Modeling with openMDX Part 2

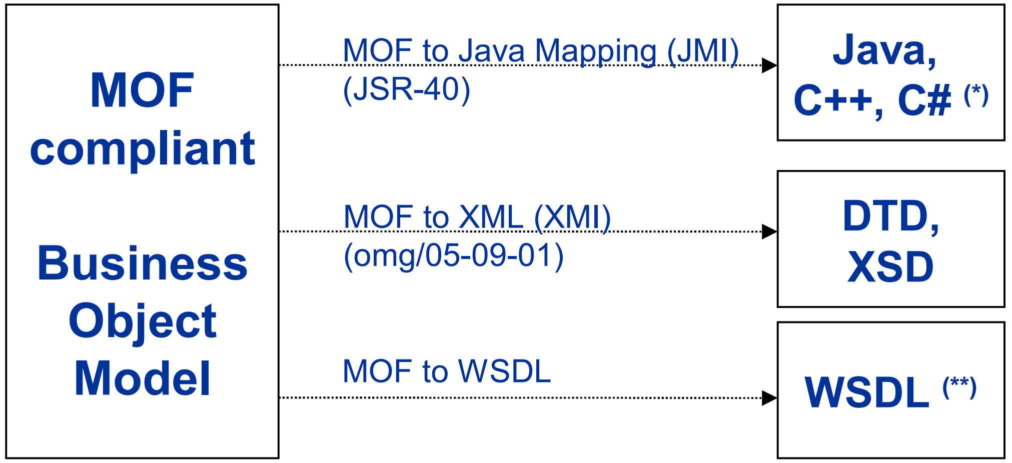
openMDX - Overview

- openMDX implements the PIM-only approach.
- As a consequence, openMDX does not require PSM modeling.

Supported Model Types

Structure Diagrams	**Supported by openMDX**
- Class Diagram - Object Diagram - Component Diagram - Composite Structure Diagram - Package Diagram - Deployment Diagram	✓ (MOF compliant models only) --- --- --- --- ---
Behavior Diagrams	
- Use Case Diagram - Activity Diagram - State Machine Diagram	Could be supported by a plugin executing activity diagrams and state machines. Plugin is not implemented yet. The recently adopted **Business Process Modeling Specification (dtc/06-02-01)** seems to be more promising.
Interaction Diagrams	
- Sequence Diagram - Communication Diagram - Timing Diagram - Interaction Overview Diagram	---

MOF compliant Class Diagrams *[1]*



- Although MOF is designed as repository standard, all MOF mappings can be applied to business object models if they are MOF compliant.

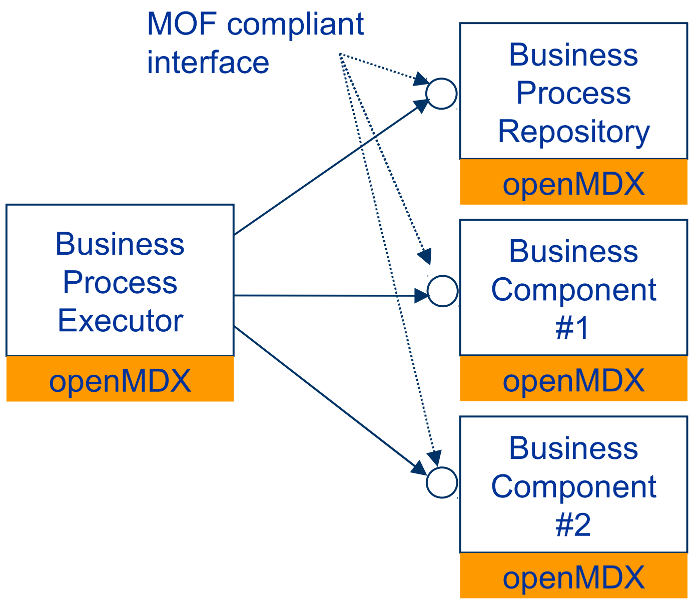
(*) C++ and C# mapping is not defined yet. They can be easily derived from the JMI mapping.
(**) WSDL mapping not defined yet. It can be derived from the MOF-to-IDL and IDL-to-WSDL mapping.

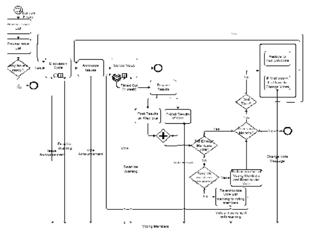
MOF compliant Class Diagrams *[2]*

UML Element		MOF Element	Supported by openMDX
Model	«metamodel»	Package	✓
ElementImport		Import	✓
Class		Class	✓
Attribute		Attribute	✓
Attribute	«reference»	Reference	✓
Operation		Operation	✓
Parameter		Parameter	✓
Exception		Exception	✓
Attribute (within Exception)		Parameter	✓
Association		Association	✓
AssociationEnd		AssociationEnd	✓
DataType		DataType	✓
DataValue		Constant	---
Constraint		Constraint	---
Generalization		Generalizes	✓
Tagged Value		Tag	✓
Qualifier		--	✓

Business Process Modeling *[1]*

Implementation Approach #1

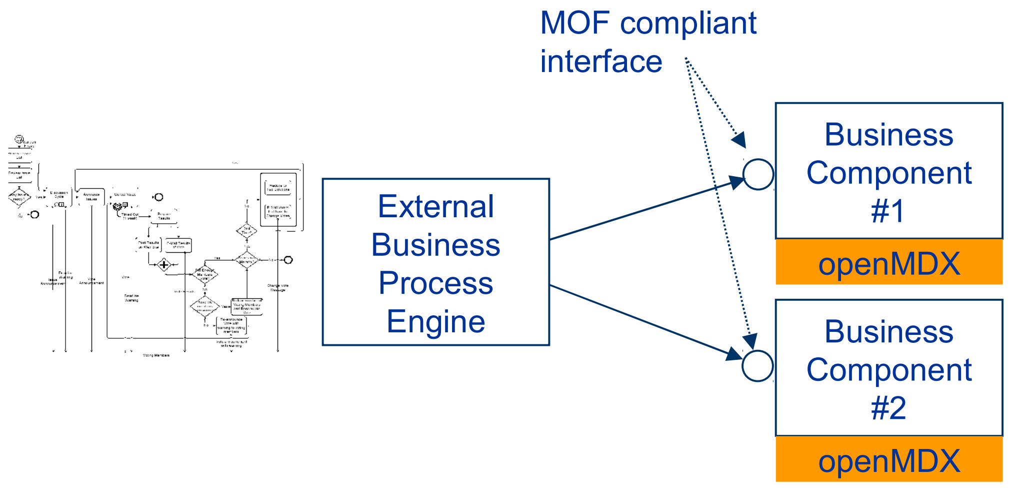




- Models are stored in a business process repository.
- They are executed by the business process executor (which is implemented as openMDX plugin).
- Most of the business logic can be expressed as platform-independent business proces model.

Business Process Modeling *[2]*

Implementation Approach #2



- External business process engine executes workflows and invokes functions of openMDX-based coomponents.

Modeling Class Diagrams

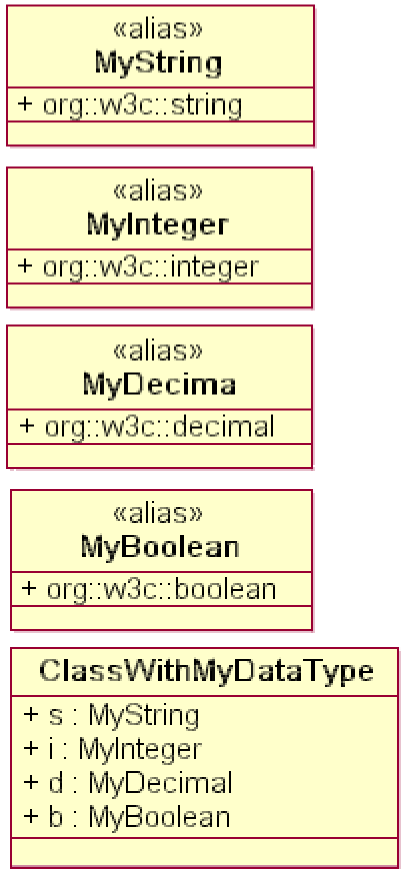
Primitive Types

CORBA IDL Types	string short int long boolean	float double decimal byte
W3C primitive and derived DataTypes (http://www.w3.org/TR/xmlschema-2/)	string integer long short byte boolean binary	decimal float double duration dateTime anyURI

- openMDX supports the W3C DataTypes:
  - W3C types better known than IDL types
  - more platform independent
  - easier mapping to XML

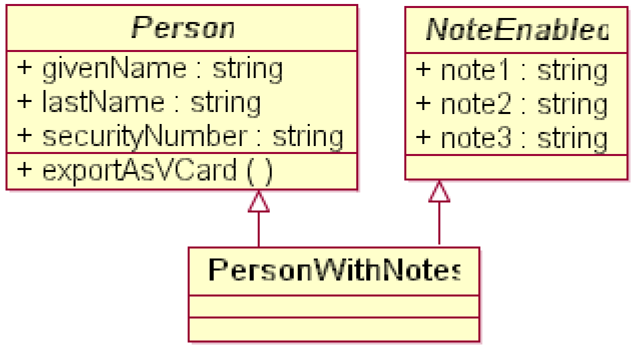
Alias Types

- Alias types allow to define user-defined datatypes.
- This allows to define a user-defined data type system.



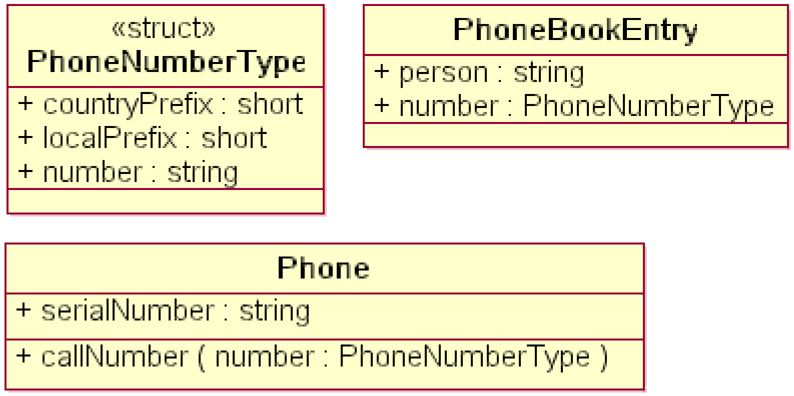
Classes

- Classes define object types.
- Implicit setters for read/write and getters for all attributes.
- Support for behaviour (operations and derived attributes)
- Support for associations
- Support for multiple inheritance.



Structure Types

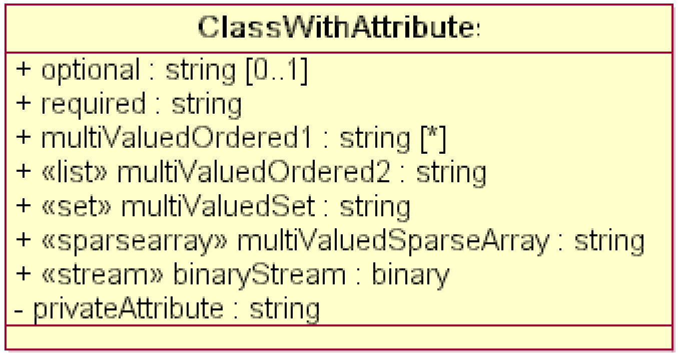
- Structures are value objects. Fields are read-only.
- No support for behaviour.
- No composite and shared associations.
- No inheritance.



	openMDX 1	openMDX 2
Can be used as operation parameters	YES	YES
Can be used as attribute types	NO	YES
Can be nested	NO	YES

Attributes, Fields

- Attributes are features of classifier types
- Fields are features of structure types
- Attributes / Fields have a type, multiplicity, changeability and visibility.
- Define attributes and fields instead of defining setter / getter operations.


ClassWithAttribute

+ optional : string [0..1]
+ required : string
+ multiValuedOrdered1 : string [*]
+ «list» multiValuedOrdered2 : string
+ «set» multiValuedSet : string
+ «sparsearray» multiValuedSparseArray : string
+ «stream» binaryStream : binary
- privateAttribute : string

Multiplicities	0..1: optional 1..1: required 0..*, «list»: multi-valued, ordered «set»: multi-valued, unordered «sparsearray»: multi-valued, ordered, sparsly set array «stream»: multi-valued, stream
Type	Primitive type Structure type (only with openMDX 2)
Changeability	changeable non-changeable
Visibility	public: features visible on interface and value objects. private: features visible on value objects only.

Associations

- Associations allow to ‚connect' classes / objects.
- The aggregation kind defines the semantics of the connection (from MOF and UML spec):
  - Composite: Coupling with life-cylce semantics.
  - Shared: Not supported by MOF. According to the UML specification the precise semantics varies by application area and modeler. openMDX supports shared associations and defines a semantics.

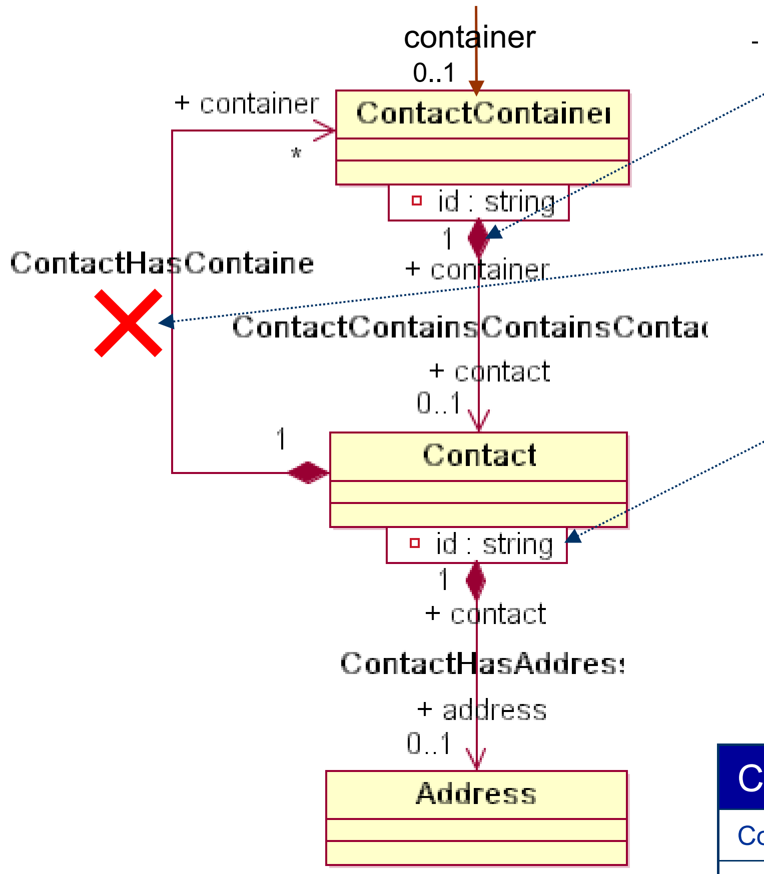
Associations – Composite *[1]*

- Strong coupling:
  - A composite relationship is asymmetrical, with one end denoting the “composite” or “whole” in the relationship and the other one denoting the “components” or “parts.”
  - An instance cannot be a component of more than one composite at a time, under any composite relationship.
  - An instance cannot be a component of itself, its components, its components’ components and so on under any composite relationship.
  - When a “composite” instance is deleted, all of its components under any composite relationship are also deleted, and all of the components’ components are deleted and so on.
  - The Composition Closure Rule: an instance cannot be a component of an instance from a different package extent.

Associations – Composite *[2]*

- openMDX specific:
  - *“An instance cannot be a component of more than one composite at a time, under any composite relationship”.*
    - openMDX enforces this rule at class-level: every class must have exactly one composite parent. The only exception are classes with stereotype «root». All other parent relationships must be modeled as ‘shared’ associations.
    - This rule allows to derive non-changeable, well-defined object identities from the model and vice versa.
  - A composite association must define a uniquely defining qualifier. This allows direct navigation from the composite object to a specific part.

Associations – Composite *[3]*



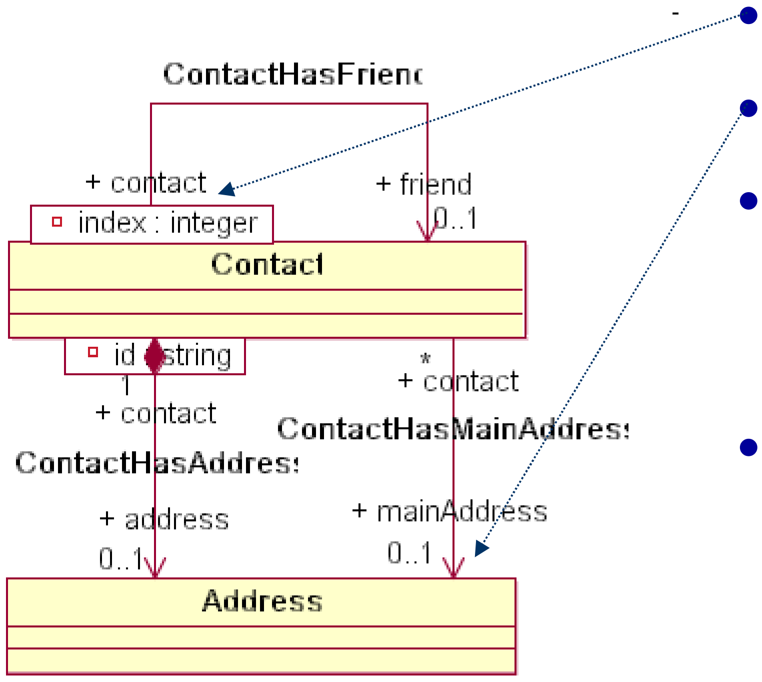
- A part must have exactly one composite
- A part can not be part of its composite
- A part must be uniquely referenced
- Instances have unique object identities (XRI), e.g.

Class	openMDX XRI pattern
ContactContainer	xri:@openmdx:<model>/.../**container**/<id>
Contact	xri:@openmdx:<model>/.../**container**/<id>/**contact**/<id>
Address	xri:@openmdx:<model>/.../**container**/<id>/**contact**/<id>/**address**/<id>

Associations – None *[1]*

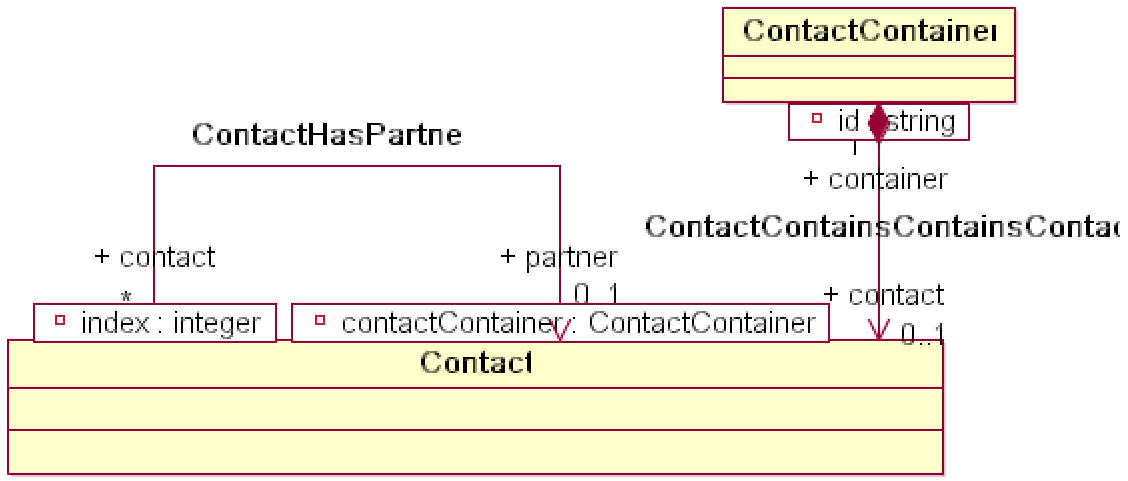
- Loose coupling:
  - There are no special restrictions on the multiplicity of the relationships.
  - There are no special restrictions on the origin of the instances in the relationships.
  - The relationships do not impact on the lifecycle semantics of related instances. In particular, deletion of an instance does not cause the deletion of related instances.

Associations – None *[2]*



- Multi-valued relationship
- Optional-value relationship
- Removal of a contact does not remove referenced contacts.
- Removal of a referenced object can lead to dangling references. With openMDX, referential integrity can be enforced by application logic.

Associations – None *[3]*

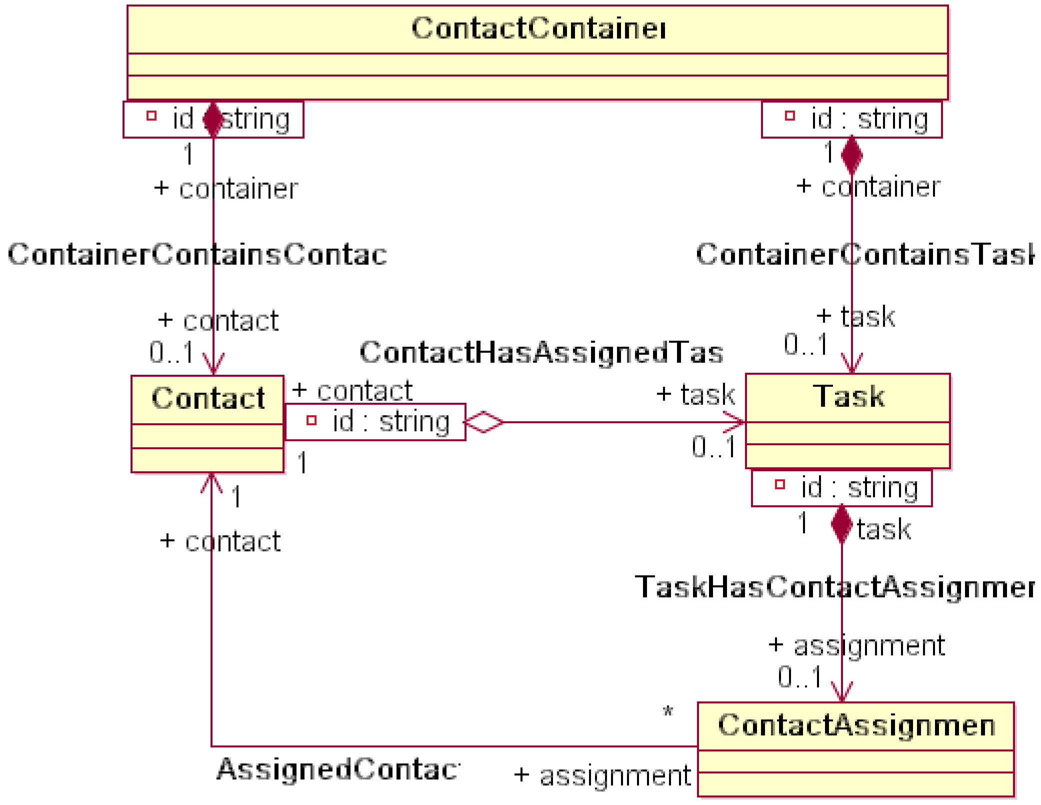


- Associations can also be navigable in both directions.
- Navigation of one end requires the index to navigate to the referenced object.
- Navigation of the other end requires the composite parent of the referenced object. The result is a collection of contacts which are referenced by the current contact.

Associations – Shared *[1]*

- Semantics defined by openMDX:
  - A shared relationship is asymmetrical, with one end denoting the “parent” in the relationship and the other one denoting the “components” or “parts.”
  - The „parent“ may or may not be the „composite“ of the part.
  - An instance can be a component of more than one parent at a time.
  - An instance cannot be a parent of itself, its components, its components’ components and so on under any shared relationship.
  - The life-cycle semantics is user-defined when a “shared” instance is deleted.
  - Composition Closure: an instance can be a parent of an instance from a different package extent.

Associations – Shared *[2]*

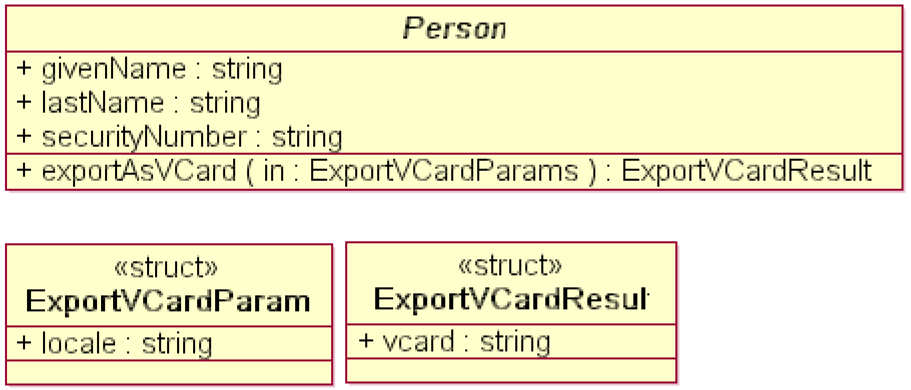


- Semantic of shared association ‚ContactHasAssignedTask' is user-defined.
- In this case the set of referenced objects are all tasks which are assigned (by ContactAssignments) to the exposing contact. This semantic should be documented.

Operations *[1]*

- An operation defines a dynamic feature that offers a service. The behavior of an operation is activated through the invocation of the operation.
- Defining an isQuery operation denotes that the behavior of the operation will not alter the state of the object. The state is the set of values of all of the object's class-scope and instance-scope structural features.
- An Operation, upon encountering an error or other abnormal condition, may raise exceptions.
- openMDX restrictions:
  - class-level operations

Operations *[2]*



- openMDX specifics:
  - parameters must be modeled as structure types
  - isQuery=false operations require an active unit of work before invocation

Model Constraints *[1]*

- The MOF constraints apply to all openMDX models. For a complete list of the MOF model constraints see MOF Specification 1.4, Section 3.9 (formal/02-04-03).
- For implementation reasons openMDX adds a few more constraints.

MOF Constraints *[2]*

ID	Description
MOF	
C-1	A ModelElement that is not a Package must have a container.
C-2	The attribute values of a ModelElement which is frozen cannot be changed.
C-6	A Generalizable Element cannot be its own direct or indirect supertype.
C-8	The names of the contents of a GeneralizableElement should not collide with the names of the contents of any direct or indirect supertype.
C-9	Multiple inheritance must obey the “Diamond Rule.”
C-10	If a Generalizable Element is marked as a “root,” it cannot have any supertypes.
C-19	Inheritance / generalization is not applicable to DataTypes.
C-59	A StructureType must contain at least one StructureField.
openMDX	
C-1004	Parameters must be structure types.
C-1011	Association end with aggregation not equal [none] requires a primitive type qualifier and multiplicity [0..1|1..1].
C-1013	Association end with aggregation [none] requires no qualifier or a qualifier [primitive with multiplicity 0..1|class with multiplicity 0..n].
C-1015	Association end1 with qualifier type class requires end2 with none or primitive qualifier.

openMDX Specifics

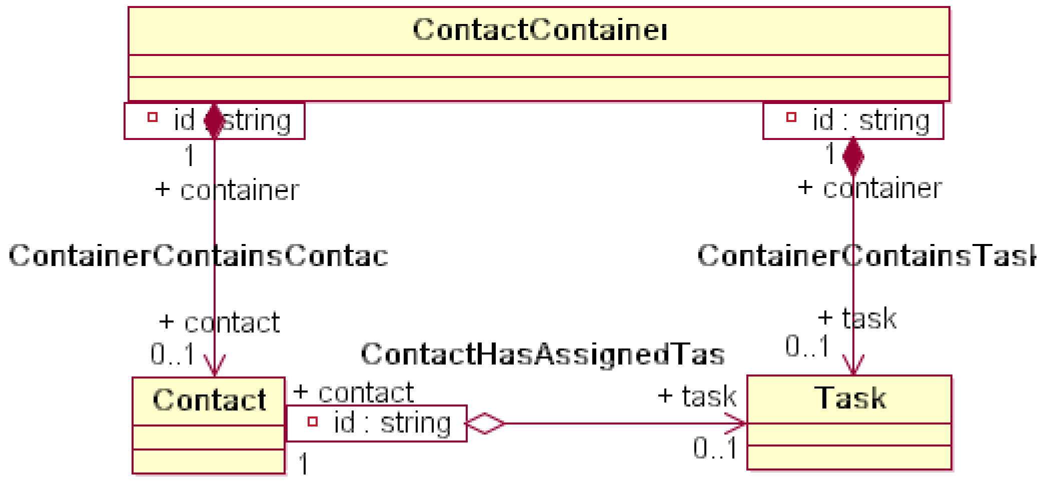
Overview

- Object identity and access path
- org:openmdx:base Package
- Object management (by providers)

Object identity and access path *[1]*

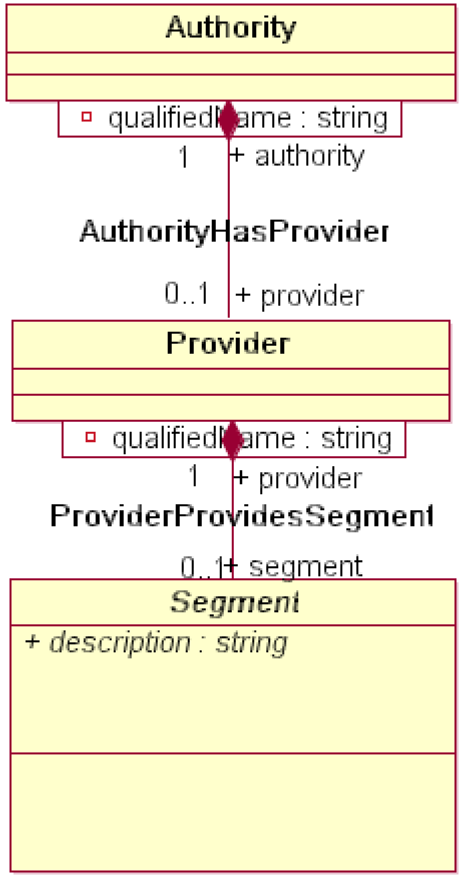
- Objects are accessed by their access paths. Valid access paths are:
  - object identities, i.e. the XRI which can be constructed from the composite assocations.
  - shared access paths, i.e. the XRIs which can be constructed from the shared associations.

Object identity and access path *[2]*



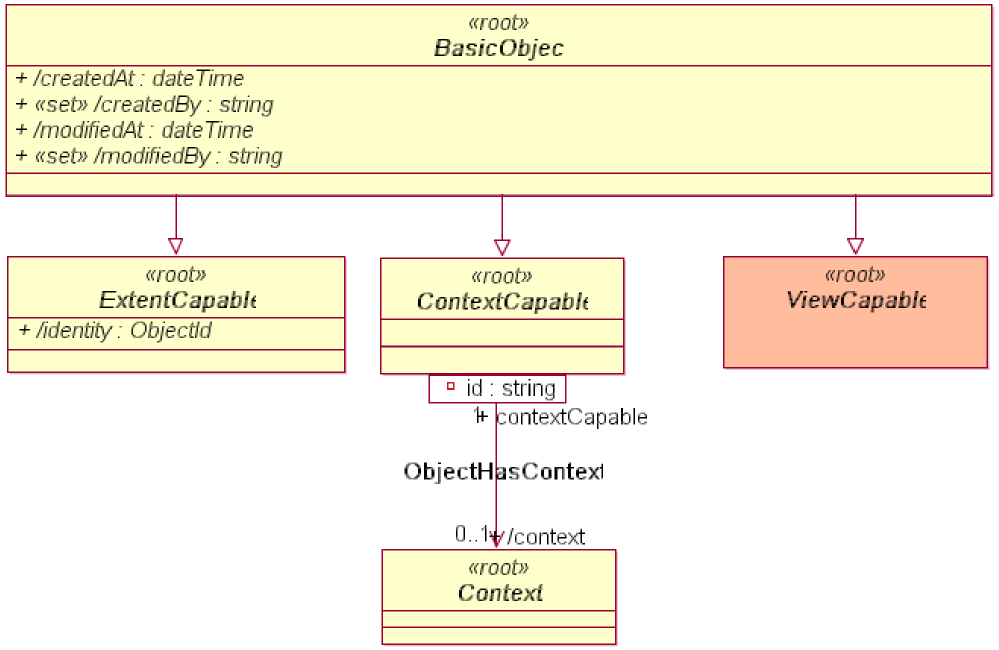
- Identity of Task objects:
  - xri:@openmdx:<model>/.../container/<id>/task/<id>
- Access Paths of Task objects:
  - xri:@openmdx:<model>/.../container/<id>/task/<id>
  - xri:@openmdx:<model>/.../container/<id>/contact/<id>/task/<id>

org:openmdx:base Package *[1]*



- The package solves the modeling bootstrap problem: *each class must have a composite parent.* The class Authority is the only class which is not required to have a composite parent.
- For implementation reasons the object space must be partitioned:
  - partioning by providers
  - partitioning by segments
- User-defined models should use as starting point the class org:openmdx:base:Segment.

org:openmdx:base Package *[2]*



- All user-defined business object classes should extend BasicObject.
- BasicObject adds features which are common to all business objects:
  - creation information
  - modification information
  - possibility to add contexts

Modeling best practices

Best practices *[1]*

- openMDX class models represent the business object model provided by the API of a business component. Good API design patterns are also valid for openMDX models:
  - model from the clients perspective
  - apply reusable patterns

Best practices *[2]*

- Models from third parties should be extended and not modified. Extensions which do not modify the original model are:
  - add subclasses in new model packages
  - existing classes can be references without modifying the orginal class