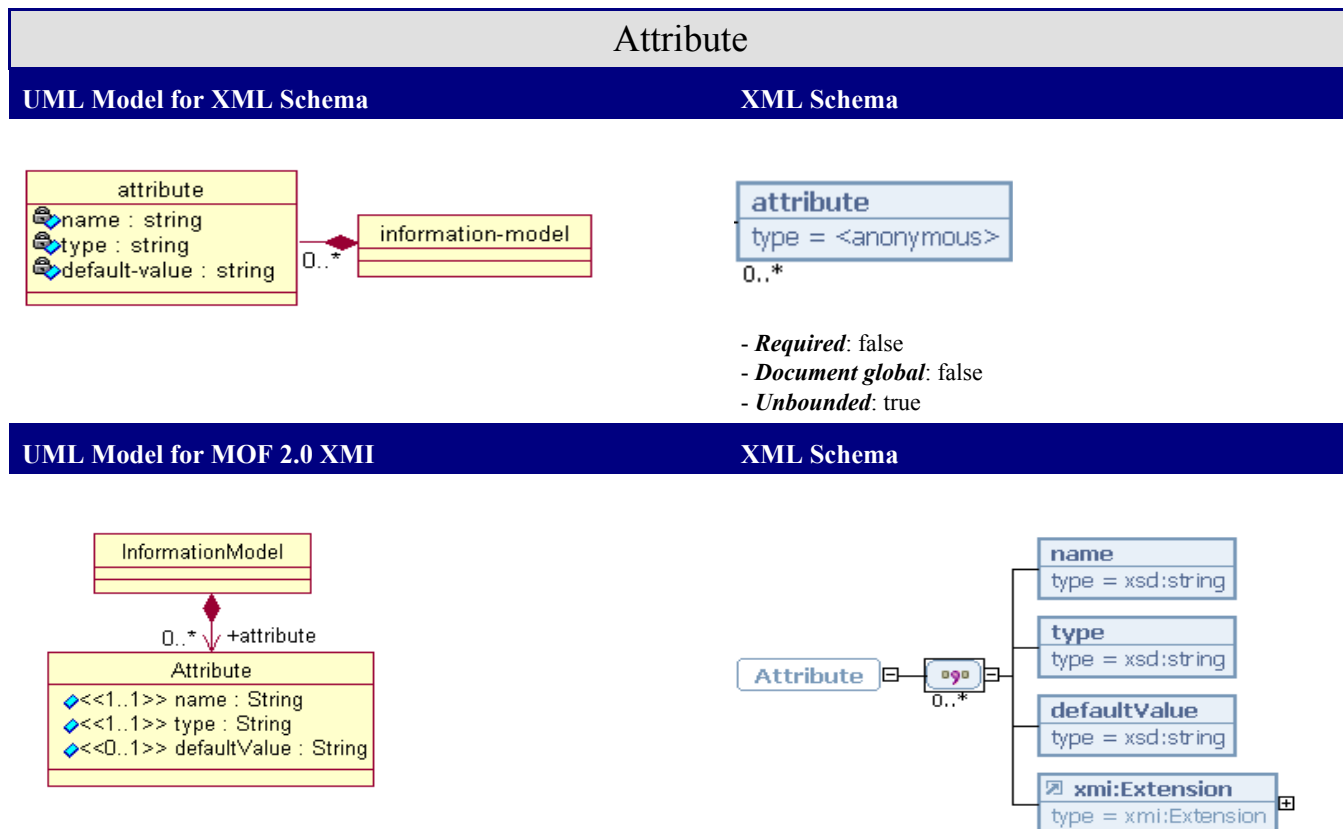
7.6.5.14 AssociationRole

The AssociationRole class captures the *name*, and *type* attributes of the state that is retained between invocations of operations on the interface. This element represents the roles that are on interface relationships. Creating instances of this element indicates that the InterfaceSpec owns the state related to the relationship.

Table 40 - Default Component Profile::AssociationRole Class

AssociationRole	
UML Model for XML Schema	**XML Schema**
	  - *Required*: false - *Document global*: false - *Unbounded*: true
UML Model for MOF 2.0 XMI	**XML Schema**

7.6.5.15 Implementation

The Implementation class has a collection of Artifacts. These Artifacts identify the binary and other files that provide the component implementation. The Implementation class has no attributes.

Table 41 - Default Component Profile::Implementation Class

Implementation	
UML Model for XML Schema	**XML Schema**

- *Required*: false
- *Document global*: false
- *Unbounded*: false

UML Model for MOF 2.0 XMI	**XML Schema**
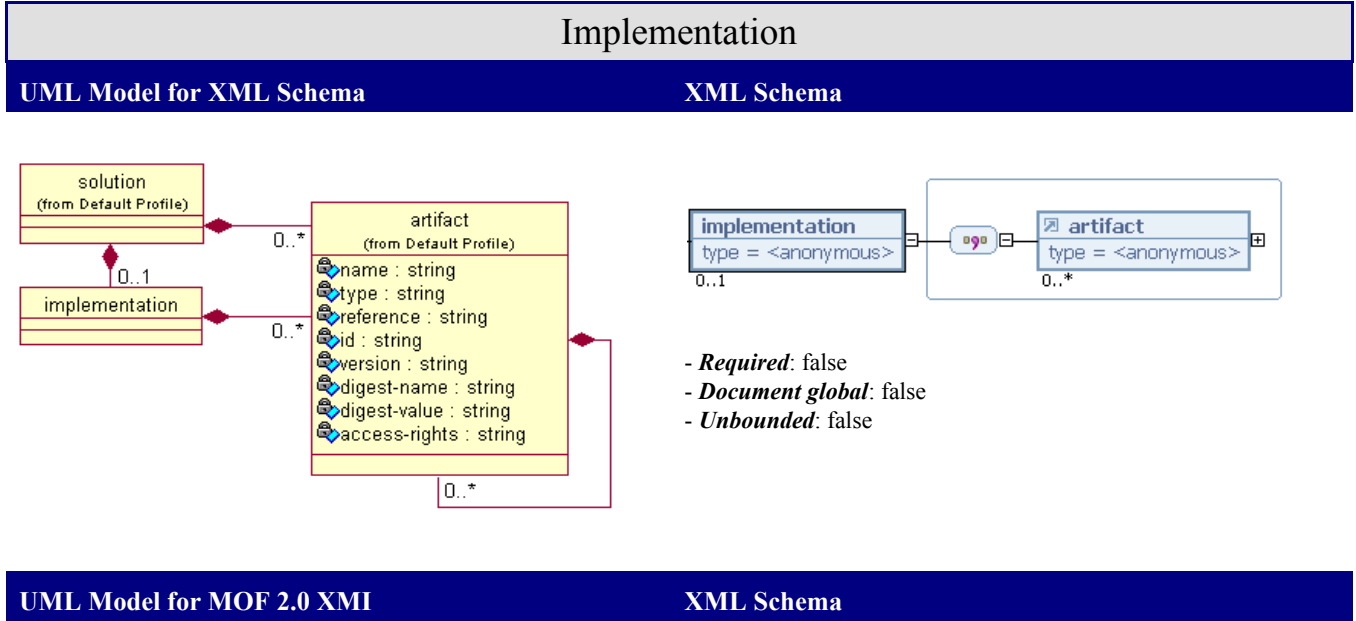
7.6.5.16 Test

The Test class has no attributes, and has associations with Model, Diagram, and Artifact. The models, diagrams, artifacts, and so on within this element are intended to describe the testing of the component for the asset consumer. The model, diagram, and artifact elements are global to the XML schema. Refer to earlier descriptions of these child elements.

Table 42 - Default Component Profile::Test Class
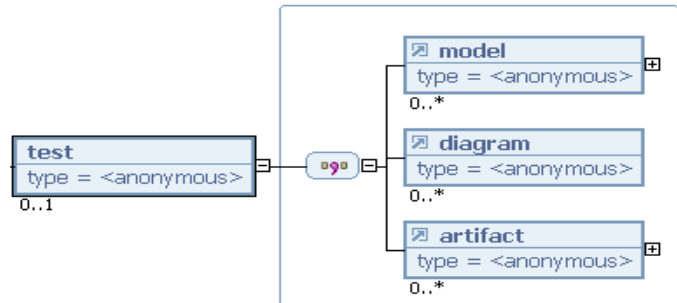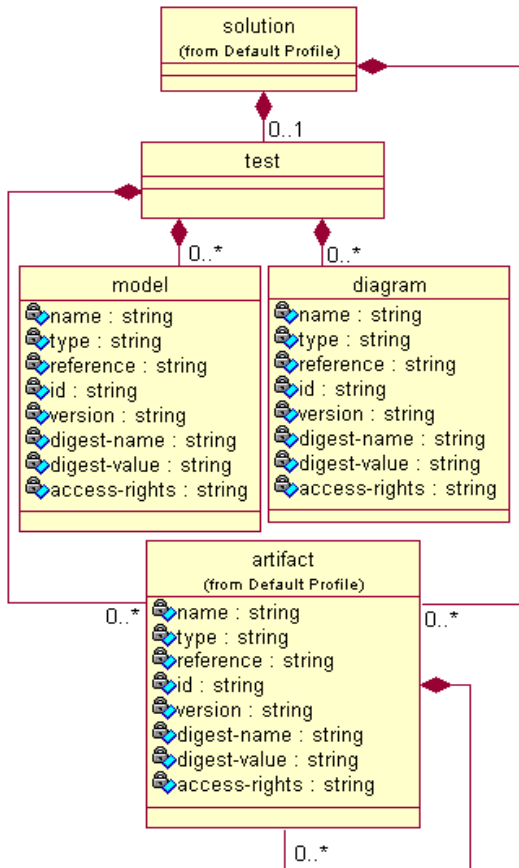
7.6.6   Default Component Profile Semantic Constraints

These constraints apply to the XML schema and may not apply to the MOF/XMI XML schema.

Constraint 1: For the <requirements> element; the child elements should contain only those artifacts that are relevant to requirements.

For the <design> element; the child elements should contain only those artifacts which are relevant to design.

For the <implementation> element; the child elements should contain only those artifacts which are relevant to implementation.

For the <test> element; the child elements should contain only those artifacts which are relevant to test.

All other artifacts should be handled in the <solution> element child <artifact>.

Constraint 2: The **diagram-id** attribute on the <diagram-dependency> element should reference a <diagram> element id in the manifest document.

Constraint 3: The **model-id** attribute on the <model-dependency> element contains the id value from a <model> element in the same manifest document.

Constraint 4: If you create an <interface-spec> element you must create one or more <operation> elements.

Constraint 5: The <condition> element **type** attribute should contain values such as "pre", "post".

Constraint 6: The <parameter> element **direction** attribute should contain values such as "in", "out", "inout".

7.7   Default Web Service Profile 1.1

Web services provide interfaces with operations, parameters, and an information-model of the guaranteed state. These characteristics are similar in nature to components; whereas the deployment and instantiation model is clearly different between these.

This profile describes the client portion of a web service. As such there are some similarities in the programming model between a component and the client side of the web service.

7.7.1   Default Web Service Profile History

The Default Web Service profile derives from the RAS Default Profile, version 2.1. In the XML schema for the Default Web Service the ancestry is traced to the Default Component profile because tooling support was very similar for these two schemas. However, in the UML model this profile derives from the Default Profile. The MOF/XMI XML schema ancestry is updated to derive from the Default profile.

The XML Schema for the Default Web Service profile has an <xsd:annotation> element. This annotation contains the profile history of this schema. This uses <xsd:appinfo> to describe the profile history which means the it can be machine readable. Note: this information does not appear in a manifest document (e.g., rasset.xml) but rather resides in the schema file.

The Default Web Service profile history is shown below, as taken from the XML schema file; again, this information only resides in the XML schema file. Therefore tool vendors need to open the XML schema file, retrieve this information, and populate the <profile> element and children elements in the manifest document (e.g., rasset.xml) as necessary.

<xsd:appinfo xmlns:rasprofile="http://www.rational.com/ras/raswebsvcprofile1_11" source="profile-history">

        <rasprofile:profile name="Default Web Service" id="F1C842AD-CE85-4261-ACA7-178C457018A1::31E5BFBF-B16E-4253-8037-98D70D07F35F::1025A790-78D4-4f57-94CE-E65B23275FCD::710CA9C5-CA9C-4be2-BB1A-D23677C62A4C" version-major="1" version-minor="11" parent="1025A790-78D4-4f57-94CE-E65B23275FCD">

                <rasprofile:description>This is the first major version of the default webservice profile. This profile can be accepted by XDE release 2 and later.</rasprofile:description>

                <rasprofile:related-profile name="Default Component" id="1025A790-78D4-4f57-94CE-E65B23275FCD" version-major="1" version-minor="11" parent="31E5BFBF-B16E-4253-8037-98D70D07F35F">This is the first major version of the default component profile. This profile can be accepted by XDE release 2 and later.</rasprofile:related-profile>

<rasprofile:related-profile name="Default" id="31E5BFBF-B16E-4253-8037-98D70D07F35F" version-major="2" version-minor="01" parent="F1C842AD-CE85-4261-ACA7-178C457018A1">This is the second major version of the default profile. This profile can be accepted by XDE release 2 and later.</rasprofile:related-profile>

<rasprofile:related-profile name="Core" id="F1C842AD-CE85-4261-ACA7-178C457018A1" version-major="1" version-minor="0" parent="">The original base of Core RAS.</rasprofile:related-profile>

</rasprofile:profile>

</xsd:appinfo>

7.7.2   Required Classes

The semantic constraints section describes the rules for certain elements and should be reviewed. In addition to the required elements in the Default profile, the Default Web Service profile adds the following required classes

- Implementation

- Wsdl

7.7.3   Required Attributes

In addition to the required attributes in the Default profile, the Default Web Service profile adds the following required attributes.

Table 43 - Default Web Service Profile::UML Model for XML Schema Required Attributes

Default Web Service Profile UML Model for XML Schema				
Required Class	**Required Attribute**		**Optional Class**	**Required Attribute**
implementation	-		interface-spec	wsdl-name
wsdl	reference			

Table 44 - Default Web Service Profile::UML Model for MOF 2.0 XMI Required Attributes

Default Web Service Profile UML Model for MOF 2.0 XMI				
Required Class	**Required Attribute**		**Optional Class**	**Required Attribute**
Implementation	-		InterfaceSpec	wsdlName
Wsdl	reference			

7.7.4   RAS Compliance

An asset based on this profile is RAS compliant if all of the following conditions are true:

1.   The RAS Compliance of the Default profile, version 2.1 is preserved.

2.   The constraints of the Default Web Service profile, version 1.11 are preserved.

7.7.5 Solution

Only the new elements for this profile are outlined here. For information on other elements refer to this profile's ancestry, namely the Default Component profile and the Default profile.

The models in this section only show those elements that are new or unique to this profile. The Solution section is the class that is extended for this profile and therefore the UML models illustrate elements from that section.

The Solution section has four new elements now including Requirements, Design, Implementation, and Test. These sections organize special kinds of Artifacts that improve browsing and navigation of the asset and also specify some required WSDL elements.
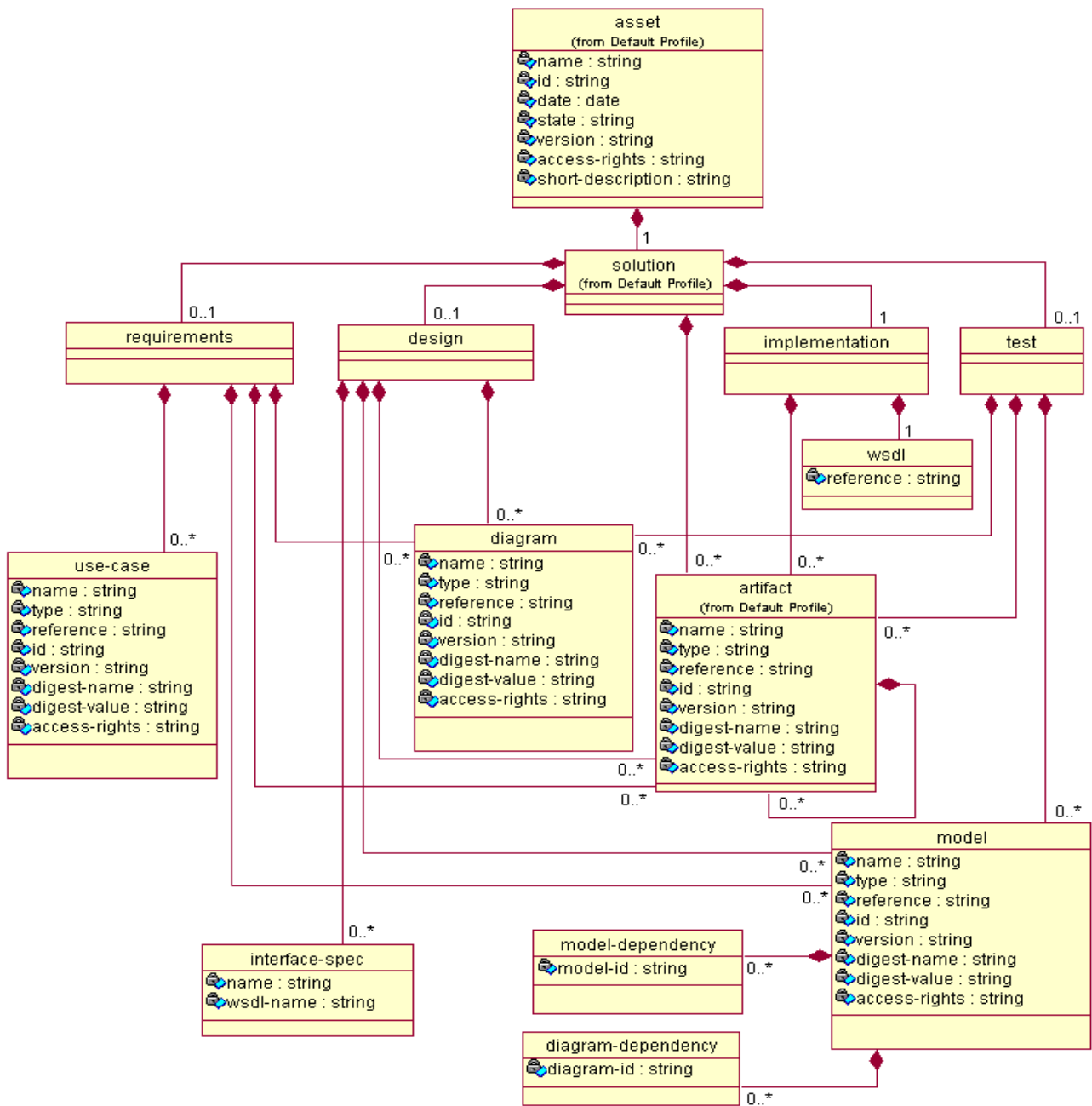
Figure 17 - Default Web Service Profile UML Model - for XML Schema

Figure 18 - Default Web Service Profile UML Model - for MOF/XMI XML Schema

The diagram above illustrates the new elements in the model. The Default Profile classes are grayed to illustrate the extensions to the Default Profile.

7.7.5.1 InterfaceSpec

Within a wsdl file is a section that describes the design of the interface. The InterfaceSpec class points to that section within a wsdl file. There may be multiple interfaces defined on the web service, so multiple instances of this element may be required. The *name* attribute is the user-consumable name of the interface and the *wsdl-name* is the name found in the wsdl.

The Wsdl class provides a reference to a formal description of the operations on the interface.

Table 45 - Default Web Service Profile::InterfaceSpec Class

7.7.5.2 Implementation

In this profile the Implementation class is required. The Implementation class has a collection of Artifacts. These Artifacts identify the binary and other files that provide the web service implementation. The Implementation class has no attributes. The Implementation class has an association with the Wsdl class.

Table 46 - Default Web Service Profile::Implementation Class

Implementation	
UML Model for XML Schema	**XML Schema**
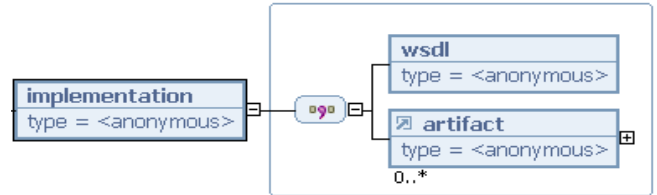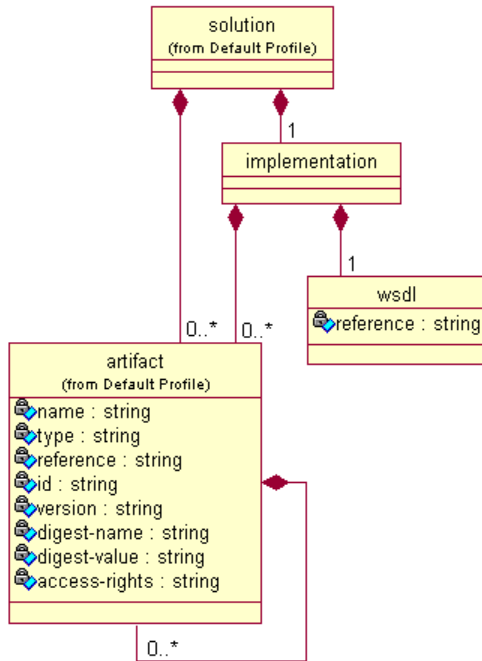
- *Required*: true
- *Document global*: false
- *Unbounded*: false

Table 46   (continued)

UML Model for MOF 2.0 XMI	XML Schema
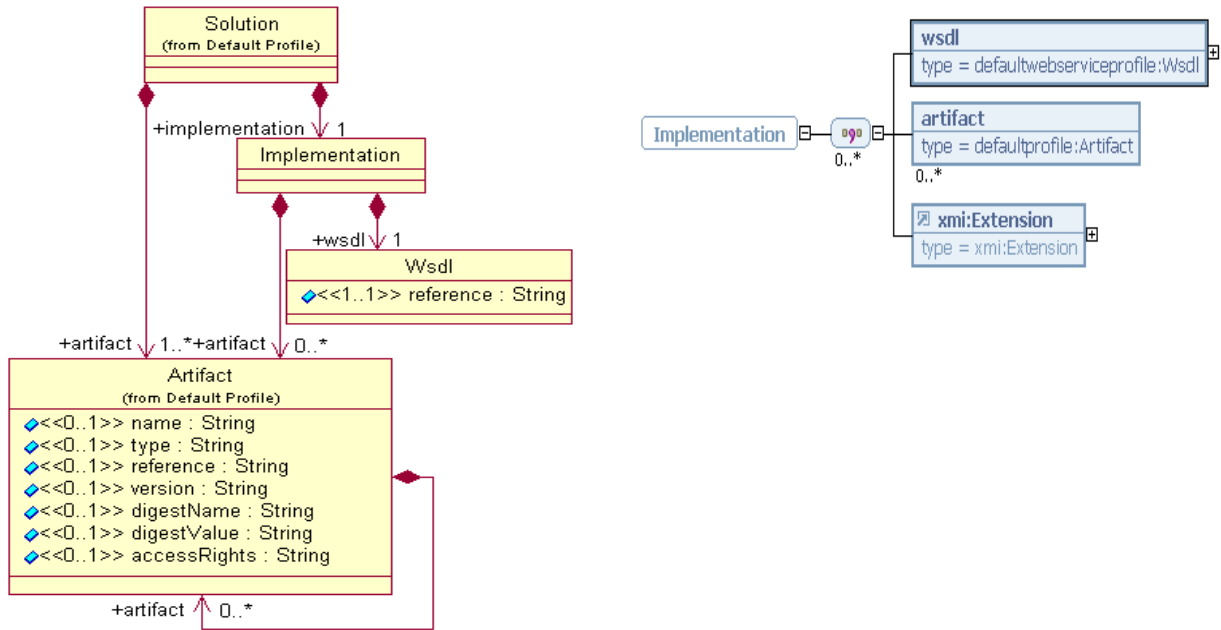
7.7.5.3 Wsdl

The Wsdl class references the file containing the web service description..

Table 47 - Default Web Service Profile::Wsdl Class

Wsdl	
UML Model for XML Schema	**XML Schema**

- *Required*: true
- *Document global*: false
- *Unbounded*: false
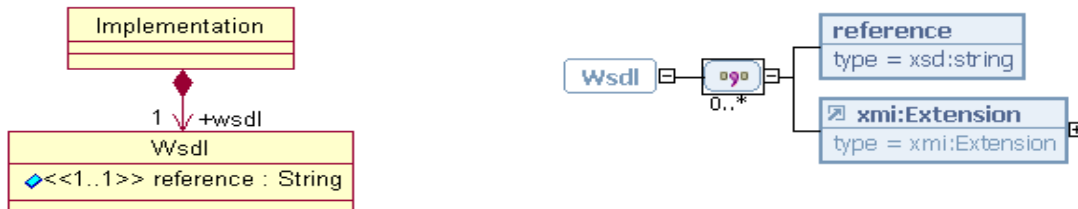
7.7.6   Default Web Service Profile Semantic Constraints

These constraints apply to the XML schema and may not apply to the MOF/XMI XML schema.

Constraint 1: For the <requirements> element; the child elements should contain only those artifacts that are relevant to requirements.

For the <design> element; the child elements should contain only those artifacts which are relevant to design.

For the <implementation> element; the child elements should contain only those artifacts which are relevant to implementation.

For the <test> element; the child elements should contain only those artifacts which are relevant to test.

All other artifacts should be handled in the <solution> element child <artifact>.

Constraint 2: The ***diagram-id*** attribute on the <diagram-dependency> element should reference a <diagram> element id in the manifest document.

Constraint 3: The ***model-id*** attribute on the <model-dependency> element contains the id value from a <model> element in the same manifest document.

Constraint 4: If you create an <interface-spec> element you must create one or more <operation> elements.

Constraint 5: The <condition> element ***type*** attribute should contain values such as "pre", "post".

Constraint 6: The <parameter> element ***direction*** attribute should contain values such as "in", "out", "inout".

8 The .ras File Format

8.1 Mapping RAS to .ras Files

While RAS is a written specification, we needed to express these principles more formally to support tools. To do so we used XML Schema (i.e., .xsd files) to describe this. XML Schema possesses the rigor to describe containment, type, multiplicity, and so on. To create this we used the RAS UML models as a baseline for creating the initial .xsd file(s). The first version of this was the RAS Default Profile, which describes any kind of asset.

From the RAS Default Profile XML Schema we can create XML documents which contain the elements necessary to describe the contents of an asset. We refer to this XML document as the manifest file and we give it a formal name: rasset.xml. The rasset.xml document serves as the entry point to the asset.

The rasset.xml file resides in the "root" directory of the asset. This file is accompanied by the .xsd (XML Schema) file and any other artifacts, files, subdirectories, and so on. These files are zipped into a single file with a .ras extension. The image below illustrates the relationship of the RAS with the .ras file.
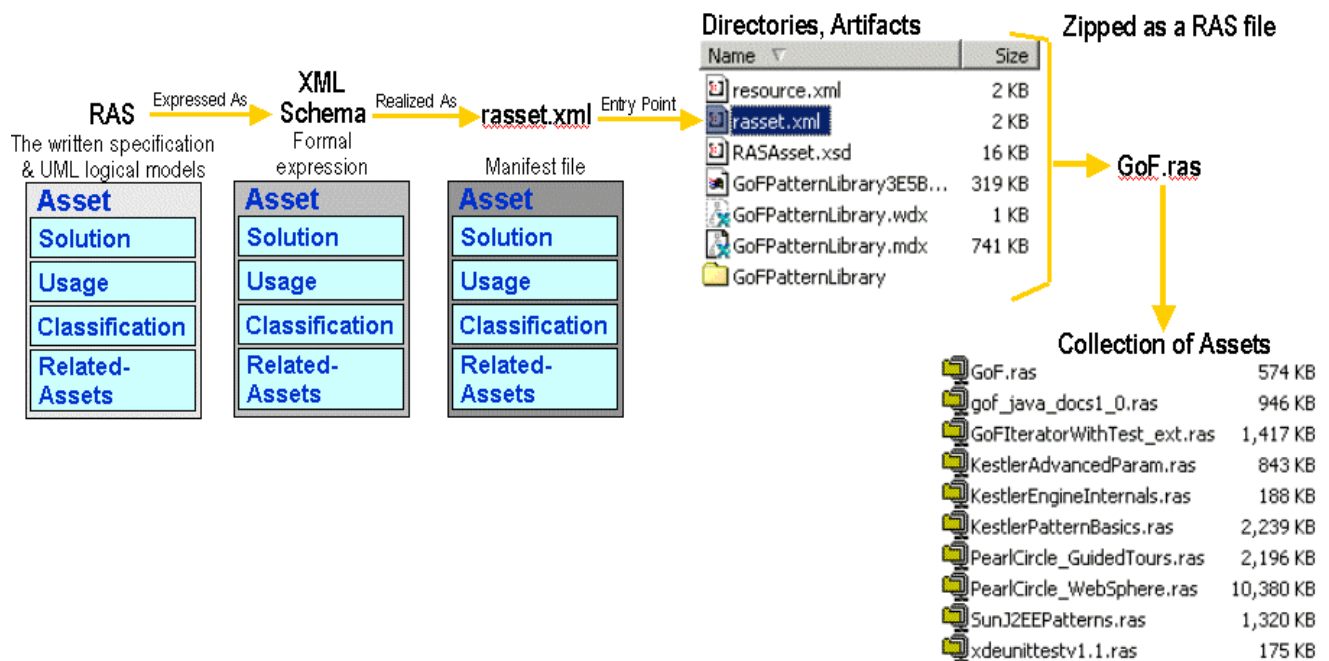
Figure 19 - Mapping RAS to .ras Files

Each .ras file includes the following types of files:

- one XML Schema file (e.g., `RASProfile.xsd`)

- one manifest file (e.g., `rasset.xml`)

- one or more artifact files (e.g., source code, models, test scripts, and so on)

The image below shows a sample .ras file for a web service client. In the image below the file is opened with WinZip to show the basic structure. There are two files in asset root directory, namely the rasset.xml file and the RASDefaultWebServiceProfile.xsd file. The remaining artifacts are located in several sub-directories that are relative to the asset's root directory. When the asset is imported into a tool the directory structure is preserved.

Each artifact in the .ras file (other than the rasset.xml file and the XML Schema file) must be referenced in the rasset.xml <solution> element. Each artifact (i.e., file) should appear one time int the .ras file.
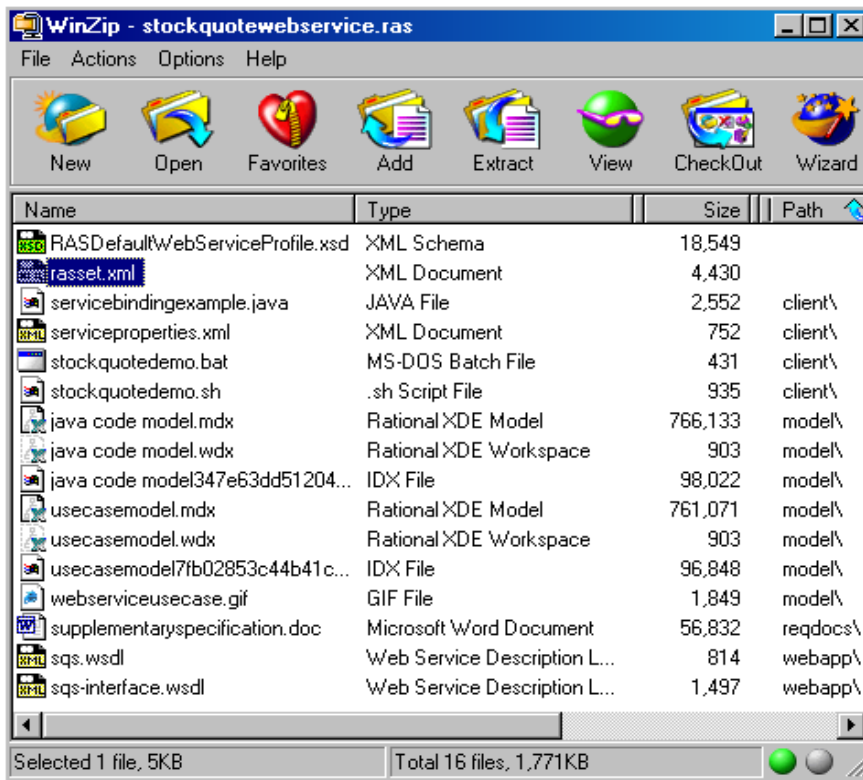
Figure 20 - Sample .ras File Contents

8.1.1   Organizing .ras Files

The .ras files can be organized on a filesystem or may be in a version control system and may be organized by asset type or by version or state, and so on.

8.1.2   Browsing .ras Files

Tool vendors can examine the rasset.xml file in a .ras file to extract the asset's name and short-description when presenting lists of assets. The rasset.xml file structure easily supports conversion to HTML for simplified browsing.

9   MOF & XMI

There are several MOF models that will be produced for describing RAS. These models include one describing the RAS Default profile, one describing the RAS Default Component profile, and one describing the RAS Default Web Service profile.

MOF has a mapping to XMI. The XMI content structure supports information interchange between various tools, as illustrated in the image below.
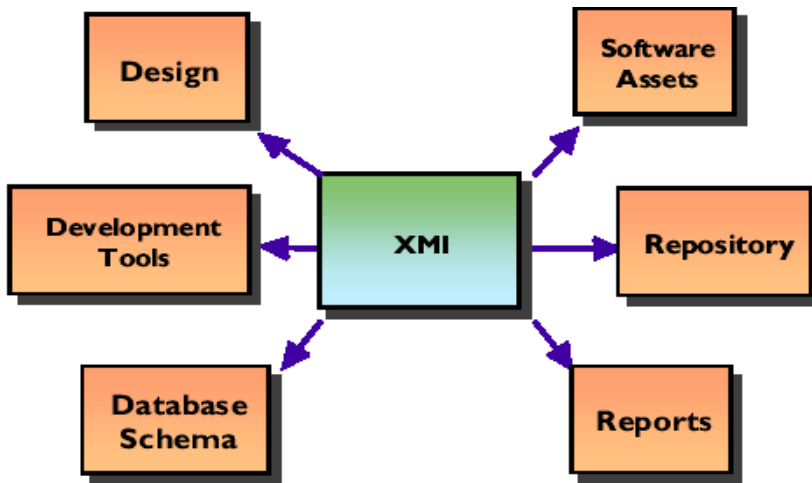
Figure 21 - Open Interchange with XMI (from XMI Opens Application Interchange document)

Using this approach increases sharing content with multiple intended target environments as well as unintended target environments. RAS describes an asset as being relevant to one or more contexts. Creating a RAS asset with XMI format does not guarantee that all artifacts will be consumable by *any* development tool. However, XMI does enable sharing assets across XMI-enabled development tools that support similar contexts, as described by RAS.

10 RAS Repository Service

With the predictable organization of the .ras files and the structure of the rasset.xml file, assets can be searched, browsed, retrieved, and so on. This section introduces a set of services for searching, browsing, and retrieving assets. This version of the RAS Repository Service does not describe asset publishing and other asset management services including metrics. While asset publishing and other services are clearly needed, we have started with the RAS Repository Service in the anticipation of helping the asset consumer to get initial value from a RAS-based repository.

These services may be implemented as web services or may be part of a larger product. For each of the services the nature of the request and the response is declared. However, this does not describe how the services should be implemented.

The services below are described with a Service Name, a Request, and a Response. The Request is formatted, as it would be for an HTTP request. The Response is a Repository Data Descriptor, of which there are two kinds, a Repository Asset Descriptor, and a Repository Folder Descriptor. The format of these descriptors in the result set is described below.
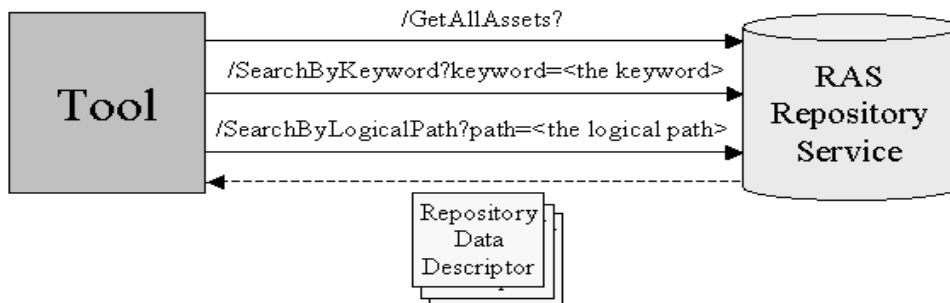
Figure 22 - RAS Repository Service Overview

The services described in this section are for small to medium size repositories. The services need to be refined for repositories with large numbers of assets.

10.1   Http Request / Response Descriptions

Service Name: Get All Assets

Request: /GetAllAssets?

Response: Collection of Repository Data Descriptors

A Repository Data Descriptor is either a Repository Asset Descriptor or a Repository Folder Descriptor

> A **Repository Asset Descriptor** contains the following:
>
>> **String**: Name (maps to the name attribute in the asset)
>>
>> **String**: Description (This should be at most a 2 sentence description -- maps to the short description attribute in the asset)
>>
>> **String**: URL to Asset Location (Downloading the file at this URL should provide the .ras file)
>>
>> **String**: Logical Path (Root is indicated by /)
>>
>> **String**   : Version (maps to the version attribute in the asset)

> **int**: Ranking (between 0 and 100, 100 being best match)

A **Repository Folder Descriptor** contains the following:

> **String**: Name

> **String**: Logical Path (Root is indicated by /)

Search by Keyword

> **Request**: /SearchByKeyword?keyword=<the keyword>

>> Where <the keyword> is a "form encoded" string of the keywords to search for.

>> This request should search at least the asset's metadata. In particular the name, id, version, short description, description, and classification section

> **Response**: Collection of Repository Asset Descriptors

>> A **Repository Asset Descriptor** contains the following:

>>> **String**: Name (maps to the name attribute in the asset)

>>> **String**: Description (This should be at most a 2 sentence description -- maps to the short description attribute in the asset)

>>> **String**: URL to Asset Location (Downloading the file at this URL should provide the .ras file)

>>> **String**: Logical Path (Root is indicated by /)

>>> **String**: Version (maps to the version attribute in the asset)

>>> **int**: Ranking (between 0 and 100, 100 being best match)

Search by (Logical) Path

> **Request**: /SearchByLogicalPath?path=<the logical path>

>> Where <the logical path> is a "form encoded" string of the logical path to an asset or folder. The root folder of the repository is indicated by /.

>> This request can be used for instance when browsing a repository. One can use this to build a tree view of the logical structure of the repository.

> **Response**: Collection of Repository Data Descriptors

>> A Repository Data Descriptor is either a Repository Asset Descriptor or a Repository Folder Descriptor

>> A **Repository Asset Descriptor** contains the following:

>>> **String**: Name (maps to the name attribute in the asset)

String: Description (This should be at most a 2 sentence description -- maps to the short description attribute in the asset)

String: URL to Asset Location (Downloading the file at this URL should provide the .ras file)

String: Logical Path (Root is indicated by /)

String: Version (maps to the version attribute in the asset)

A **Repository Folder Descriptor** contains the following:

String: Name

String: Logical Path (Root is indicated by /)

10.2   Java API Descriptions

The Http request / response descriptions outlined above are expressed as a Java API in this section.
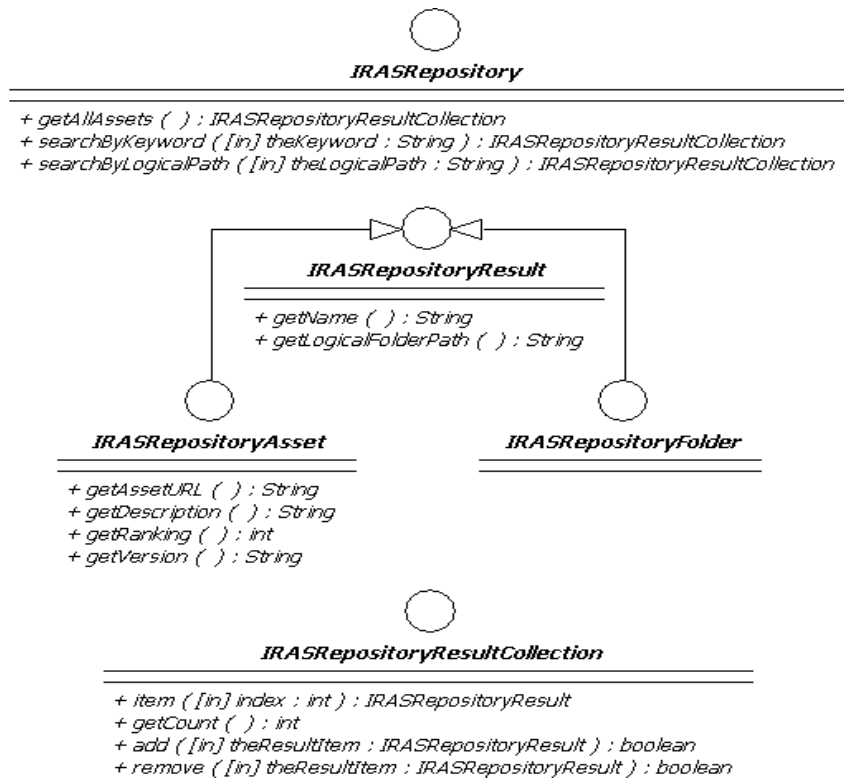
Figure 23   - RAS Repository Service - Java API

Each of these interfaces and their operations are described further below.

10.2.1 IRASRepository

```
/**
 * IRASRepository:
 * This is the interface for talking to a repository
 */
public interface IRASRepository {

  /**
   * Get All Assets:
   * This searches the repository and returns all the assets in
     them. If no Assets are found to match, then it returns an
     empty collection.
   */
  public IRASRepositoryResultCollection getAllAssets() throws
  java.io.IOException;

  /**
   * Search by Keyword:
   * This searches the repository and returns a collection of
     assets that match the given keyword. If no assets are found to

     match, then it returns an empty collection.
   */
  public IRASRepositoryResultCollection searchByKeyword(String
  theKeyword) throws java.io.IOException;

  /**
   * Search by Logical Path:
   * This searches the repository and returns a collection of
     assets and folders in the folders logical path. If none are
```

```
            found to match, then it returns an empty collection.

    * The root is indicated by /

   */

  public IRASRepositoryResultCollection searchByLogicalPath(String

  theLogicalPath) throws java.io.IOException;

}
```

10.2.2 IRASRepositoryResult

```
/**

 * This is the base result object that is returned from a search of

   a RAS repository. The 2 known flavors are IRASRepositoryAsset

   and IRASRepositoryFolder

*/

public interface IRASRepositoryResult {

  /**

   * Get Name:

   * The display name of the result item.

   */

  public String getName() throws java.io.IOException;


  /**

   * Get Logical Folder Path:<br>

   * The logical path of the result item. This is used to organize

     the assets.

   * The root folder is indicated using /

   */

  public String getLogicalFolderPath() throws java.io.IOException;

}
```

10.2.3 IRASRepositoryFolder

```
/**
 * This is the object represents an (logical) folder in a RAS
   repository.
*/
public interface IRASRepositoryFolder extends IRASRepositoryResult
{
}
```

10.2.4 IRASRepositoryAsset

```
/**
 * This is the object represents an asset in a RAS repository.
*/
public interface IRASRepositoryAsset extends IRASRepositoryResult
{

  /**
   * Get Asset URL:
   * The URL to download the asset from.
   */
  public String getAssetURL() throws java.io.IOException;


  /**
   * Get Description:
   * A short description of the asset.
   */
  public String getDescription() throws java.io.IOException;


  /**
   * Get Ranking:
```

```
     * The ranking (low of 0 to high of 100) on how the item ranked.

       This is used for sorting the results. Note rankings outside

       the range of 0 - 100, are given a 0 ranking.

    */

    public int getRanking() throws java.io.IOException;


    /**

     * Get Version:

     * The version of the asset.

     */

    public String getVersion() throws java.io.IOException;

}
```

10.2.5 IRASRepositoryCollection

```
/**

 * IRASRepositoryResultCollection:

 * This interface holds a list of assets and folders.

 */

public interface IRASRepositoryResultCollection {


  /**

    * Item:

    * Get the IRASRepositoryResult at the given index.

   */

   public IRASRepositoryResult item(int index) throws

   java.io.IOException;


   /**

    * Get Count:

    * The number of IRASRepositoryResult in the list.

   */
```

```
public int getCount() throws java.io.IOException;


/**
 * Add:
 * Add an item to the collection.
*/
public boolean add(IRASRepositoryResult theResultItem) throws

java.io.IOException;


/**
 * Remove:
 * Remove an item from the collection.
*/
public boolean remove(IRASRepositoryResult theResultItem) throws

java.io.IOException;
}
```

11 Roadmap

There are several areas to continue with defining RAS. Some of these are listed below, although these are not listed in any particular order. Ultimately this roadmap needs to include timing.

1. The <descriptor> node and HTML encoding

    Add attributes to this node to describe the type of encoding rather than relying on that it might be plain text or HTML. Additional attributes to consider include:

    • formatting={HTML|PostScript|RTF|SGML|TeX|LaTeX etc.}

    • language={English|Spanlish|| etc.}

2. The Default Web Service <interface-spec> element

    The web service <interface-spec> element is missing the operation and other elements from the Default Component profile. This needs to be restored; although the WSDL will define these operations for us.

3. Deprecate the old UML models for XML schema, and the XML schemas themselves, at some point.

4. Create a UML profile for RAS.

5. Integration and reuse of UML 2.0 and related OMG meta-models (such as Software Portfolio Management).

6. There are several items to update in the schema including:

    • Remove the use of concatenated GUIDs to represent profile ancestry

    • Need to add a reference attribute on the Description node so the description can be a separate document.

    • Need to specify that the Description node should contain only plain text and not HTML; we have the Artifact node with the Artifact Type that allows us to declare different types of documents.

    • Replace the "01" from the asset minor attribute with "1".

Glossary

Entries in this glossary are defined within the Asset-Based Development (ABD) context. Their meanings are therefore written with a bias towards ABD.

Apply Asset	An ABD activity where a consumer uses a reusable asset to solve a problem. Applying an asset usually involves following the usage guidance specified by the asset.
Archive	A bundled collection of files that can be handled as a single unit. Each file's name and relative location in the directory structure is preserved. The collection's contents may compressed. In this sense a .ras file is an archive.
Artifact	A logical or physical element of an asset. A logical asset is a container of at least one physical artifact. Physical artifacts correspond to a file on a filesystem and represent a workspace product.
Asset	An asset is a solution to a software development problem. The problem may be related the evolution of the system's artifacts or be directly related to the domain problem that the system is being developed for. (see Reusable Asset)
Asset Based Development (ABD)	A sub-methodology in the software development process. Although not a complete software development process, asset-based development is a set of processes, activities and standards that facility the reuse of assets. Asset-based development is architecture centric.
Black Box Asset	A type of reusable asset in which the artifacts of the asset are not viewable by its consumers. Examples of black box assets are components and framework libraries.
Clear Box Asset	A type of reusable asset in which the artifacts of the asset are visible by its consumers, however they cannot be altered or modified in any way. These types of asset expose their internals to help consumers understand how to better use and debug the asset.
Component	A type of asset that adheres to a documented interface. Components typically have their implementations hidden (i.e., binary components).
Consumer	A role in the Asset-based development process. A consumer is a software developer that applies a reusable asset.
Context	A frame reference or conceptual boundary that establishes meaning for things associated with the context.
Core RAS	The baseline description of the reusable asset specification.
Dependency	A relationship between two objects (things), where one object is "dependent" on the other. When the dependent object changes it effects the depending object. A dependent object may not be aware of the depending object.
Descriptor	A key / value pair of information used to describe an asset. A descriptor name is the key and is typically a human readable word or two. The value is also human readable and may be a sentence or as long as a paragraph or two.

Descriptor Group	A group of related descriptors.
Document Type Definition (DTD)	A formal specification that defines the structure of XML document instances. This specification is managed by the W3C.
Framework	A type of asset that solves many problems. A framework is often a collection of individual assets, or a set of middleware that applications are built on top of.
Gray Box Asset	A type of asset in which some of the internals remain hidden to the consumer, but others are visible and modifiable. Gray box assets maintain variability somewhat between black and white box assets.
Harvest	An ABD activity for creating assets from existing, functioning systems. Harvesting is performed by the asset producer. The producer looks in existing system's for things that could be reworked as reusable assets. Harvesting attempts to find elements in existing systems that with minor effort could be turned into reusable assets.
Idiom	A type of asset that is small and at the code or algorithm level.
Librarian	A role in the ABD. The librarian is responsible for the maintenance of the asset repository. The librarian may perform additional classification of asset and is responsible for managing any feedback from consumers.
Manifest	A meta information document that describes a reusable asset. A manifest is an XML document that validates against a Profile.
Metadata	Information about data. A manifest document is meta data about an asset. It describes the structure and elements of the asset.
Package	A collection of artifacts (files) that make up an asset. A package could be realized as a directory on a filesystem or as an archive.
Pattern	A type of asset that is an abstraction of the structure and behavior of a system or part of a system. A pattern can be applied to a system, in which the application causes elements of the system to be structured in a certain way.
Problem	An impediment in the software development life cycle. Problems encountered in the development life cycle must be solved (or avoided) in order to meet the target application's requirements. A reusable asset solves wholly or in part a software development life cycle problem.
Producer	A role in the ABD that is responsible for the creation of reusable assets. A producer can harvest assets from existing systems or create reusable assets from scratch that solve a reoccurring problem.
Profile	A collection semantic constraints and an XML Schema that together are used to validate a manifest document. A profile defines what information is required and optional in the manifest to describe an asset of a particular type.
Repository	A centralized access and storage point for reusable assets. A repository facilitates consumer activities such as searching and analysis.

Reusable Asset	A reusable asset is a solution to a recurring problem. A reusable asset is an asset has been developed with reuse in mind.
Reusable Asset Library	The Reusable Asset Library is a conceptual composite artifact that encompasses all possible Reusable Assets of which an Asset Consumer has access.
Reuse Coordinator	A senior management role. Responsible for the overall reuse program in an organization. The coordinator ensures that developers are leveraging reusable assets when appropriate.
Reuse scope	The conceptual bounds of the reuse program in an organization (or beyond). The reuse scope attempts to identify the limits of the terminology and unique identifies the elements in a reuse program are compared against.
Root context	The top-level directory of an asset package. The root context defines the boundary of an asset's artifacts. This will not be true of course if we allow URL artifacts.
Solution strategy	The general strategy taken by a reusable asset to solve the problem. The solution strategy is an abstracted or simplified version of the solution design.
Target application	An application or system with problem(s) that reusable assets can solve. A reusable asset is applied to a target application by the consumer.
Tooling	A generic term use to describe software programs written to handle and manage RAS manifest documents and RAS asset packages. Rational XDE is an example of a commercial tool that can create and consume RAS assets.
Unified Modeling Language (UML)	The defacto standard visual modeling language for software intensive systems.
Variability Point	A point in an artifact that is expected to be modified when the asset is applied to a target application.
White Box Asset	A type of asset that where all of the internals are exposed for review or modification.
Workspace Product	An artifact of the software development process. A workspace product is a tangible artifact that can be manipulated by a worker.
XML Schema	A formal specification that defines the structure of XML document instances. This specification is managed by the W3C.