

EDOC2000

MOF/XMI Exposed Tutorial

Handouts

Presented by the

Cooperative Research Centre for
Enterprise Distributed Systems
Technology (DSTC)

<http://www.dstc.edu.au/MOF>

mofia@dstc.edu.au

Simple Trader MODL

The following MODL code was used as input to DSTC's dMOF product to automatically generate the simple trader IDL.

Table 1 Trader.modl

```
//
// Trader.modl
//
// The MODL for the Trader type system is declared within the scope of a
// Package. The Trader Package contains two Classes and two Associations.
package Trader {

    // The property_type Class contains two Attributes: the property type's name
    // and the type of its value.
    class property_type {
        attribute string name;
        attribute TypeCode value_type;
    };

    // The service_type Class contains two Attributes: the service type's name
    // and the interface type of the interface offering this service type.
    class service_type {
        attribute string name;
        attribute string interface_type;
    };

    // The has Association describes the relationship between a service type
    // and its properties. A single service type may have zero or more
    // properties. The list of properties associated with a service type is
    // unique but unordered.
    association has {
        end single service_type service;
        end set [0..*] of property_type property;
    };

    // The inherits Association describes the inheritance relationship between
    // service types. Zero or many service types may act as the base service
    // types for zero or many derived service types. The lists of base and
    // derived service types is unique but unordered.
    association inherits {
        end set [0..*] of service_type base;
        end set [0..*] of service_type QUOTE "derived";
    };
};
```

IDL Generated from Simple Trader MODL

The following IDL was generated from Trading.modl. Some editing has been performed to simplify presentation.

Table 2 Trader.idl

```
// Automatically generated using mof2idl 1.0-alpha2
#ifndef TRADER_INCLUDED
#define TRADER_INCLUDED
#include <Reflective.idl>

#include <Repository.idl>

//
// Trader.modl
//
// The MODL for the Trader type system is declared within the scope of a
// Package. The Trader Package contains two Classes and two Associations.
module Trader {
    interface TraderPackage;

    interface PropertyTypeClass;
    interface PropertyType;
    typedef sequence < PropertyType > PropertyTypeSet;

    interface ServiceTypeClass;
    interface ServiceType;
    typedef sequence < ServiceType > ServiceTypeSet;

    // The property_type Class contains two Attributes: the property type's name
    // and the type of its value.
    interface PropertyTypeClass :
        Reflective::RefObject
    {
        // all PropertyType including subclasses of PropertyType
        readonly attribute PropertyTypeSet all_of_type_property_type;

        // all PropertyType excluding subclasses of PropertyType
        readonly attribute PropertyTypeSet all_of_class_property_type;

        PropertyType create_property_type (
            in string name /* from Class PropertyType */,
            in TypeCode value_type /* from Class PropertyType */)
            raises (Reflective::MofError);
    }; // end of interface PropertyTypeClass

    interface PropertyType :
        PropertyTypeClass
    {
        string name ()
            raises (::Reflective::MofError);

        void set_name (
```

```

    in string new_value)
    raises (::Reflective::MofError);

TypeCode value_type ()
    raises (::Reflective::MofError);

void set_value_type (
    in TypeCode new_value)
    raises (::Reflective::MofError);

}; // end of interface PropertyType

// The service_type Class contains two Attributes: the service type's name
// and the interface type of the interface offering this service type.
interface ServiceTypeClass :
    Reflective::RefObject
{
    // all ServiceType including subclasses of ServiceType
    readonly attribute ServiceTypeSet all_of_type_service_type;

    // all ServiceType excluding subclasses of ServiceType
    readonly attribute ServiceTypeSet all_of_class_service_type;

    ServiceType create_service_type (
        in string name /* from Class ServiceType */,
        in string interface_type /* from Class ServiceType */)
        raises (Reflective::MofError);

}; // end of interface ServiceTypeClass

interface ServiceType :
    ServiceTypeClass
{
    string name ()
        raises (::Reflective::MofError);

    void set_name (
        in string new_value)
        raises (::Reflective::MofError);

    string interface_type ()
        raises (::Reflective::MofError);

    void set_interface_type (
        in string new_value)
        raises (::Reflective::MofError);

}; // end of interface ServiceType

// data types for Association Has
struct HasLink {
    // The has Association describes the relationship between a service type
    // and its properties. A single service type may have zero or more
    // properties. The list of properties associated with a service type is
    // unique but unordered.
    ServiceType service;

```

```
PropertyType property;
};

typedef sequence < HasLink > HasLinkSet;

interface Has : ::Reflective::RefAssociation
{
    // list of associated elements
    HasLinkSet all_has_links ()
        raises (::Reflective::MofError);

    boolean exists (
        in ServiceType service,
        in PropertyType property)
        raises (::Reflective::MofError);

    ServiceType service (
        in PropertyType property)
        raises (::Reflective::MofError);

    PropertyTypeSet property (
        in ServiceType service)
        raises (::Reflective::MofError);

    void add (
        in ServiceType service,
        in PropertyType property)
        raises (::Reflective::MofError);

    void modify_service (
        in ServiceType service,
        in PropertyType property,
        in ServiceType new_service)
        raises (::Reflective::NotFound, ::Reflective::MofError);

    void modify_property (
        in ServiceType service,
        in PropertyType property,
        in PropertyType new_property)
        raises (::Reflective::NotFound, ::Reflective::MofError);

    void remove (
        in ServiceType service,
        in PropertyType property)
        raises (::Reflective::NotFound, ::Reflective::MofError);

};

// data types for Association Inherits
struct InheritsLink {
    // The inherits Association describes the inheritance relationship between
    // service types. Zero or many service types may act as the base service
    // types for zero or many derived service types. The lists of base and
    // derived service types is unique but unordered.
    ServiceType base;
    ServiceType derived;
};
```

```
};

typedef sequence < InheritsLink > InheritsLinkSet;

interface Inherits : ::Reflective::RefAssociation
{
  // list of associated elements
  InheritsLinkSet all_inherits_links ()
  raises (::Reflective::MofError);

  boolean exists (
    in ServiceType base,
    in ServiceType derived)
  raises (::Reflective::MofError);

  ServiceTypeSet base (
    in ServiceType derived)
  raises (::Reflective::MofError);

  ServiceTypeSet derived (
    in ServiceType base)
  raises (::Reflective::MofError);

  void add (
    in ServiceType base,
    in ServiceType derived)
  raises (::Reflective::MofError);

  void modify_base (
    in ServiceType base,
    in ServiceType derived,
    in ServiceType new_base)
  raises (::Reflective::NotFound, ::Reflective::MofError);

  void modify_derived (
    in ServiceType base,
    in ServiceType derived,
    in ServiceType new_derived)
  raises (::Reflective::NotFound, ::Reflective::MofError);

  void remove (
    in ServiceType base,
    in ServiceType derived)
  raises (::Reflective::NotFound, ::Reflective::MofError);
};

interface TraderPackageFactory
{
  TraderPackage create_trader_package ()
  raises (::Reflective::MofError);
};

interface TraderPackage :
  ::Reflective::RefPackage
{
```

```
// Contained Class references
readonly attribute PropertyTypeClass property_type_ref;
readonly attribute ServiceTypeClass service_type_ref;

// Contained Association references
readonly attribute Has has_ref;
readonly attribute Inherits inherits_ref;
};

interface TraderManager :
  TraderPackageFactory,
  ::Repository::BaseManager {
};

}; // end of module Trader

#endif
```

Extended Trader MODL

The following MODL code was used as input to DSTC's dMOF product to automatically generate the extended trader IDL.

Table 3 Trader.modl

```
//  
// Trader.modl  
//  
package Trader {  
  
  abstract class base_type {  
    attribute string name;  
  };  
  
  class property_type: base_type {  
    attribute TypeCode value_type;  
  };  
  
  class service_type: base_type {  
    attribute string interface_type;  
  
    reference props to property of has;  
  
    exception duplicate_names {  
      set [2..*] of property_type duplicates;  
    };  
  
    set [0..*] of property_type all_props()  
      raises (duplicate_names);  
  };  
  
  association has {  
    end single service_type service;  
    composite end set [0..*] of property_type property;  
  };  
  
  association inherits {  
    end set [0..*] of service_type base;  
    end set [0..*] of service_type QUOTE "derived";  
  };  
};
```

IDL Generated from Extended Trader MODL

The following IDL was generated from Trading.modl. Some editing has been performed to simplify presentation.

Table 4 Trader.idl

```
// Automatically generated using mof2idl 1.0-alpha2
#ifndef TRADER_INCLUDED
#define TRADER_INCLUDED
#include <Reflective.idl>

#include <Repository.idl>
module Trader {
    interface TraderPackage;

    interface BaseTypeClass;
    interface BaseType;
    typedef sequence < BaseType > BaseTypeSet;

    interface PropertyTypeClass;
    interface PropertyType;
    typedef sequence < PropertyType > PropertyTypeSet;

    interface ServiceTypeClass;
    interface ServiceType;
    typedef sequence < ServiceType > ServiceTypeSet;

    interface BaseTypeClass :
        Reflective::RefObject
    {
        // all BaseType including subclasses of BaseType
        readonly attribute BaseTypeSet all_of_type_base_type;
    }; // end of interface BaseTypeClass

    interface BaseType :
        BaseTypeClass
    {
        string name ()
            raises (::Reflective::MofError);

        void set_name (
            in string new_value)
            raises (::Reflective::MofError);
    }; // end of interface BaseType

    interface PropertyTypeClass :
        BaseTypeClass
    {
        // all PropertyType including subclasses of PropertyType
        readonly attribute PropertyTypeSet all_of_type_property_type;

        // all PropertyType excluding subclasses of PropertyType
    }; // end of interface PropertyTypeClass
}; // end of module Trader
```

```
readonly attribute PropertyTypeSet all_of_class_property_type;

PropertyType create_property_type (
  in string name /* from Class BaseType */,
  in TypeCode value_type /* from Class PropertyType */)
  raises (Reflective::MofError);

}; // end of interface PropertyTypeClass

interface PropertyType :
  BaseType,
  PropertyTypeClass
{
  TypeCode value_type ()
  raises (::Reflective::MofError);

  void set_value_type (
    in TypeCode new_value)
  raises (::Reflective::MofError);

}; // end of interface PropertyType

interface ServiceTypeClass :
  BaseTypeClass
{
  // all ServiceType including subclasses of ServiceType
  readonly attribute ServiceTypeSet all_of_type_service_type;

  // all ServiceType excluding subclasses of ServiceType
  readonly attribute ServiceTypeSet all_of_class_service_type;

  exception DuplicateNames {
    PropertyTypeSet duplicates;
  };

  ServiceType create_service_type (
    in string name /* from Class BaseType */,
    in string interface_type /* from Class ServiceType */)
    raises (Reflective::MofError);

}; // end of interface ServiceTypeClass

interface ServiceType :
  BaseType,
  ServiceTypeClass
{
  string interface_type ()
  raises (::Reflective::MofError);

  void set_interface_type (
    in string new_value)
  raises (::Reflective::MofError);

  PropertyTypeSet props ()
  raises (::Reflective::MofError);
```

```
void set_props (
  in PropertyTypeSet new_value)
  raises (::Reflective::MofError);

void add_props ( in PropertyType new_element)
  raises (::Reflective::MofError);

void modify_props (
  in PropertyType old_element,
  in PropertyType new_element)
  raises (::Reflective::NotFound, ::Reflective::MofError);

void remove_props (
  in PropertyType old_element)
  raises (::Reflective::NotFound, ::Reflective::MofError);

PropertyTypeSet all_props ()
  raises (
    ServiceTypeClass::DuplicateNames,
    ::Reflective::MofError);

}; // end of interface ServiceType

// data types for Association Has
struct HasLink {
  ServiceType service;
  PropertyType property;
};

typedef sequence < HasLink > HasLinkSet;

interface Has : ::Reflective::RefAssociation
{
  // list of associated elements
  HasLinkSet all_has_links ()
  raises (::Reflective::MofError);

  boolean exists (
    in ServiceType service,
    in PropertyType property)
  raises (::Reflective::MofError);

  ServiceType service (
    in PropertyType property)
  raises (::Reflective::MofError);

  PropertyTypeSet property (
    in ServiceType service)
  raises (::Reflective::MofError);

  void add (
    in ServiceType service,
    in PropertyType property)
  raises (::Reflective::MofError);

  void modify_service (
```

```
in ServiceType service,
in PropertyType property,
in ServiceType new_service)
raises (::Reflective::NotFound, ::Reflective::MofError);

void modify_property (
in ServiceType service,
in PropertyType property,
in PropertyType new_property)
raises (::Reflective::NotFound, ::Reflective::MofError);

void remove (
in ServiceType service,
in PropertyType property)
raises (::Reflective::NotFound, ::Reflective::MofError);

};

// data types for Association Inherits
struct InheritsLink {
ServiceType base;
ServiceType derived;
};

typedef sequence < InheritsLink > InheritsLinkSet;

interface Inherits : ::Reflective::RefAssociation
{
// list of associated elements
InheritsLinkSet all_inherits_links ()
raises (::Reflective::MofError);

boolean exists (
in ServiceType base,
in ServiceType derived)
raises (::Reflective::MofError);

ServiceTypeSet base (
in ServiceType derived)
raises (::Reflective::MofError);

ServiceTypeSet derived (
in ServiceType base)
raises (::Reflective::MofError);

void add (
in ServiceType base,
in ServiceType derived)
raises (::Reflective::MofError);

void modify_base (
in ServiceType base,
in ServiceType derived,
in ServiceType new_base)
raises (::Reflective::NotFound, ::Reflective::MofError);
```

```
void modify_derived (
    in ServiceType base,
    in ServiceType derived,
    in ServiceType new_derived)
    raises (::Reflective::NotFound, ::Reflective::MofError);

void remove (
    in ServiceType base,
    in ServiceType derived)
    raises (::Reflective::NotFound, ::Reflective::MofError);

};

interface TraderPackageFactory
{
    TraderPackage create_trader_package ()
        raises (::Reflective::MofError);
};

interface TraderPackage :
    ::Reflective::RefPackage
{
    // Contained Class references
    readonly attribute BaseTypeClass base_type_ref;
    readonly attribute PropertyTypeClass property_type_ref;
    readonly attribute ServiceTypeClass service_type_ref;

    // Contained Association references
    readonly attribute Has has_ref;
    readonly attribute Inherits inherits_ref;
};

interface TraderManager :
    TraderPackageFactory,
    ::Repository::BaseManager {
};

}; // end of module Trader

#endif
```

XMI for Extended Trader

The following XMI contains a description of two Trader service types.

Table 5 Trader.xmi

```
<!DOCTYPE Trader SYSTEM "Trader.dtd">
<XMI version='1.1' xmlns:Trader="com.dstc/Trader">

  <XMI.header>
    <XMI.model xmi.name="Trader1" href="Trader1.xml"/>
    <XMI.metamodel xmi.name="Trader" href="Trader.xml"/>
  </XMI.header>

  <XMI.content>

    <Trader:Trader xmi.id="Trader_1234">

      <Trader:ServiceType name="Savings_Account" XMI.Id="Service_1"
        interface_type="::Banking::Savings">

        <Trader:ServiceType.props>

          <Trader:PropertyType name="interest_rate" XMI.Id="Property_1" value_type="float"/>

        </Trader:ServiceType.props>

      </Trader:ServiceType>

      <Trader:ServiceType name="Cheque_Acount" XMI.Id="Service_2"
        interface_type="::Banking:Cheque">

        <Trader:ServiceType.props>

          <Trader:PropertyType name="overdraft" XMI.Id="Property_2" value_type="boolean"/>

        </Trader:ServiceType.props>

      </Trader:ServiceType>

      <Trader:Inherits>

        <Trader:ServiceType xmi.idref="Service_1"/>

        <Trader:ServiceType xmi.idref="Service_2"/>

      </Trader:Inherits>

    </Trader:Trader>

  </XMI.content>

</XMI>
```

DTD Generated from Extended Trader MODL

The following DTD was generated for the Trader example. Some editing has been performed to simplify presentation.

Table 6 Trader.dtd

```
<!-- Fixed content CORBA elements here -->

<!ELEMENT XMI.content (Trader | XMI.extension)* >

<!ELEMENT Trader ((ServiceType | PropertyType | Inherits | XMI.extension)* )>
<!ATTLIST Trader %xmi.element.att; >

<!ELEMENT BaseType.name (PCDATA | XMI.extension)* >

<!ELEMENT ServiceType
  (BaseType.name, ServiceType.interface_type, (PropertyType)*, (XMI.extension)* ) >
<!ATTLIST ServiceType %xmi.element.att; %xmi.link.att;
  name #CDATA #IMPLIED
  interface_type #CDATA #IMPLIED>

<!ELEMENT ServiceType.interface_type (PCDATA | XMI.extension)* >

<!ELEMENT PropertyType (BaseType.name, PropertyType.value_type, (XMI.extension)*)>
<!ATTLIST PropertyType %xmi.element.att; %xmi.link.att; >

<!ELEMENT PropertyType.value_type (PCDATA | XMI.extension)*>

<!ELEMENT Inherits (ServiceType, ServiceType)>
```