A presentation of JMI

Java Metadata Interface

LIP6, Laboratoire d'Informatique de Paris 6
8, rue du Capitaine Scott
75015 Paris, France
http://www-src.lip6.fr/homepages/Marie-Pierre.Gervais

Context of this work

- The present courseware has been elaborated in the context of ModelWare European IST FP6 project (http://www.modelware-ist.org/)
- The MODELWARE project (Modelling solution for software systems) brings together 19 partners from Europe and Israel. Its main objectives are to develop a solution to reduce the cost of software systems large-scale deployment by the means of Model Driven Development techniques.
- To achieve the goals of large-scale dissemination of MDD techniques, ModelWare is also promoting the idea of collaborative development of courseware in this domain.
- The MDD courseware provided here with the status of open source software is produced under the EPL 1.0 licence.

Outline

- Overview of the standard
- The JMI interfaces categories
- The reflective interfaces
  - Translation rules
  - An exemple : the Use Case Diagram
- The tailored interfaces
  - Translation rules
  - An exemple : the Use Case Diagram

Rationale

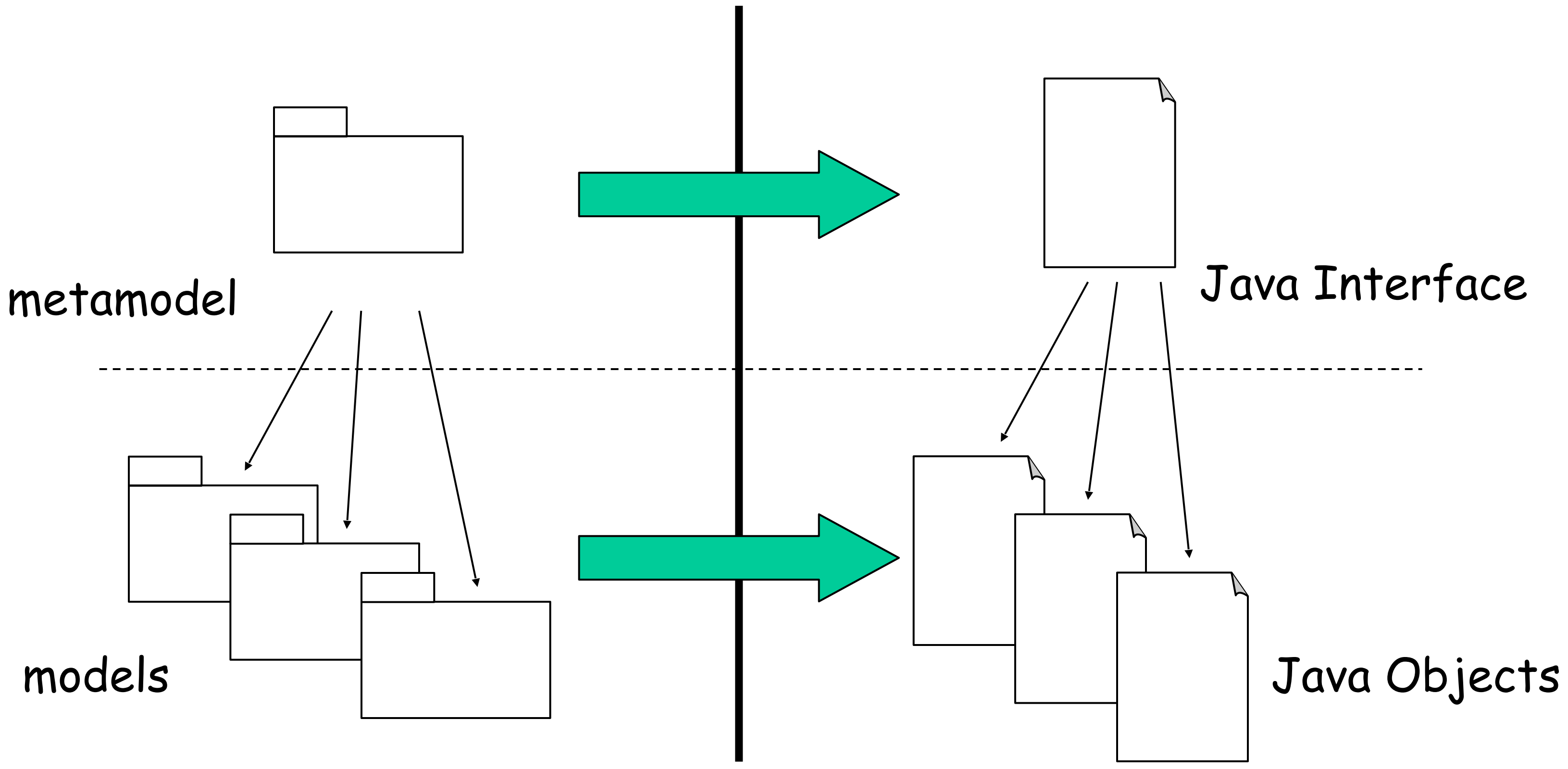
- A model is an abstract entity
- A model is structured by its metamodel
- A metamodel is an abstract entity
- A metamodel is structured by MOF
- MOF is an abstract entity!

- To handle (meta*)models, they need to be represented in an electronic format

Java Model Interface

- Defined by JCP (Java Community Process)
- Enable to represent models in the form of Java objects
- Defines rules enabling to build Java interfaces from a metamodel
  - JMI1.0 (final release, 28 June 2002) applies to MOF1.4

JMI: principle



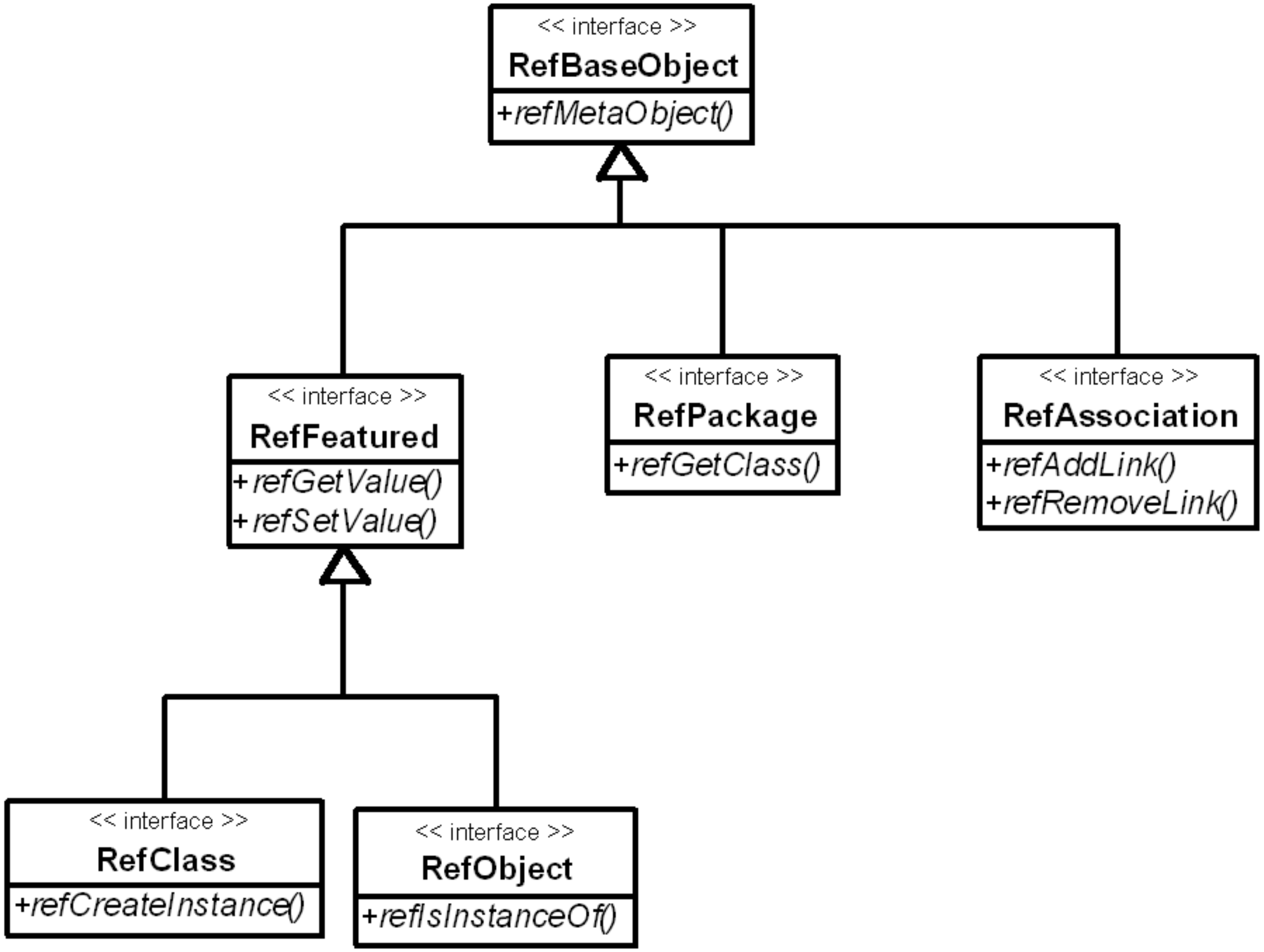
JMI

- Java API for handling models
  - Interfaces providing operations
- Develop a CASE tool for handling a model = develop a program using (invoking) the API

The JMI Interfaces

- Two categories
  - Reflective
    - Provides means to dynamically discover information on a model element (i.e., access to its meta-class)
    - Usable for all types of models
    - Independent from the metamodel
      - A model is a set of linked model elements, instances of meta-classes
  - Tailored
    - Dedicated to a type of models (models that are instances of the same metamodel)
    - Depending on the structure of this type of models (i.e., depending on the metamodel)

The Reflective Interfaces



RefBaseObject

- Represents any element (of a model or of a metamodel)
- Offers the `refMetaObject()` operation
  - Returns the metaclass of the element
  - Type of the metaclass : `refObject` (inherits from `RefBaseObject`)
  - Enables the navigation to the meta levels for discovering the structure of models.

RefFeatured

- Specialization of `RefBaseObject`
- Offers operations to access the element properties
  - Attribute, reference, operation
- `refGetValue()` and `refSetValue()` operations
  - Read and Write the property value
  - Signatures:
    - `void refSetValue(String propName, Object propValue)`
    - `Object refGetValue(String propName)`
  - Input parameter `propName`: a string identifying the property

RefClass

- Specialization of `RefFeatured`
- Represents the notion of element factory
  - Enables to build instances of a metaclass
- There is a `RefClass` per a metaclass
- Offers the `refCreateInstance()` operation
  - Creation of an instance of a metaclass

RefObject

- Specialization of `RefFeatured`
- Represents the notion of an element that is an instance of a metaclass
- Offers the `refIsInstance()` operation
  - Check whether this element is the instance of a given metaclass

RefAssociation

- Specialization of `RefBaseObject`
- Represents the notion of links between elements (i.e. `RefObjects`)
- Offers the `refAddLink()` and `refRemoveLink()` operations
  - Add and Remove links between elements

RefPackage

- Specialization of `RefBaseObject`
- Represents the notion of package (container of metaclasses).
- Offers the `refGetClass()` operation
  - List of the metaclasses (`RefClass`) contained in the package

Example : the Use Case Diagram (UCD)

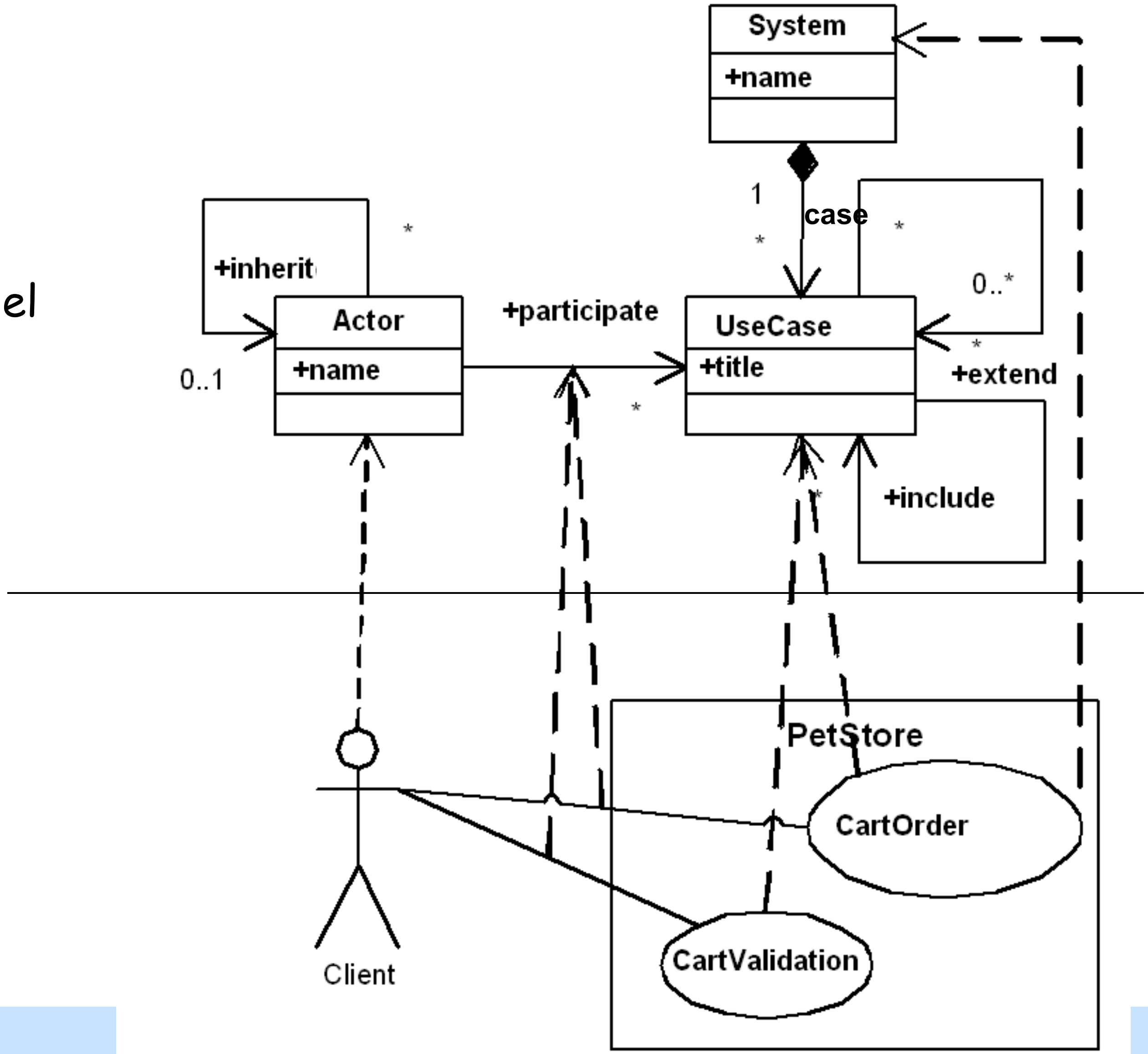
Construction of a model M

(instance of a metamodel)

The UCD metamodel and the model M

The UCD metamodel

A model M compliant to UCD



Construction of the model M

- With the reflective interfaces (without generating the metamodel-specific API)
- Writing the program that creates the model by using directly the operations of the reflective interfaces
- Need the implementation of the reflective interfaces (such as ModFact*, MDR*)
  - Bootstrapping application (i.e. obtaining the instance of `RefPackage`): implementation-specific mechanism

* See references

Construction of the model M

```
RefPackage p = //proprietary mechanism
RefObject act =
      p.refClass("Actor").refCreateInstance(null);
act.refSetValue("name","Client");
RefObject ca1 = p.refClass("UseCase").refCreateInstance(null);
ca1.refSetValue("title","CartOrder");
RefObject ca2 = p.refClass(" UseCase").refCreateInstance(null);
ca2.refSetValue("title","CartValidation");
Collection col = (Collection) act.refGetValue("participate");
col.add(ca1);
col.add(ca2);
RefObject sys = p.refClass("System").refCreateInstance(null);
sys.refSetValue("name","PetStore");
Collection cas = (Collection) sys.refGetValue("case");
cas.add(ca1);
cas.add(ca2);
```

The Tailored Interfaces

- Offer dedicated operations for handling models that are instances of a particular metamodel
- Example : operations for handling UCD models (instances of the UCD metamodel)

Navigation

- The tailored interfaces generated for the UCD metamodel enable, for each UCD model:
  - To get the number of actors, use cases, systems
  - To get the name of an actor (of a use case, of a system) and to modify it
  - To get the inheritance links between actors and to modify them
  - To get the use cases in which an actor participates and to modify them
  - To add/remove a use case
  - …

Generation Rules

- JMI1.0 defines the taylored interfaces generation from MOF1.4 metamodels
- Presentation of
  - Metaclass Rule
  - Meta-association Rule
  - Metapackage Rule

The metaclass Rule

- For a metaclass of a metamodel :
  TWO interfaces
- Instance Interface
  - Offers the operations to read/modify the instances of the metaclass
- Factory Interface
  - Offers the operations to create the instances of the metaclass

The Instance Interface

- Its name = name of the metaclass
- Offers the get/set operations for each meta-attribute of the metaclass
  - Ex. Setting the name of an Actor.
- Offers the operations of navigation for each metareference of the metaclass
  - Ex. Navigating from a UseCase to an Actor.
- Specialization of the reflective interface `RefObject`

The Factory Interface

- Its name : `name_metaclassClass`
- Offers the operations to create instances of the metaclass
  - Ex. Creating instances of Actor.
- Specialization of the reflective interface `RefClass`

The Meta-association Rule

- For a meta-association of a metamodel :
  ONE interface (Meta-association interface)
- Its name = name of the meta-association
- Offers the operations to create links between the instances of metaclasses
  - Ex. Linking the instances I1 and I2 with the meta-association A1.
- Offers the operations to navigate through the links
  - Ex. Obtaining I2 from I1.
- Specialization of the reflective interface `RefAssociation`

The Metapackage Rule

- For a package of a metamodel : ONE interface (Package interface)
- Its name = package name + suffix "`Package`"
  - Ex. "`Ucd`" + "`Package`" → `UcdPackage`
- Offers the operations enabling to get all the Factory interfaces of the metaclasses in this metapackage
  - Ex. Getting Factories for the metaclasses `Usecase, System, Actor.`
- Offers the operations enabling to get all the Meta-association interfaces of this metapackage
  - Ex. Getting Assocation interfaces for the meta-associations `"include", "extend".`
- Specialization of the reflective interface `RefPackage`

Example: the Use Case Diagram (UCD)

Generation of the tailored interfaces

Construction of the model M

List (1): Results of Metaclass Rule

- `Actor.java`
  - Generated from the metaclass `Actor` (Instance interface).
  - provides the operations: `getName()`, `setName()`, `getParticipate()`
- `ActorClass.java`
  - Generated from the metaclass `Actor` (Factory interface).
  - provides the operation: `createActor()`
- `System.java`
  - Generated from the metaclass `System` (Instance interface).
  - provides the operations: `getName()`, `setName()`, `getUseCase()`
- `SystemClass.java`
  - Generated from the metaclass `System` (Factory interface).
  - provides the operation: `createSystem()`
- `UseCase.java`
  - Generated from the metaclass `UseCase` (Instance interface).
  - provides the operations: `getTitle()`, `setTitle()`, `getInclude()`, `getExtend()`
- `UseCaseClass.java`
  - Generated from the metaclass `UseCase` (Factory interface).
  - provides the operation: `createUseCase()`

List (2) : Results of Meta-association Rule

- `AUseCaseSystem.java`
  - Generated from the meta-association between the metaclasses `UseCase` and `System`.
- `AInheritActor.java`
  - Generated from the meta-association on the metaclass `Actor` (an Actor inherits another Actor).
- `AParticipateActor.java`
  - Generated from the meta-association between the metaclasses `Use Case` et `Actor` (an Actor participates in a UseCase).
- `AIncludeUseCase.java`
  - Generated from the meta-association on the metaclass `Use Case` (A UseCase includes another UseCase).
- `AExtendUseCase.java`
  - Generated from the meta-association on the metaclass `Use Case` (A UseCase extends another UseCase).

List (3) : Result of Metapackage Rule

- `UcdPackage.java`
  - Generated from the metapackage containing all these metaclasses and meta-associations.
  - Has the following operations:
    - Getting all Factory interfaces:
      - `SystemClass getSystem()`
      - `UseCaseClass getUseCase()`
      - `ActorClass getActor()`
    - Getting all Meta-association interfaces:
      - `AUseCaseSystem getAUseCaseSystem();`
      - `AInheriteActor getAInheriteActor();`
      - …..

Construction of the model M

```
[1]UcdPackage extent = //proprietary mechanism
[2]System sys = extent.getSystem().createSystem("PetStore");
[3]Actor ac = extent.getActor().createActor("Client");
[4]UseCase ca = extent.getUseCase ().createUseCase ("CartOrder");
[5]UseCase ca2 =
   extent.getUseCase() .createUseCase("CartValidation");
[6]ac.getParticipate().add(ca);
[7]ac.getParticipate().add(ca2);
[8]sys.getUseCase().add(ca);
[9]sys.getUseCase().add(ca2);
```

References

- White Papers
  - http://java.sun.com/products/jmi/
- Download JMI
  - http://java.sun.com/products/jmi/download.html
  - http://packages.debian.org/unstable/libs/libgnujmi-java
- Repository
  - http://mdr.netbeans.org
  - http://modfact.lip6.fr