MOF/XMI Exposed

Kerry Raymond

mofia@dstc.edu.au

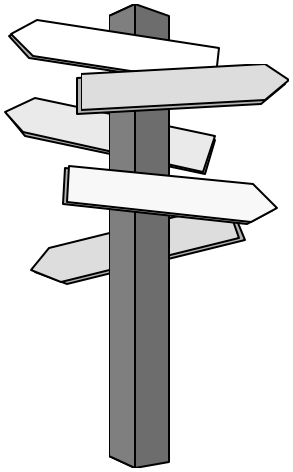
http://www.dstc.edu.au/MOF/

CRC for Enterprise Distributed Systems Technology (DSTC)

University of Queensland,

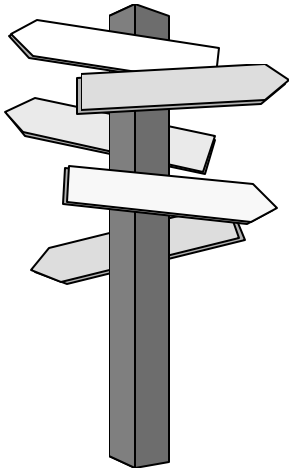
Brisbane, Australia

The Agenda



- The Meta-Object Facility in a nutshell
- Motivation for a MOF
- What is it and how do I use it?
- The MOF Model: The Basics
- An Example: Trader type system
- Generating IDL for type systems

The Agenda ... continued



- The MOF Model: Advanced
- Extending the example
- Generated IDL for the extended example
- Reflective IDL
- Building a Complete Meta-information service
- Standardisation of the MOF
- Summary and questions

What is Meta-Information?

- Information which describes other information
  - database schemas
  - programming language types
  - UML models
- Type systems

In a Nutshell!

- The Meta-Object Facility (or MOF) is an open, standards-based tool for meta-information management
- CORBA Common Facility
- Well-founded meta-information model closely aligned with UML core
- An OMG adopted technology with a standardised mapping to CORBA IDL

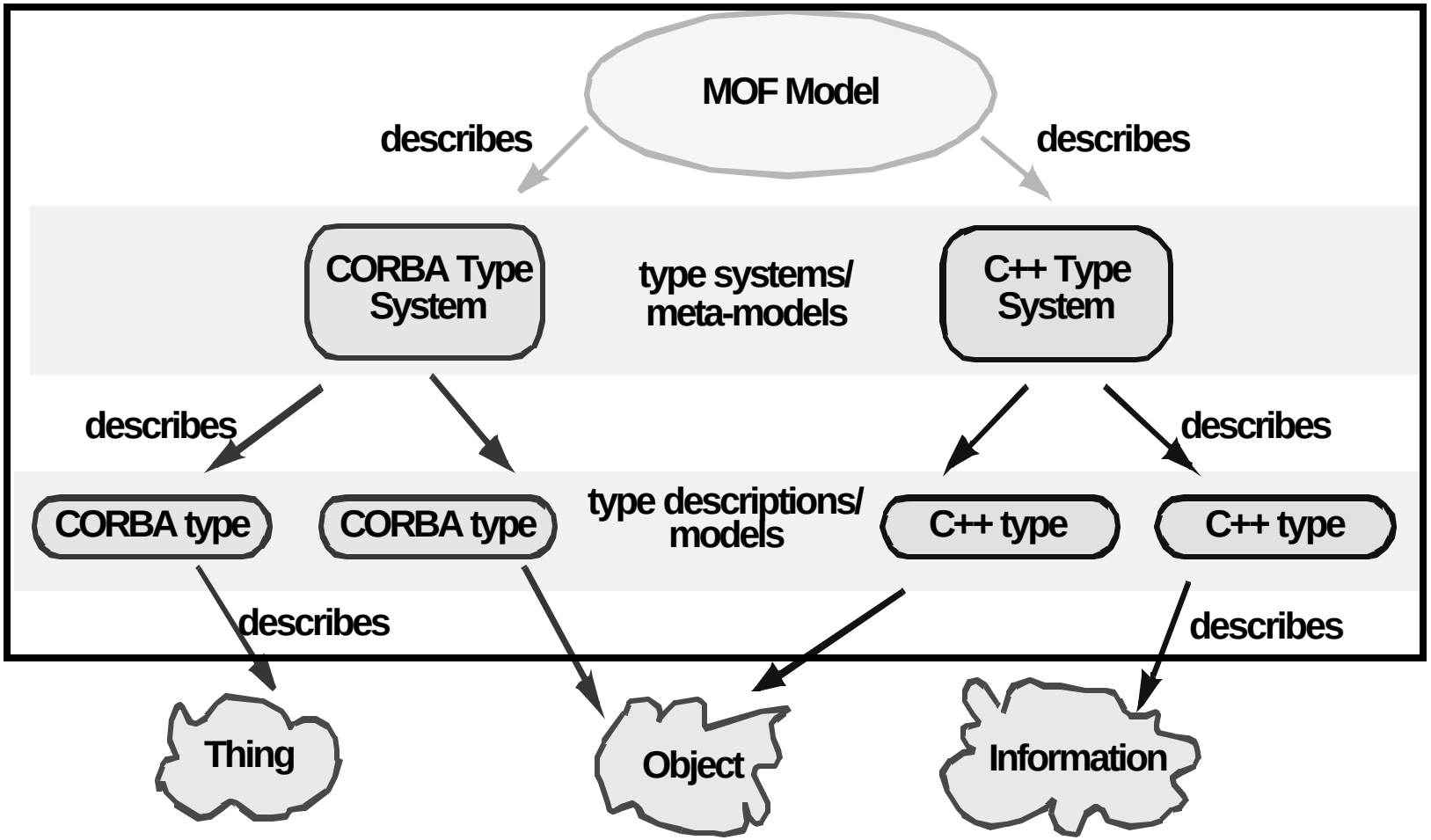
OMG Motivation for MOF

- OMG's problem: How to manage meta-information within a CORBA system?
- Interface Repository does part of the job but has shortcomings:
  - Only understands the CORBA interface type system
  - no relationships
- The adopted solution (MOF) addresses meta-information management in CORBA but is general enough for non-CORBA environments

MOF Specification

- A semantically well-defined model for describing type systems (ie. meta-models) called the MOF Model
- CORBA interfaces to create, read, update and delete a type system description
- A semantically well-defined model for describing types (models) and
- CORBA interfaces to create, read, update and delete type descriptions

Hierarchy of Descriptions



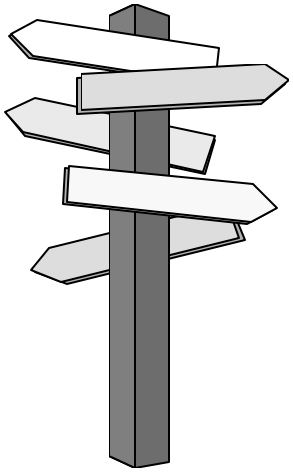
What is the MOF?

- A MOF implementation consists of:
  - a server implementing the MOF model's interfaces
  - a tool to populate the server with type system descriptions
  - a tool which generates CORBA IDL for the interfaces which create and manipulate types within a type system

How do I use the MOF?

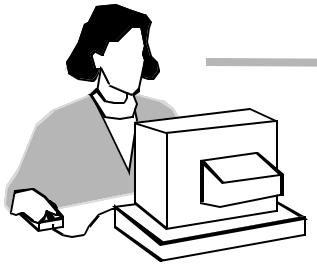
- Describe your type system
- Define your types:
  - Using the operations declared on the IDL generated for the specific type system
- Use your types
  - Software development, type management, information management, data warehouse management, ...

The Agenda



- *The Meta-Object Facility in a nutshell*
- *Motivation for a MOF*
- *What is it and how do I use it?*
- **The MOF Model: The Basics**
- An Example: Trader type system
- Generating IDL for type systems

First Describe the Type System



Describe the type system

Meta-Object Facility (MOF)

The MOF Model: The Basics

- The MOF model is the “language” for describing type systems
- UML core alignment
- Described as collections of:
  - *Class*: describes a concept in a type system
  - *Association*: describes a relationship between concepts
  - All contained in a *Package*

Describing type systems: *Class*

- Classes are type system concepts, e.g.
  - “interface” and “operation” in CORBA
  - “class” and “constructor” in C++
- A Class can contain MofAttributes, e.g.
  - Return type of an operation in CORBA
  - Visibility (public/protected/private) of a data member in C++

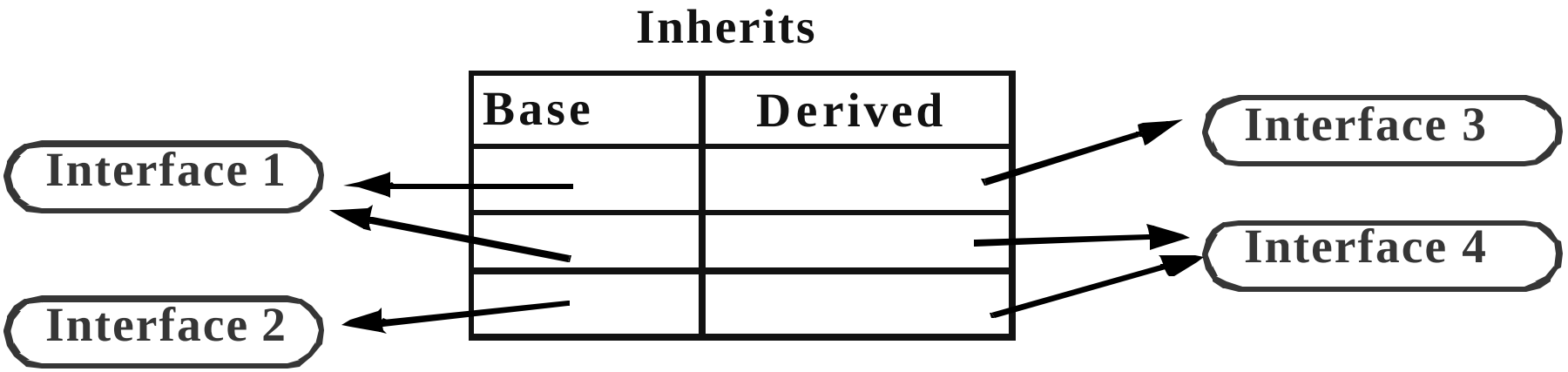
Describing type systems: *Association*

- Associations between Classes correspond to relationships between concepts in a type system, e.g.
  - inheritance between interfaces in CORBA
  - equivalence between a CORBA interface and a C++ implementation class

Describing type systems: *Association End*

- Associations contain 2 *AssociationEnds* which define the roles, their “type” and their cardinality in the Association, e.g.
  - the “inherits” Association has “base” and “derived” AssociationEnds which are both “interfaces”, a many-to-many Association

Associations are Tables

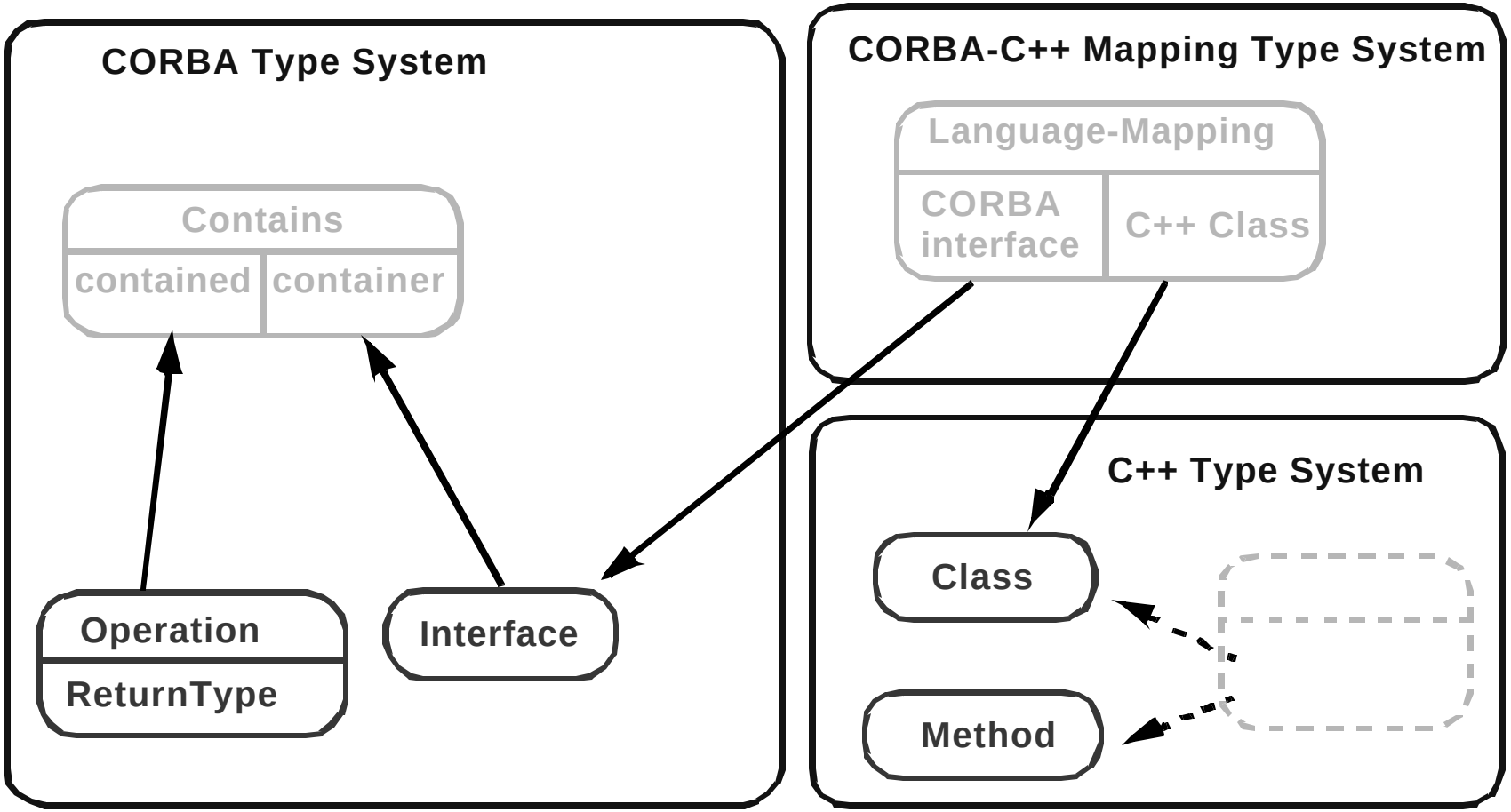


- Associations provide a view of a set of relationships between Classes as a table, e.g.
  - Interface3 inherits from Interface1
  - Interface4 inherits from Interface1 and Interface2
- The rows in the table are unique

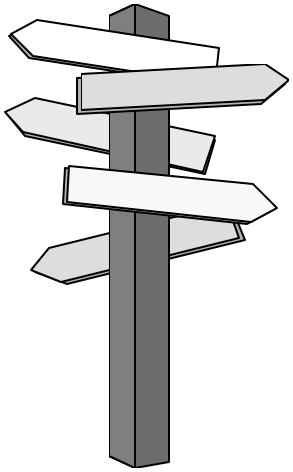
Describing type systems: *Package*

- A Package represents a type system consisting of concepts and relationships between concepts
- The concepts are represented by Classes and relationships by Associations
- Examples of Packages are the “C++” Package or the “CORBA” Package

Classes, Associations and Packages



The Agenda

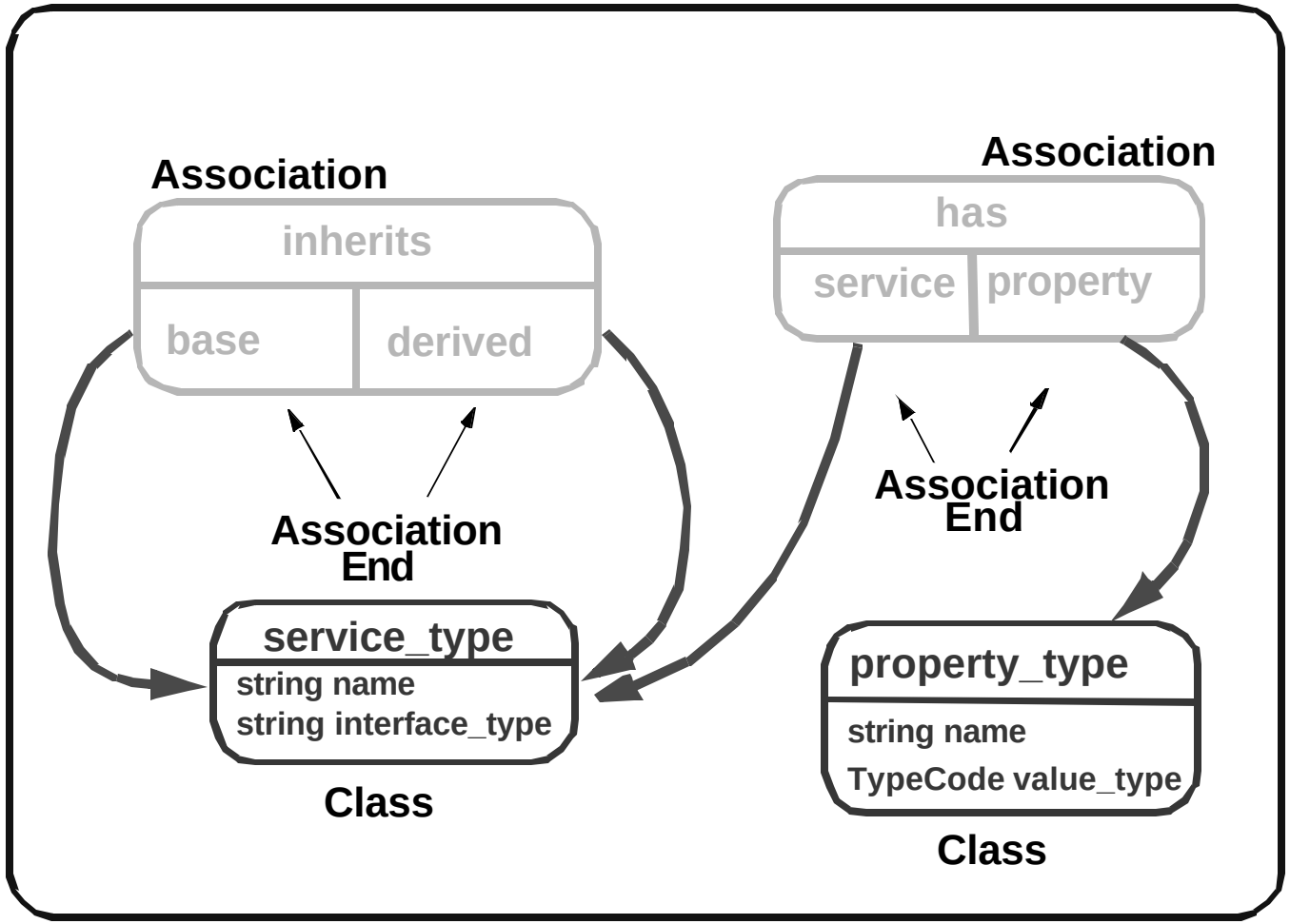


- *The Meta-Object Facility in a nutshell*
- *Motivation for a MOF*
- *What is it and how do I use it?*
- *The MOF Model: The Basics*
- **An Example: Trader type system**
- Generating IDL for type systems

An Example: Trader

- Trader type system consists of:
  - Service types:
    - an interface type
    - zero or more property types
  - Interface types
    - the interface implementing the service
  - Properties
    - name/value pair describing other characteristics of the service

Trader Type System



MODL: Meta-Object Definition Language

- A textual notation to describe type systems using concepts from the MOF Model
- Primary means by which DSTC's prototype servers are populated with type system descriptions because:
  - Familiar MOF Model concepts
  - Ease of use
  - Syntax similar to CORBA IDL
- But, sometimes the syntax is "clunky"

MODL for Trader

```
package Trader {
   class property_type {...};
   class service_type {...};

   association has {...};
   association inherits {...};
};
```

MODL for Trader: Classes

```
class property_type {
    attribute string name;
    attribute TypeCode value_type;
};

class service_type {
    attribute string name;
    attribute string interface_type;
};
```

MODL for Trader: Associations

association has **{**

end single service_type service**;**

end set [0**..***] **of** property_type property**;**

};

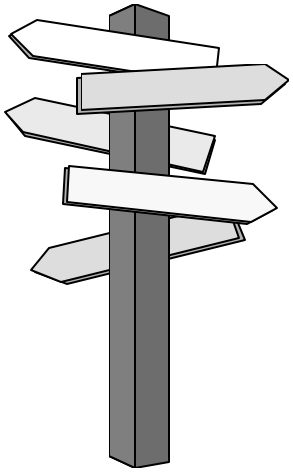
association inherits **{**

end set [0**..***] **of** service_type base**;**

end set [0**..***] **of** service_type QUOTE “derived”**;**

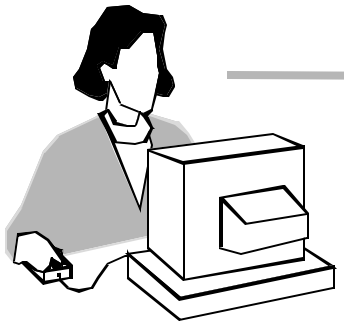
};

The Agenda



- *The Meta-Object Facility in a nutshell*
- *Motivation for a MOF*
- *What is it and how do I use it?*
- *The MOF Model: The Basics*
- *An Example: Trader type system*
- **Generating IDL for type systems**

Generating IDL



Describe the type system

Meta-Object Facility (MOF)

IDL Generator

Specific IDL

Specific Type Repository

IDL Generation

- CORBA's standard mapping from type system descriptions to IDL
- The generated IDL declares the interfaces for a type repository for that specific type system
- IDL generation rules ensure interoperability between generated repositories

Mapping for Class

- A Class is mapped onto an interface:
  - the Class's Attributes are mapped onto "get" and "set" operations

Example: ServiceType Class

```
interface ServiceType: ... {

    string name ()
        raises (...);

    void set_name (string new_value)
        raises (...);

    ...
};
```

Example: ServiceType Class

```
interface ServiceType: ... {
    ...
    string interface_type ()
        raises (...);

    void set_interface_type (in string new_value)
        raises (...);
};
```

Example: PropertyType Class

```
interface PropertyType: ... {

    string name ()
        raises (...);

    void set_name (string new_value)
        raises (...);

    ...
};
```

Example: PropertyType Class

```
interface PropertyType: ... {
    ...
    TypeCode value_type ()
        raises (...);

    void set_value_type (
        in TypeCode new_value)
        raises (...);
};
```

Mapping for Association

- An Association maps onto a data type and an Association interface:
  - The *Link* struct represents a row (pair) in the Association table
  - A set of these structs represents rows in the Association table

Example: Has Link

```
struct HasLink {
   ServiceType     service;
   PropertyType    property;
};

typedef sequence <HasLink> HasLinkSet;
...
```

Mapping for Association (cont)

- An interface containing:
  - an operation returning all Association links
  - an operation to check if a link between two Association Ends exists within this Association
  - an operation returning the value(s) associated with a particular Association End value
  - operations to add, modify and remove the value(s) of Association links

Example: Has Association

```
interface Has: ... {

    HasLinkSet all_has_links ()
            raises(...);

    boolean exists (
            in ServiceType service,
            in PropertyType property)
             raises(...);
    ...
};
```

Example: Has Association

```
interface Has: ... {
    ...
    PropertyTypeSet property (
        in ServiceType service)
        raises(...);

    ServiceType service (
        in PropertyType property)
        raises(...);

    ...
};
```

Example: Has Association

```
interface Has: ... {
    ...
    void add (
        in ServiceType service,
        in PropertyType property) raises (...);
    void remove(
        in ServiceType service,
        in PropertyType property) raises (...);
    ...
};
```

Example: Has Association

```
interface Has: ... {
    ...
    void modify_service (
        in ServiceType service,
        in PropertyType property,
        in ServiceType new_service)
        raises(...);

    void modify_property(...) raises(...);
};
```

Example: Inherits Association

```
struct InheritsLink {
   ServiceType base;
   ServiceType derived;
};

typedef sequence <InheritsLink>
   InheritsLinkSet;
...
```

Example: Inherits Association

```
interface Inherits: ... {

   InheritsLinkSet all_inherits_links ();

   boolean exists (
         in ServiceType base,
         in ServiceType derived)
         raises(...);
   ...
};
```

Example: Inherits Association

```
interface Inherits: ... {
    ...
    ServiceTypeSet derived (
        in ServiceType base)
        raises(...);

    ServiceTypeSet base (
        in ServiceType derived)
        raises(...);
    ...
};
```

Example: Inherits Association

```
interface Inherits: ... {
   ...
   void modify_base (
        in ServiceType base,
        in ServiceType derived,
        in ServiceType new_base) raises (...);

   void modify_derived(...) raises (...);
   ...
};
```

Example: Inherits Association

```
interface Inherits: ... {
    ...
    void add (
        in ServiceType base,
        in ServiceType derived) raises (...);

    void remove (
        in ServiceType base,
        in ServiceType derived) raises (...);
};
```

Mapping for Class

- A Class is really mapped onto **two** interfaces:
  - The interface we described earlier **and**
  - A *Class* interface containing:
    - an operation to create an instance of the Class
    - an attribute giving all instances of the Class including subtypes (all_of_type)
    - an attribute giving all instances of the Class excluding subtypes (all_of_class)

Example: PropertyType Class

```
interface PropertyTypeClass: ... {
    ...
    PropertyType create_property_type (
        in string name,
        in TypeCode value_type)
        raises (...);
};
```

Example: PropertyType Class

```
interface PropertyTypeClass: ... {

   readonly attribute PropertyTypeSet
        all_of_type_property_type;

   readonly attribute PropertyTypeSet
        all_of_class_property_type;

   ...
};
```

Mapping for Package

- A Package maps onto a module containing:
  - data types for multi-valued elements declared within the Package
  - a Package factory interface containing:
    - an operation to create an instance of the Package
  - a Package instance interface containing:
    - Class *Class* interfaces and Association instance interfaces

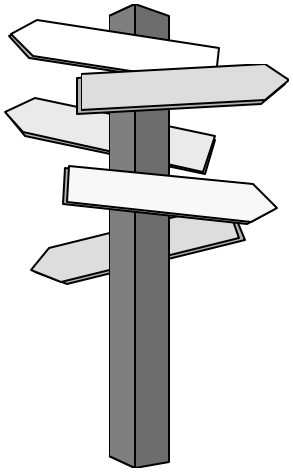
Example: Trader Package

```
module Trader {

   interface TraderPackageFactory {

         TraderPackage create_trader_package()
                  raises (...);
   };

   interface TraderPackage {...};
};
```

Example: Trader Package

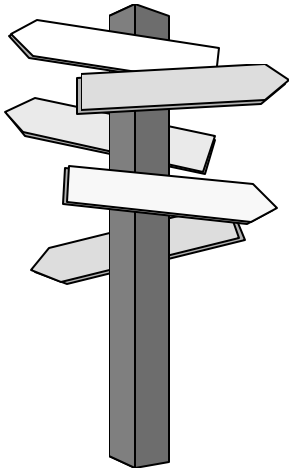
```
interface TraderPackage {
    readonly attribute ServiceTypeClass
            service_type_ref;

    readonly attribute PropertyTypeClass
            property_type_ref;

    readonly attribute Has has_ref;

    readonly attribute Inherits inherits_ref;
};
```

The Agenda



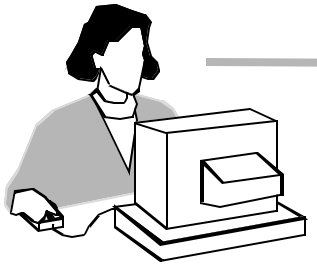
- *The Meta-Object Facility in a nutshell*
- *Motivation for a MOF*
- *What is it and how do I use it?*
- *The MOF Model: The Basics*
- *An Example: Trader type system*
- *Generating IDL for type systems*

The Agenda ... continued



- **The MOF Model: Advanced**
- Extending the example
- Generated IDL for the extended example
- Reflective IDL
- Building a Complete Meta-information service
- Standardisation of the MOF
- Summary and questions

Describing Type Systems



Describe the type system

Meta-Object Facility (MOF)

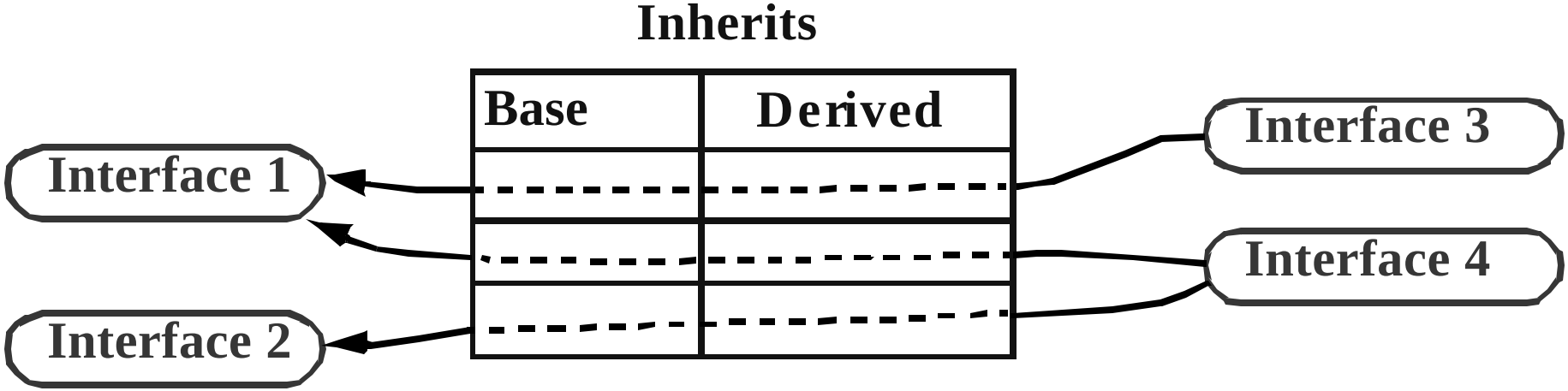
More on the MOF Model

- UML alignment provides the MOF Model with a rich “language” for describing type systems, e.g.
  - References
  - Operations and Exceptions
  - Class and Package inheritance
  - Abstract Classes
  - Multiplicity
  - Aggregation/Composition/Containment

Describing type systems: *References*

- Classes can be aware (or not aware) of their participation in Associations, e.g.
  - in CORBA, a derived interface is aware of the base interface, but not vice versa
- A Class is aware of its participation if it contains a Reference to an Association End, e.g.
  - the “interface” Class has a “bases” Reference to the “base” Association End in the “inherits” Association

Navigation by Reference



- References provide a Class with a view of an Association from its perspective, e.g.
  - Interface3 is “aware” that its base interface is Interface1
  - Interface4 is “aware” that its base interfaces are Interface1 and Interface2

Describing type systems: *Operation* and *Exception*

- Operations may declared on a Class
  - provides type system designers with additional functionality for their type system
  - Operations may throw user-defined exceptions or standard MOF exceptions

Describing type systems: Inheritance

- The MOF Model allows inheritance for Class and Package:
  - Classes inherit Attributes, Operations and References from super-Classes, e.g.
    - in C++, a “discriminated union” Class would inherit from a “union” Class
  - Packages inherit the contents of super-Packages in a similar manner
  - Support for multiple inheritance

Describing type systems: Abstract Classes

- Abstract Classes:
  - Cannot be instantiated
  - But can be inherited from
  - Convenient way to collect common subsets of features

Multiplicity

- Cardinality
  - single (1..1), optional (0..1), arbitrary (m..n) where m/n can range from 0 to “*”/infinity
- Uniqueness
  - for multi-valued cardinalities, whether the values must be unique or can be duplicates
- Ordering
  - for multi-valued cardinalities, whether the order is significant

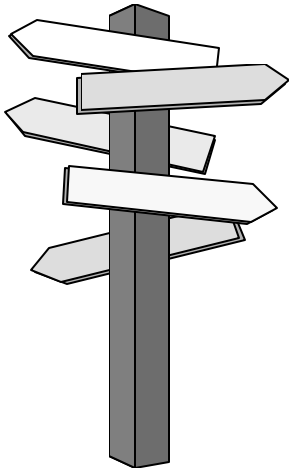
Common Multiplicities

- Single (1..1)
- Optional (0..1)
- Set (m..n) unique, unordered
- Bag (m..n) not unique, unordered
- List (m..n) not unique, ordered
- Unique List (m..n) unique, ordered
  - also called UList

Aggregation/Composition/ Containment

- An Association can be defined as being an aggregation/composition/containment
- This means one AssociationEnd contains the other AssociationEnd
- Impact on semantics of lifecycle operations
  - must do deep-structure “copy” and “delete”

The Agenda ... continued



- *The MOF Model: Advanced*
- **Extending the example**
- Generated IDL for the extended example
- Reflective IDL
- Building a Complete Meta-information service
- Standardisation of the MOF
- Summary and questions

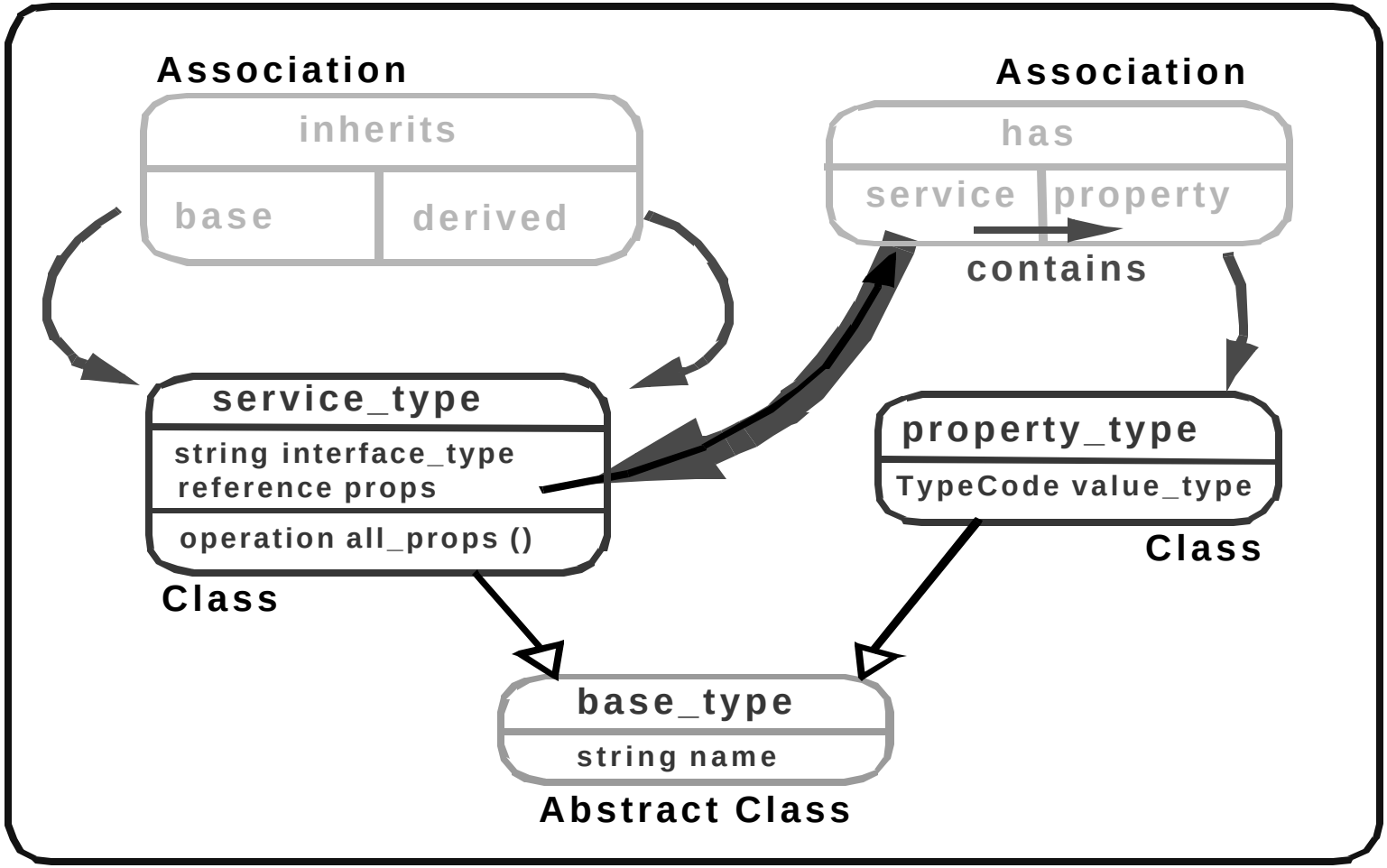
Extending the Example

- Create an abstract Class called base_type which contains a single attribute name
  - Modify service_type class and property_type class to inherit from base_type class
- Add a reference called props to service_type class to return all direct properties of the service type

Extending the Example

- Add an operation all_props to service_type to return all properties including inherited properties
- Change the property_type Association End to be *contained* by the service_type Association End in the Has Association

Extended Trader Type System



MODL for Extended Trader

```
package Trader {
    abstract class base_type {...};
    class service_type: base_type {...};
    class property_type: base_type {...};

    association has { ... };
    association inherits { ... };
};
```

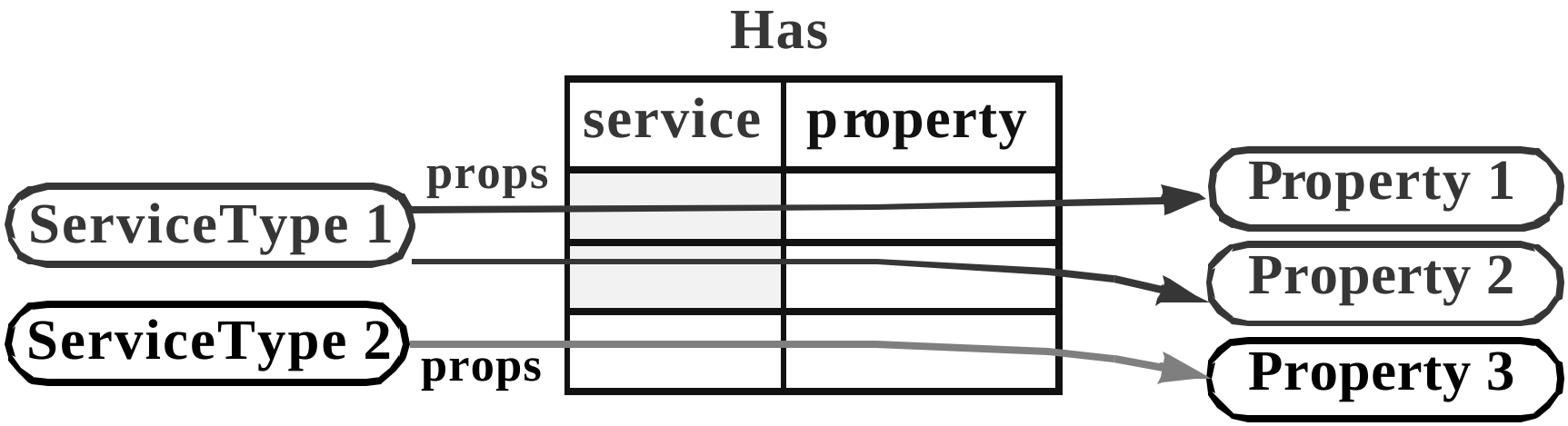
Extended MODL: Abstract Class and Inheritance

```
package Trader {
    abstract class base_type {
        attribute string name;
    };
    class property_type: base_type {
        attribute TypeCode value_type;
    };
    ...
};
```

Extended MODL: Reference

```
class service_type: base_type {
    attribute string interface_type;
    reference props to property of has;
    ...
};
```

Extended MODL: Reference




```
class service_type: ... {
   ...
   reference props to property of has;
};
```

Extended MODL: Operation

```
class service_type: base_type {
    ...
    exception duplicate_names {
        set [2..*] of property_type duplicates;
    };

    set [0..*] of property_type all_props ()
        raises (duplicate_names);
};
```

Extended MODL: Associations

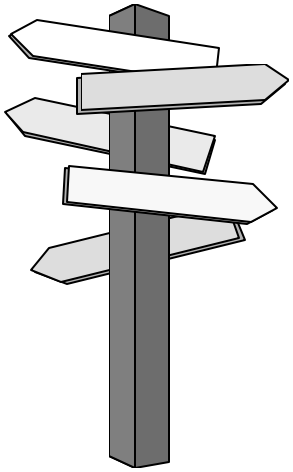
```
association has {
    end single service_type service;
    composite end
        set [0..*] of property_type property;
};
```

- Service types contain property types

Extended MODL: Associations

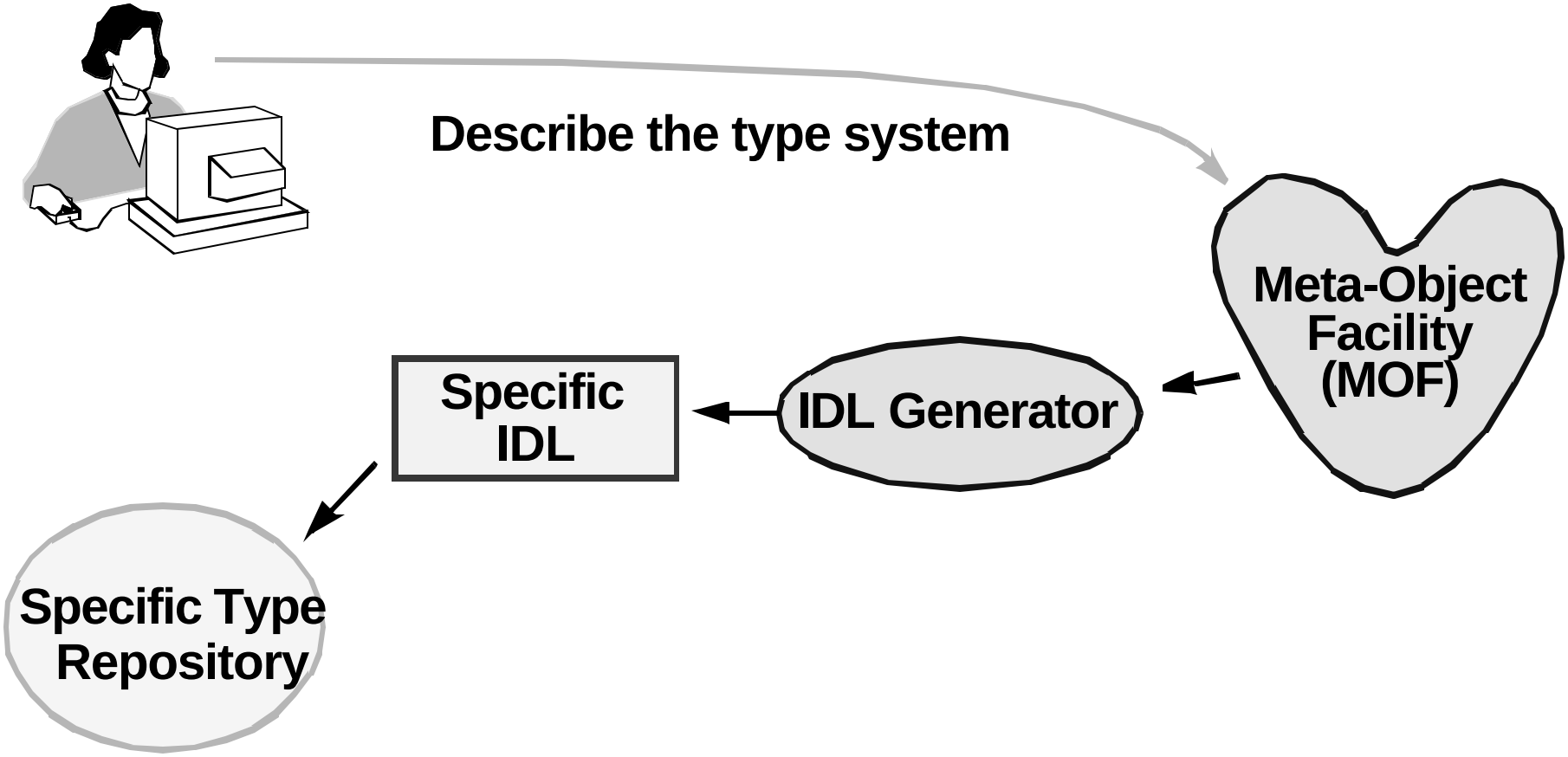
```
association inherits {
    end set [0..*] of service_type base;
    end set [0..*] of service_type QUOTE
"derived";
};
```

The Agenda ... continued



- *The MOF Model: Advanced*
- *Extending the example*
- **Generated IDL for the extended example**
- Reflective IDL
- Building a Complete Meta-information service
- Standardisation of the MOF
- Summary and questions

Generating IDL



Mapping for an abstract Class

- If a Class is declared as abstract
  - no change to its “instance” interface
  - its *Class* interface does not contain a create operation or an “all_of_class” attribute

Example: BaseType Class

```
interface BaseTypeClass: ... {
    readonly attribute BaseTypeSet
            all_of_type_base_type;
};

interface BaseType: BaseTypeClass, ... {
    string name () ...;
    void set_name (in string new_value) ...;
};
```

Impact of Inheritance

- Impact of inheritance on Class mapping
  - the derived Class's interfaces inherit from the base Class's interfaces
  - the "create" operation on the *Class* interface of a derived Class contains parameters to initialise the base Class's attributes

Example: ServiceType Class

```
interface ServiceTypeClass: BaseTypeClass, ... {
    ...
    ServiceType create_service_type(
        in string name,
        in string interface_type)
        raises (...);
};
```

Mapping for Reference

- References map onto operations to create, read, modify and delete values
- The signature of the generated operations depend on the multiplicity of the Association Ends
- Multiplicity on Attributes follow similar mapping rules...

Effect of Cardinality on Attributes and References

- Single has “set”, “get”
- Optional has “set”, “unset”, “get”
- Set and Bag also have “add”, “modify” and “remove” (to work with multiple values)
- List and UList also have “add_before”, “add_at”, “modify_at” and “remove_at” (to work with ordered values)

Impact of Composition within Associations

- No impact on generated IDL
- Impact on semantics of lifecycle operations
  - must do deep-structure “copy” and “delete”

Example: ServiceType Class

```
interface ServiceType: ..., BaseType {
    ...
    PropertyTypeSet props ()...;
    void unset_props ()...;
    void set_props (in PropertyTypeSet
new_value)...;
    ...
};
```

Example: ServiceType Class

```
interface ServiceType: ..., BaseType {
    ...
    void add_props (in PropertyType
new_element)...;
    void modify_props (
            in PropertyType old_element,
            in PropertyType new_element)...;
    void remove_props (
            in PropertyType old_element)...;
};
```

Generation rules for Operation and Exception

- An Operation maps onto an IDL operation on the instance interface for the Class
- An Operation can throw Exceptions which are declared within the *Class* interface for the Class

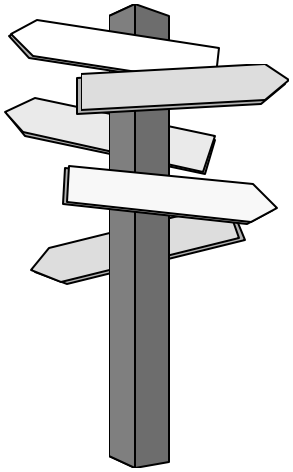
Example: Exception

```
interface ServiceTypeClass: ... {
    ...
    exception DuplicateNames {
        PropertyTypeSet duplicates;
    };
    ...
};
```

Example: Operation

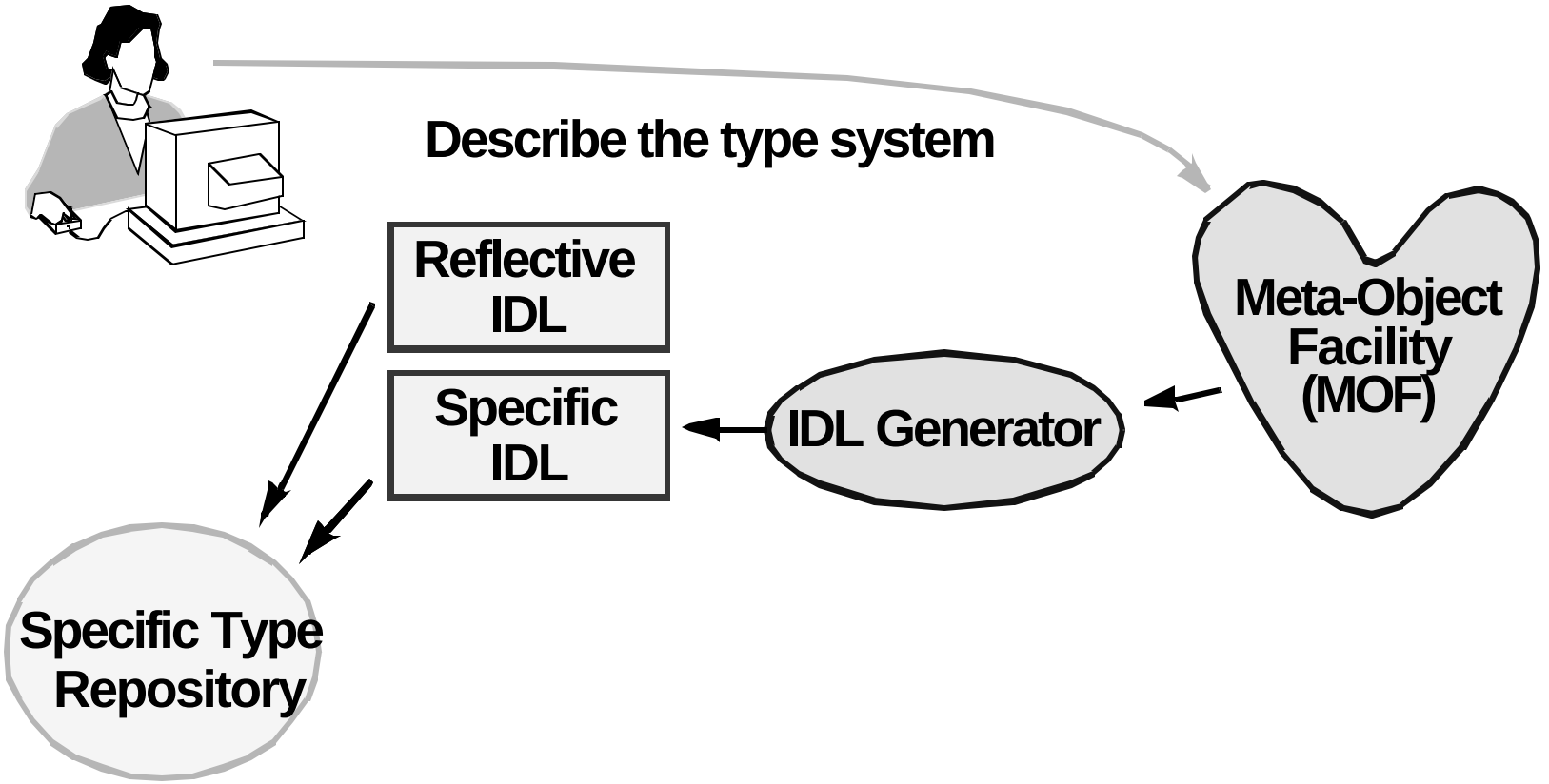
```
interface ServiceType: ... {
    ...
    PropertyTypeSet all_props ()
        raises (ServiceTypeClass::DuplicateNames,
    ...);
    ...
};
```

The Agenda ... continued



- *The MOF Model: Advanced*
- *Extending the example*
- *Generated IDL for the extended example*
- **Reflective IDL**
- Building a Complete Meta-information service
- Standardisation of the MOF
- Summary and questions

Reflective Interfaces



Reflective Interfaces

- IDL generation creates interfaces for a specific type system
- Reflective interfaces are hand-built and can be used with any type system
- Similar in use (and complexity) to CORBA's Dynamic Invocation Interface (DII)
  - use of "anys" throughout
  - useful for tool builders
- Specific interfaces inherit Reflective interfaces

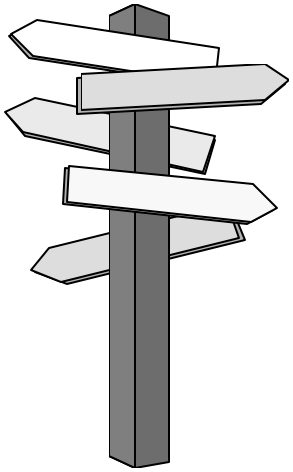
Reflective Get/Set Attribute

```
typedef RefObject DesignatorType;
typedef any ValueType;
interface RefObject: RefBaseObject {
   ValueType value (in DesignatorType feature)
        raises (...);
   void set_value (
        in DesignatorType feature,
        in ValueType value) raises (...);
};
```

Specific Get/Set Attribute

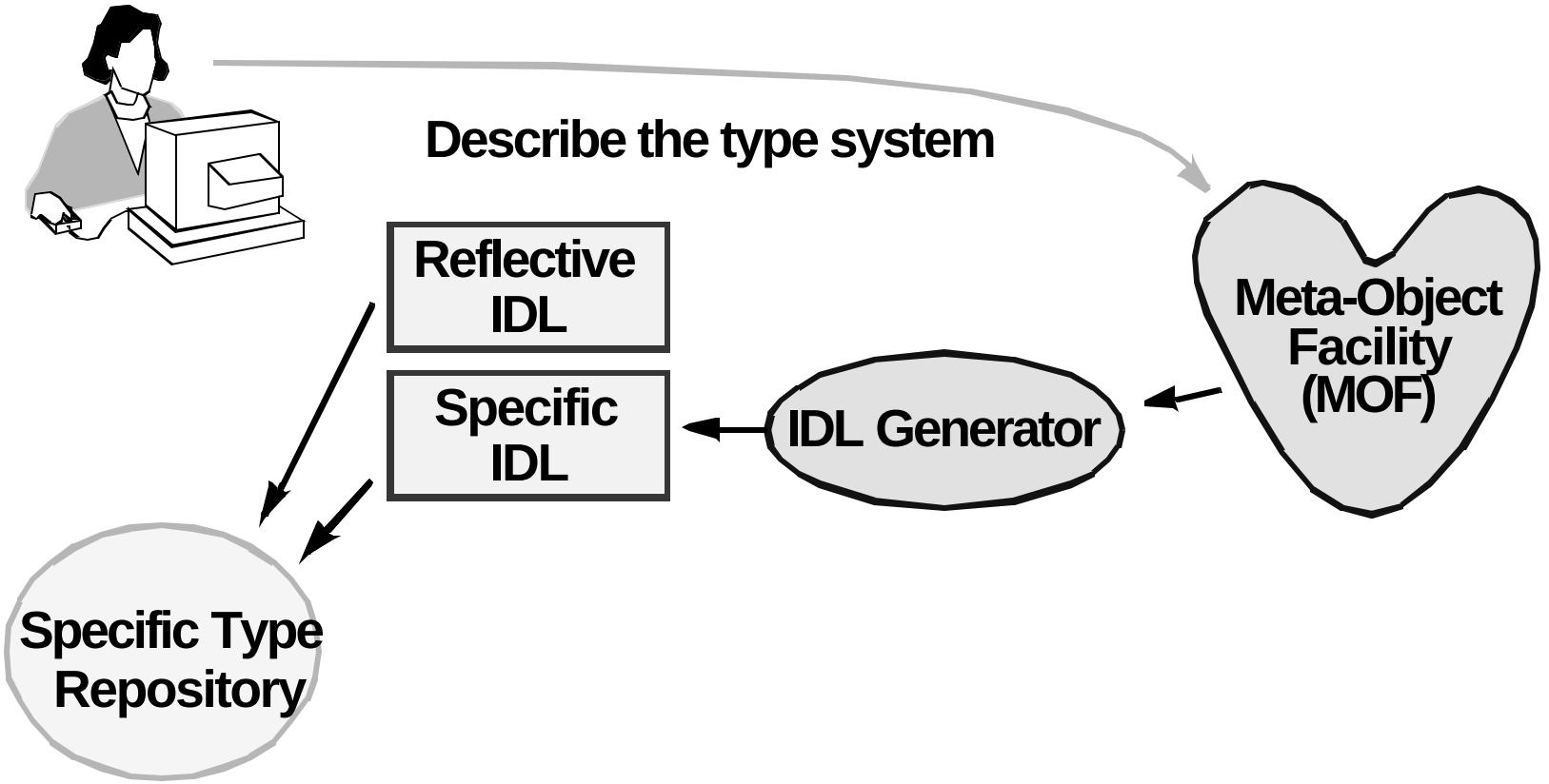
```
interface service_type: ... {
    string name () raises (...);
    void set_name (in string new_value)
        raises (...);
};
```

The Agenda ... continued



- *The MOF Model: Advanced*
- *Extending the example*
- *Generated IDL for the extended example*
- *Reflective IDL*
- **Building a Complete Meta-information service**
- Standardisation of the MOF
- Summary and questions

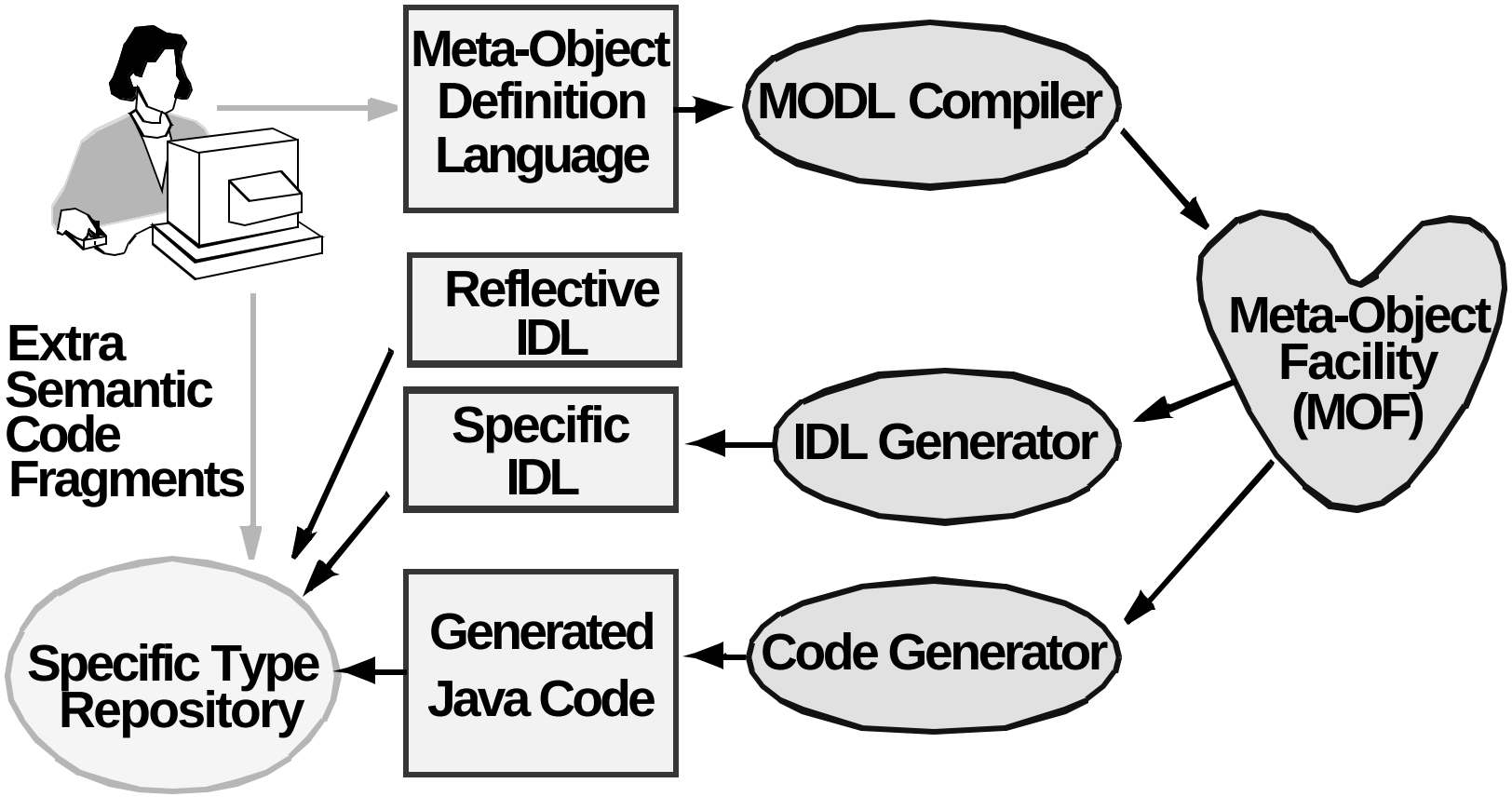
Building a Complete Meta-Information Service



MOF Based Tools

- How to populate a MOF?
  - MODL compiler (modl2mof): to input a type system description
- From IDL generation to server generation
  - mof2idl: automatically generates specific IDL
  - mof2moflet: generates server-side Java implementations
  - Allows the generation of type system repositories in about 5 minutes!

Automation Process



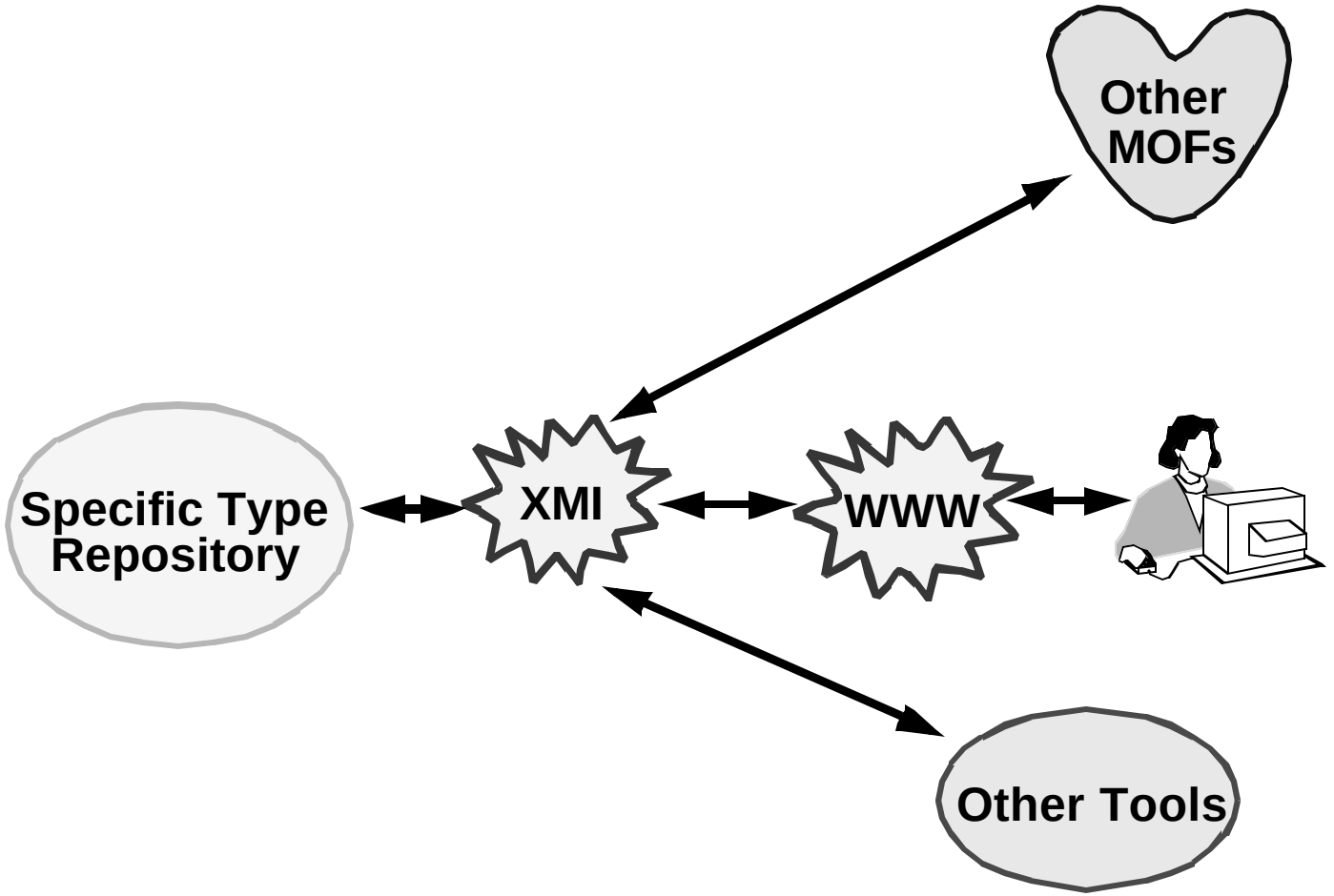
IDL for MOF Server

- MOF is a type system (for type systems)
  - so MOF can describe MOF
  - so feed the MOF description into the MOF
  - and poof! out comes the IDL for the MOF
- Like any specific type repository created by MOF
  - MOF itself has reflexive and specific interfaces
  - MOF's IDL has the same look-and-feel as any other type system
- Yes, the MOF really was bootstrapped this way!

XMI: MOF in XML

- How do we interchange types?
- XMI: **X**ML **M**etadata **I**nterchange
  - Uses the e**X**tensible **M**arkup **L**anguage from W3C
  - Allows the interchange of types between designers, repositories, etc
  - Leverages existing XML/HTML infrastructure
  - Subject of OMG RFP for Stream-Based Model Interchange Format (SMIF)

Type Interchange with XMI



Type Interchange

- XMI defines a DTD for each type system
- XMI generates XML for types in the MOF
- XMI reifies types into the MOF from XML

XMI for a Trader Service Type

```
<!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>
   ...
</XMI.content>
```

XMI.Content: Service Type

```
<Trader:ServiceType name="Savings_Account“ ...>
        ...
</Trader:ServiceType>

<Trader:ServiceType name="Cheque_Acount" ...>
        ...
</Trader:ServiceType>

<Trader:Inherits>
      <Trader:ServiceType xmi.idref="Service_1"/>
      <Trader:ServiceType xmi.idref="Service_2"/>
</Trader:Inherits>
```

XMI for a Trader Service Type

```
<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>
```

DTD for Trader Service Types

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

<!ELEMENT Trader ((ServiceType | PropertyType |
                                  Inherits | 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>
```

DTD for Trader Service Types

```
<!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)>
```

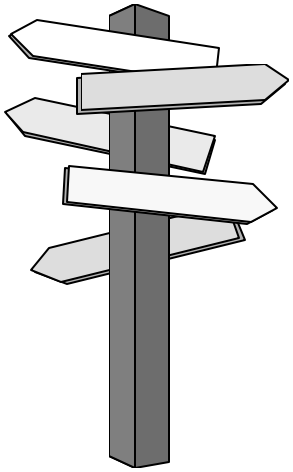
Type System Interchange

- Type systems are just types in the MOF Model
  - XMI defines a DTD for the MOF Model
  - XMI generates XML for a type system in the MOF
  - XMI reifies a type system into the MOF from XML

DSTC MOF Prototypes

- MOF Server
- dMOF tools:
  - IDL generator
  - Java implementation generator
- XMI DTD generator
- XMI XML generator
- XMI XML Reify-er

The Agenda ... continued



- *The MOF Model: Advanced*
- *Extending the example*
- *Generated IDL for the extended example*
- *Reflective IDL*
- *Building a Complete Meta-information service*
- **Standardisation of the MOF**
- Summary and questions

OMG Standardisation

- MOF submitted September 1997 by
  - DSTC, UNISYS, et el.
- MOF was adopted November 1997
- MOF 1.3 is the latest (June 1999) result of RTFs

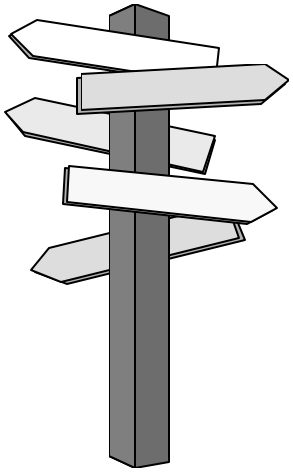
OMG Standardisation

- OMG specifications using the MOF:
  - UML
  - Event Notification
  - NORTEL WorkFlow submission
  - Component Model submission
  - EDOC (Enterprise Distributed Object Computing)
  - CWMI (Common Warehouse Metadata Interchange)

ISO/ITU-T Standardisation

- RM-ODP
  - MOF to be ODP Type Repository function
  - Draft International Standard (DIS) 1999
  - International Standard (IS) 2000
- ITU-T X.960 Recommendation 2000

The Agenda ... continued



- *The MOF Model: Advanced*
- *Extending the example*
- *Generated IDL for the extended example*
- *Reflective IDL*
- *Building a Complete Meta-information service*
- *Standardisation of the MOF*
- **Summary and questions**

Summary

- The Meta-object Facility (MOF) is a meta-information management facility:
  - supports multiple arbitrary type systems
  - allows new type systems to be added or composed/extended from existing type systems
  - supports type relationships within and between type systems

Summary

- The MOF specification mandates IDL mapping rules:
  - Ensures interoperable and portable type repositories
  - Access to repositories via the generated specific interfaces or the reflective interfaces
  - IDL generation can be automated

Summary

- MOF is:
  - an OMG adopted technology
  - progressing through the ISO and ITU-T standardisation process
- An active area of R&D within DSTC:
  - research (e.g. workflow, XMI, database interoperability, CSCW, enterprise modelling)
  - applications (e.g. military messaging, data warehousing, resource management)

More information

- http://www.dstc.edu.au/MOF/
- The public MOF mailing list: mof@dstc.edu.au
- The “MOFia”: mofia@dstc.edu.au
  - Dr Stephen Crawley (crawley@dstc.edu.au)
  - Simon McBride (sjm@dstc.edu.au)
  - Dr Kerry Raymond (kerry@dstc.edu.au)