
Moses Repository

Martin Naedele

Computer Engineering Group

Swiss Federal Institute of Technology Zurich

Overview

- ◆ MOSES repository framework architecture
- ◆ How to
 - create new entity types
 - change entity type behavior

Introduction

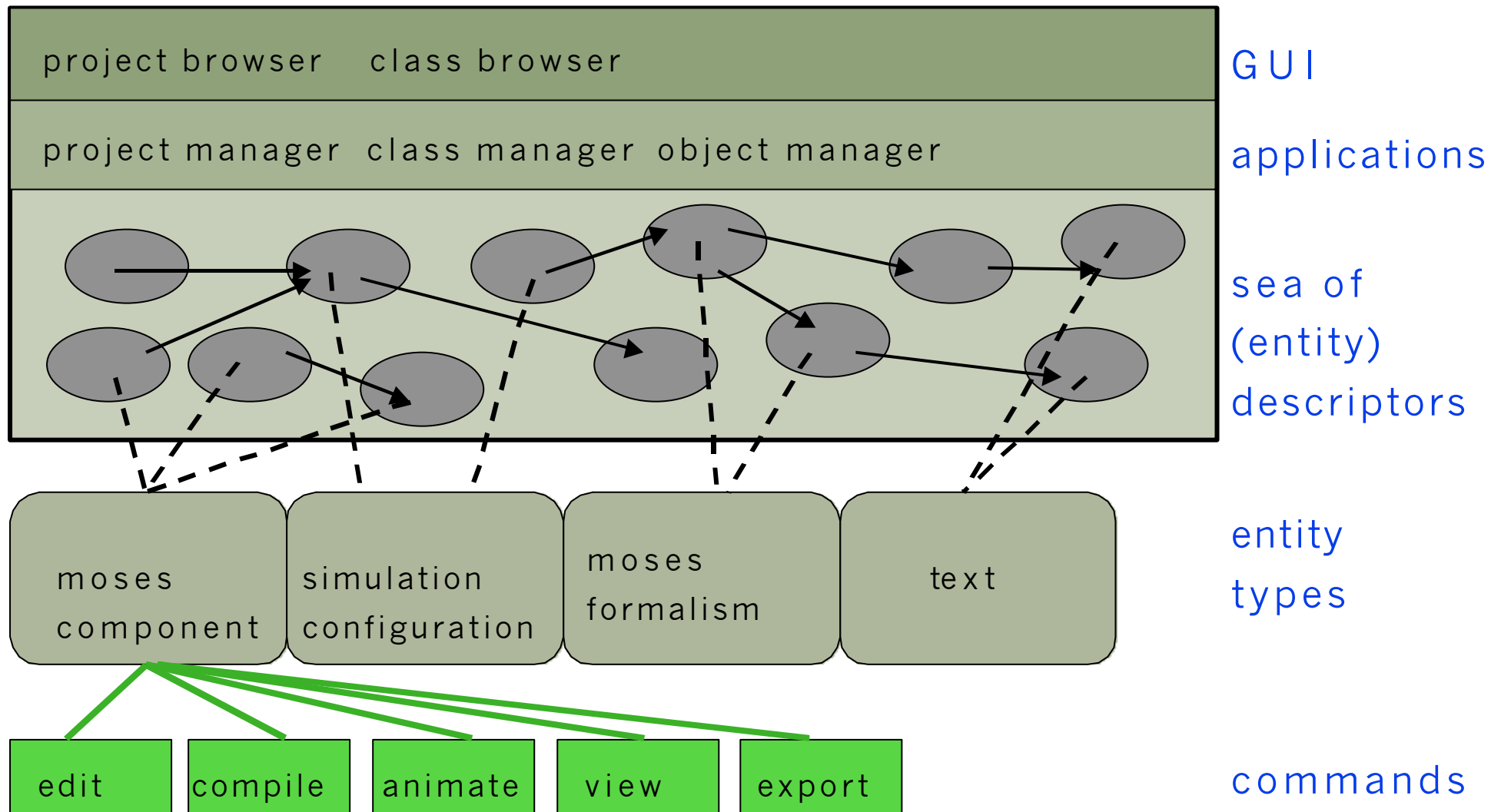
- ◆ Management of **different types** of design documents and entities
 - model components
 - formalisms
 - simulation experiment configurations
 - text documentation
 - program code
 - ...

in **one repository** while allowing **multiple views**

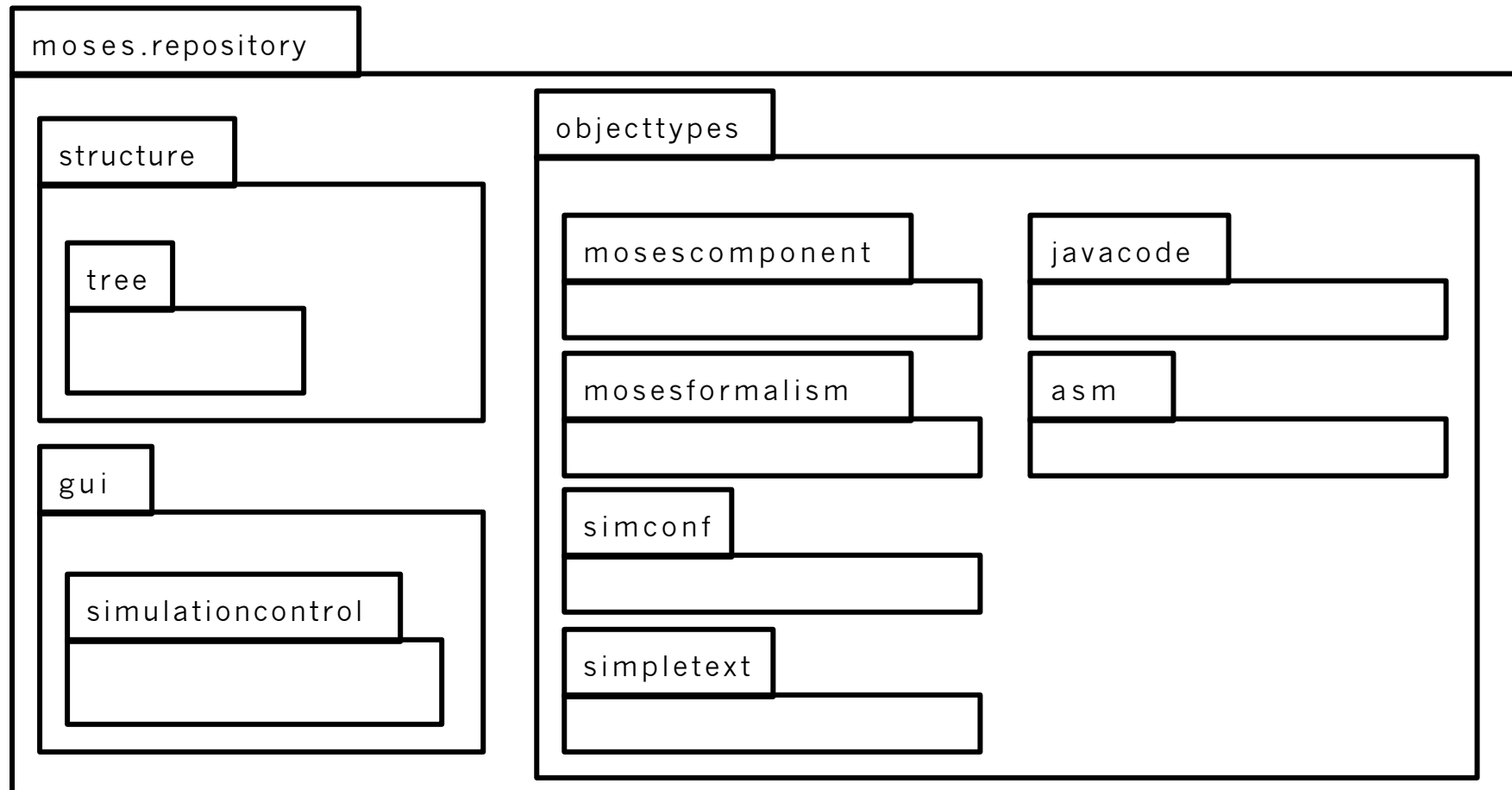
- project hierarchy
- class hierarchy
- version history
- ...

on them.

Introduction



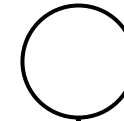
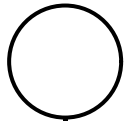
Repository Package Structure



Descriptor

Serializable

Cloneable



moses.repository.structure.Descriptor

storage: Dictionary

Descriptor()

Descriptor(Hashtable h)

get(Object key): Object throws PropertyVetoException

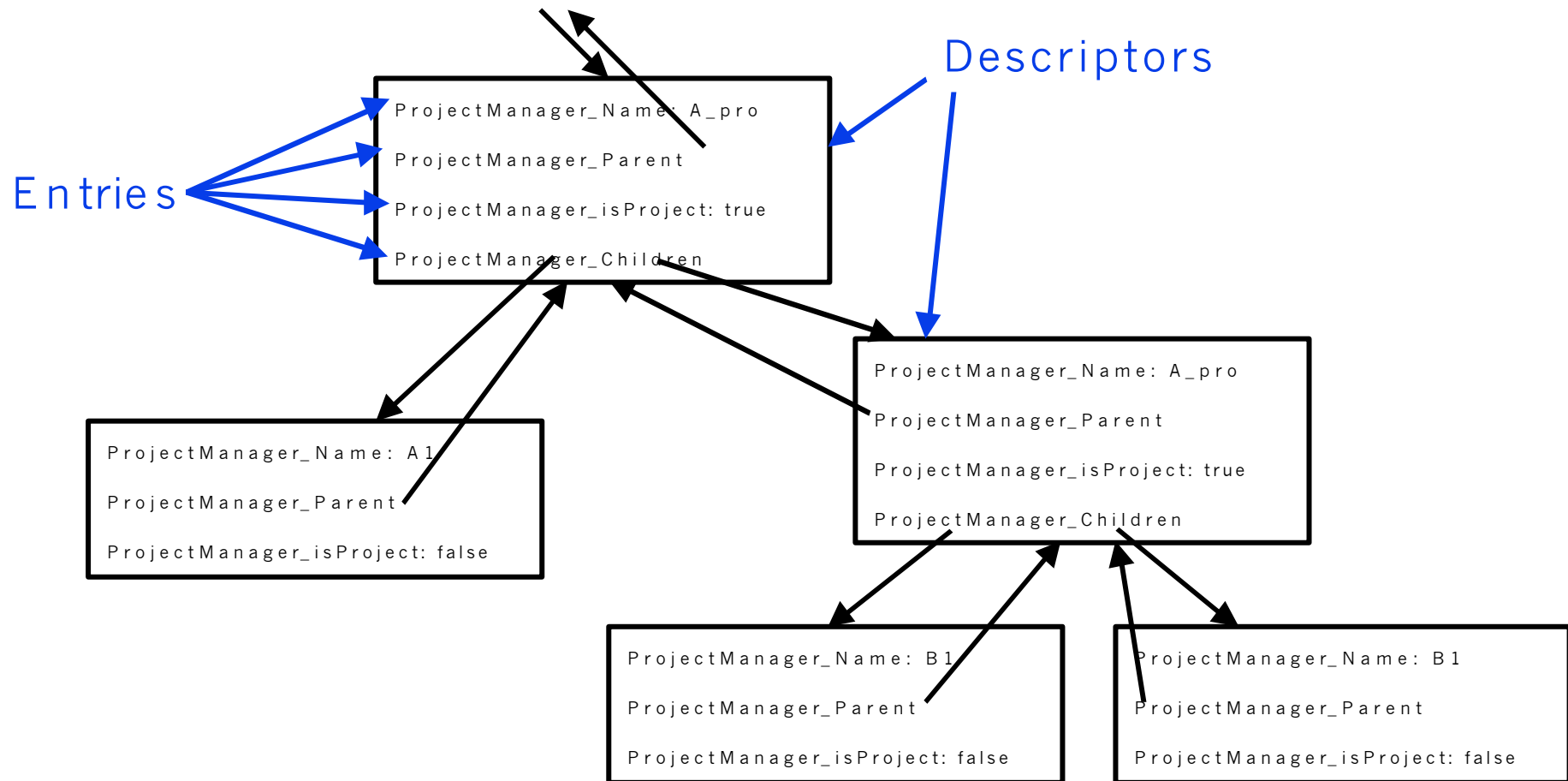
set(Object key, Object value) throws PropertyVetoException

...

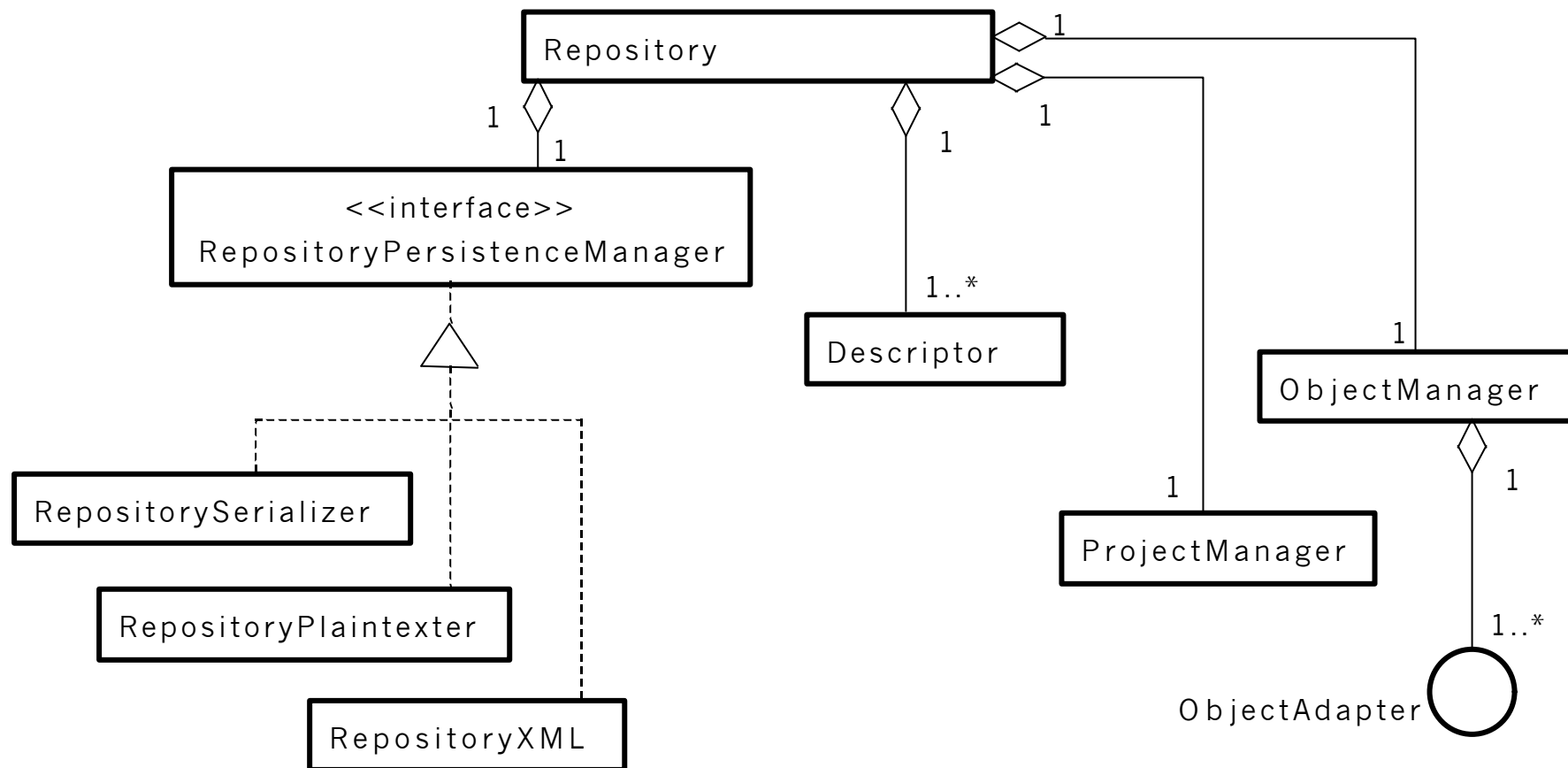
Descriptor

- ◆ Repository is composed of `descriptors` (`moses.repository.structure.Descriptor`)
- ◆ Descriptors *may* refer to a `payload` (e.g. design entities), but need not (e.g. project nodes)
- ◆ A descriptor *encapsulates* a `Dictionary`/`Hashtable` for properties
- ◆ The `properties` carried by the descriptor are *type/role specific*
- ◆ Reference properties form *graphs of descriptors* according to certain *views* on the system
- ◆ Get/set may throw `PropertyVetoException`
- ◆ Objects can register themselves with any descriptor as `PropertyChangeListener` or `VetoableChangeListener`, e.g. to manage dependencies or implement access control
- ◆ No own *persistence* management!

Repository: Project View



Repository Class Structure



Class Repository

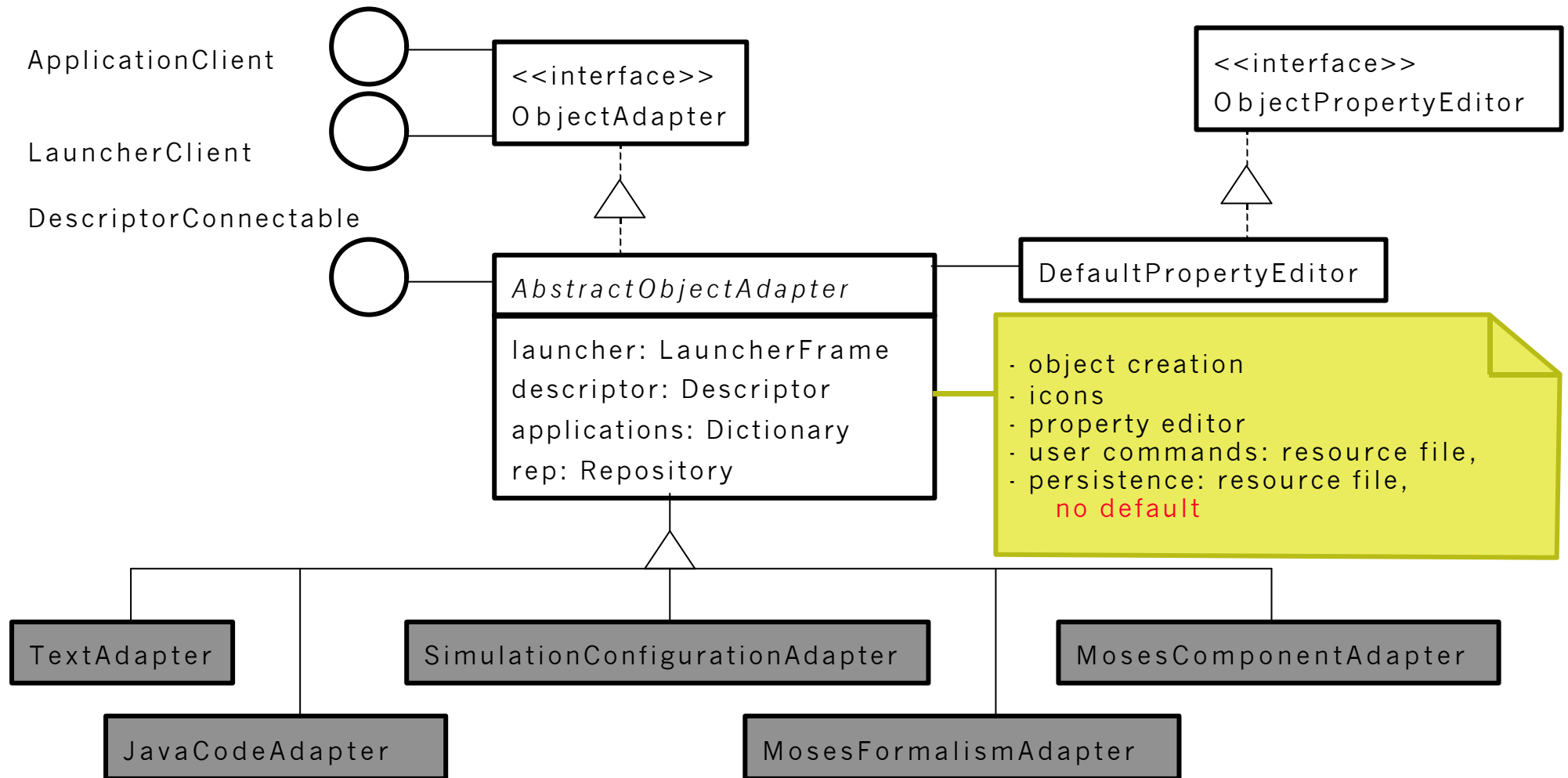
- ◆ Descriptor life-cycle management
 - add/remove
 - persistence
- ◆ Management of listeners
 - PropertyChangeListeners
 - VetoableChangeListeners
- ◆ Configured using Repository.properties

```
# strategy class used to store/load descriptor content
persistencemanager=moses.repository.structure.RepositoryPlaintexter
# name of the actual file in which the descriptors are stored
storagename=demo.rep
```
- ◆ Starting of repository applications (e.g. project manager)
- ◆ Regeneration of entity descriptor to ObjectAdapter instance relations on startup

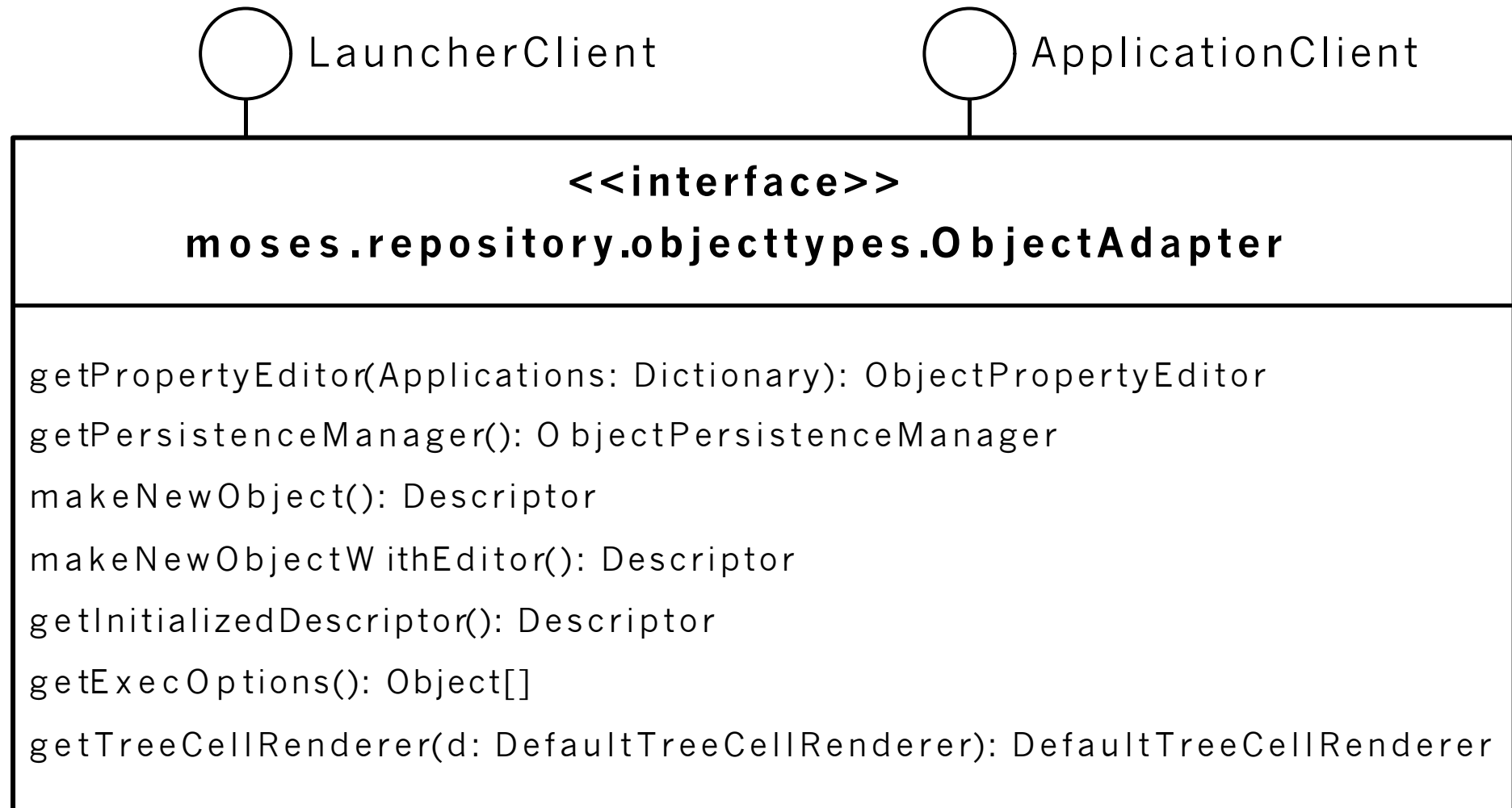
Entity Types

- ◆ The repository can manage **different types of objects** (design entities), e.g.
 - model components
 - formalisms
 - simulation experiment configurations
- ◆ For each object type **specific behavior** must be provided for
 - persistence of payload
 - user commands
 - iconic representation in browsers
 - object creation and property initialization
 - property editing
- ➔ **ObjectAdapter**
 - referenced in descriptor
 - configured using its own resource file

ObjectAdapter Class Structure



Interface ObjectAdapter



How to ...

- ◆ ...create a new `ObjectAdapter`

- write a class that extends `AbstractObjectAdapter`
- implement a suitable constructor
- provide a resource file named `<newOAName>.properties` which at least declares a `PersistenceManager`
- overwrite the implementation of the methods
 - `getPropertyEditor(Applications: Dictionary): ObjectPropertyEditor`
 - `getInitializedDescriptor(): Descriptor`
 - `getTreeCellRenderer(d: DefaultTreeCellRenderer): DefaultTreeCellRenderer`as needed to
 - use a custom property editor
 - change the initial values of descriptor properties or define new default properties
 - change the iconic representation of objects of this type in the repository browser
- register with the framework: project manager menu options -> object adapters

Format of the resource file

Icons=*LockedIcon:UnlockedIcon:additionalicon*

LockedIcon=*moses/repository/images/generic2.gif*

UnlockedIcon=*moses/repository/images/c.gif*

additionalicon=*mymoses/icons/myIcon.gif*

Actions=*act1:act2:act3*

act1=*mymoses.actions.Act1*

act2=*mymoses.actions.Act2*

act2DisableOptions=*isNoEdit*

act3=*mymoses.actions.Act3*

act3DisableOptions=*isNoExec*

StorageLocationBase=*./storage/aaa*

ObjectPersistenceManager=*mymoses.mytype.MyTypePersistenceManager*

... (entity type specific entries)

How to ...

- ◆ ... create an actual object of a certain type
 - use the methods
 - » `makeNewObject()`
 - or
 - » `makeNewObjectWithEditor()`
- ◆ ... initialize object specific descriptor attributes
 - override method **`getInitializedDescriptor()`** of `AbstractObjectAdapter` and code the necessary assignments there
 - within the new method call **`getInitializedDescriptor()`** of the superclass

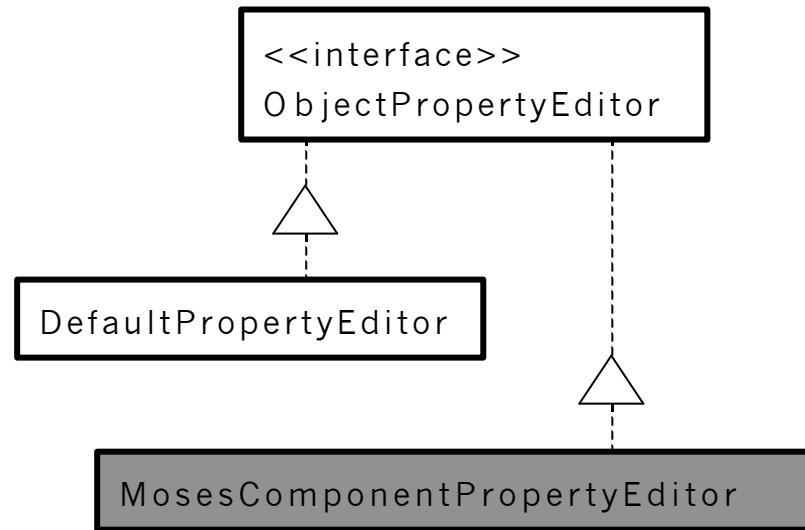
How to ...

- ◆ ... **display other icons** for the **locked/unlocked state** of an entity
 - give references to the desired GIF-files in the resource file
 - defaults are: **unknown.gif** and **unknown_locked.gif**
- ◆ ... **change the rendering policy** of an object in the tree browser
 - override the method **getTreeCellRenderer()** of `AbstractObjectAdapter` to implement the desired behavior
 - useful to visualize different states of an entity
 - default is a renderer that shows whether the entity is locked
- ◆ ... **use additional icons** for the rendering
 - add the corresponding information in the 'icons' section of the resource file to automatically load the icons
 - use the methods **getIcons()** of `AbstractObjectAdapter` to access the loaded icons, e.g. within a `TreeCellRenderer`

How to ...

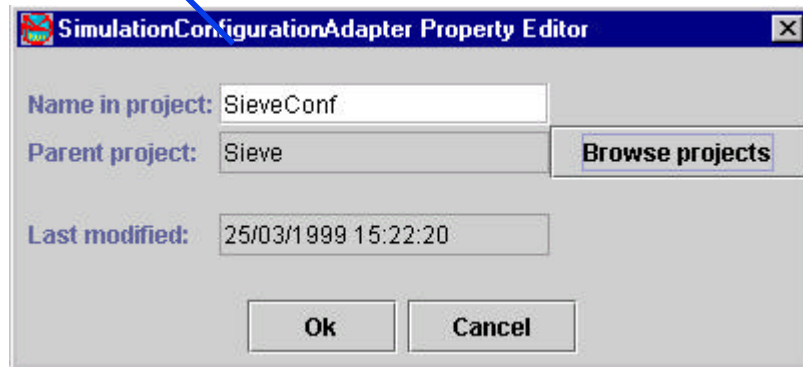
- ◆ ... construct an objecttype specific property editor
 - write a new class that implements `ObjectPropertyEditor`
 - override **`getPropertyEditor()`** of `AbstractObjectAdapter` to return an instance of this new editor class
 - default is the class `DefaultPropertyEditor` which allows to change the name of the entity in the project hierarchy and the parent project

ObjectPropertyEditor Class Structure

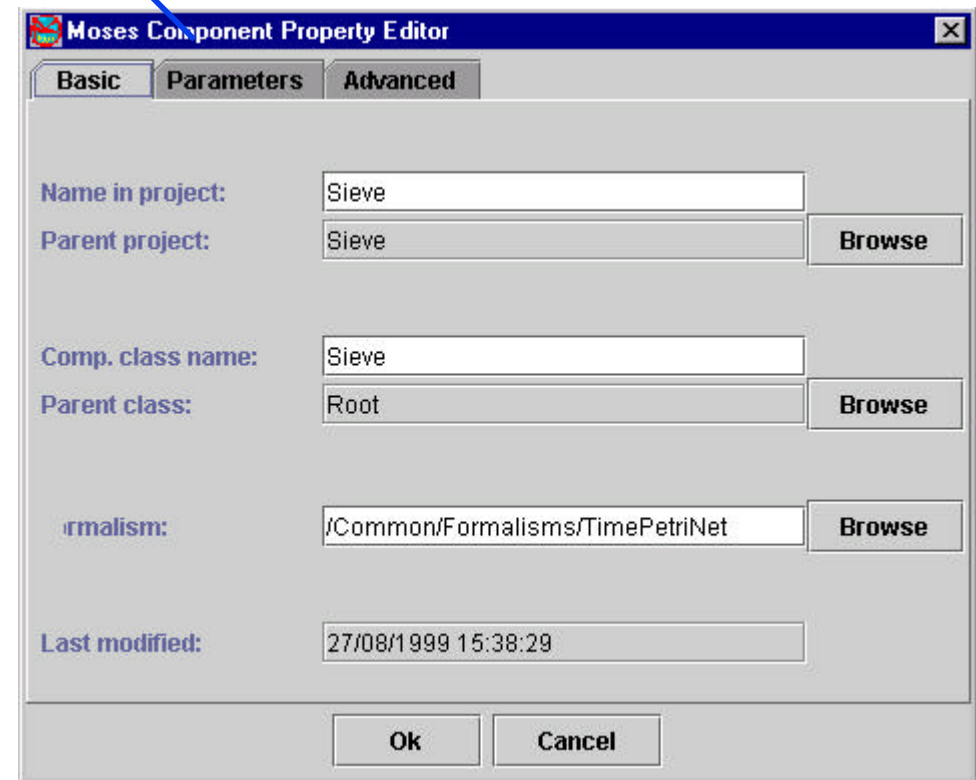


Object property editors

default: DefaultPropertyEditor



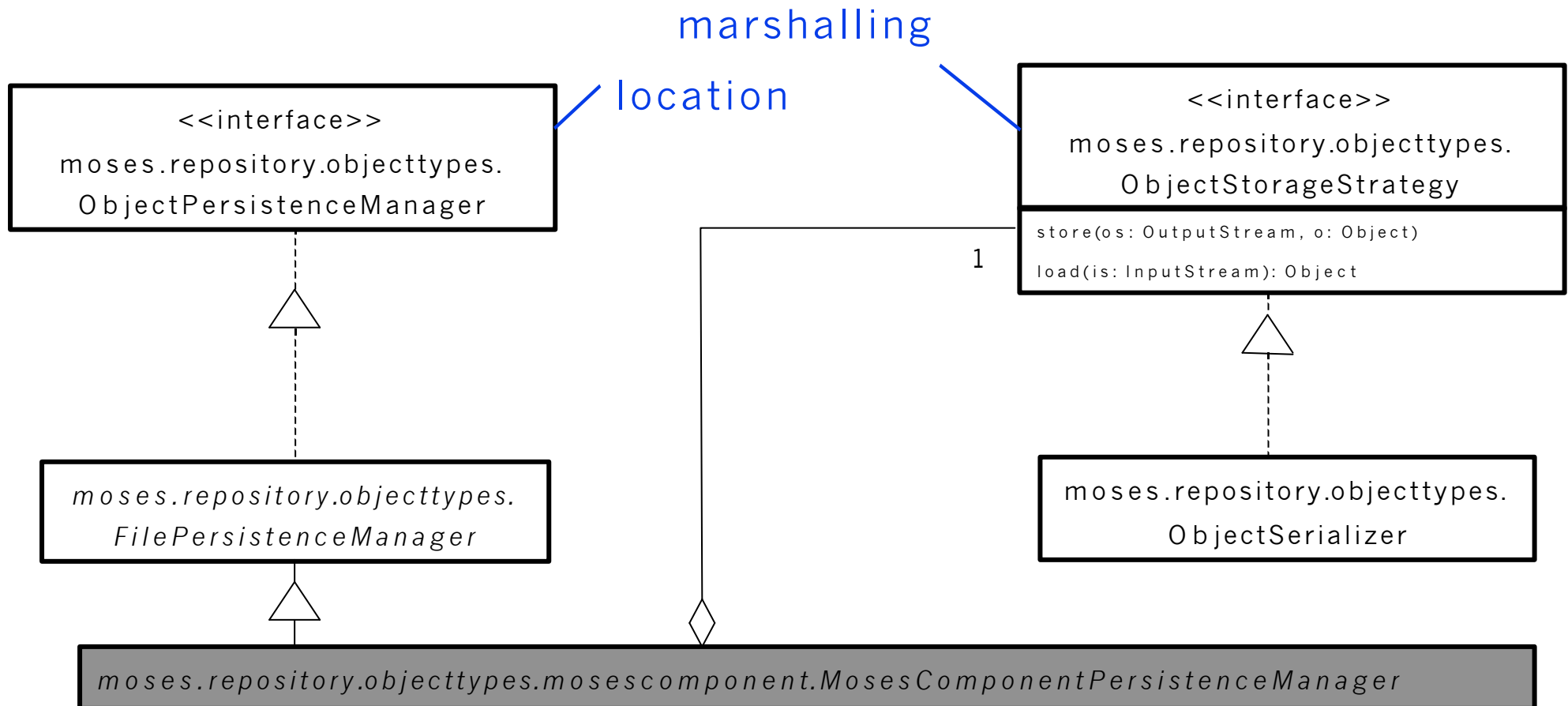
custom: MosesComponentPropertyEditor



How to ...

- ◆ ... make an entity persistent
 - persistent storage management of an entity is assumed to be a prerequisite or consequence of a user action and must be coded there
 - the framework provides some support through flexible configuration of strategies for
 - » managing the **storage location** (e.g. file system, database): ObjectPersistenceManager
 - » managing the **storage format** (e.g. ASCII text, serialized): ObjectStorageStrategy

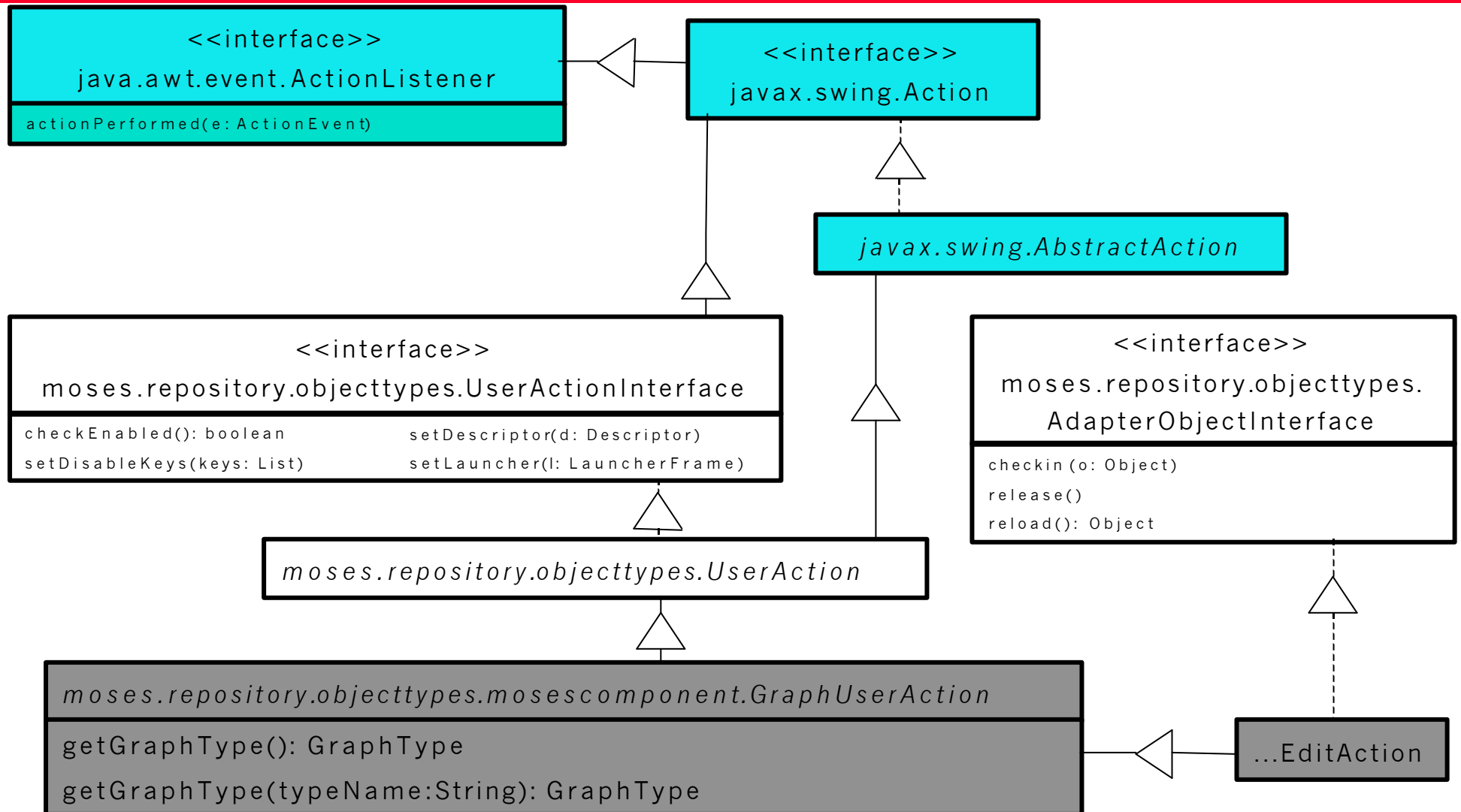
Object Storage Class Structure



How to ...

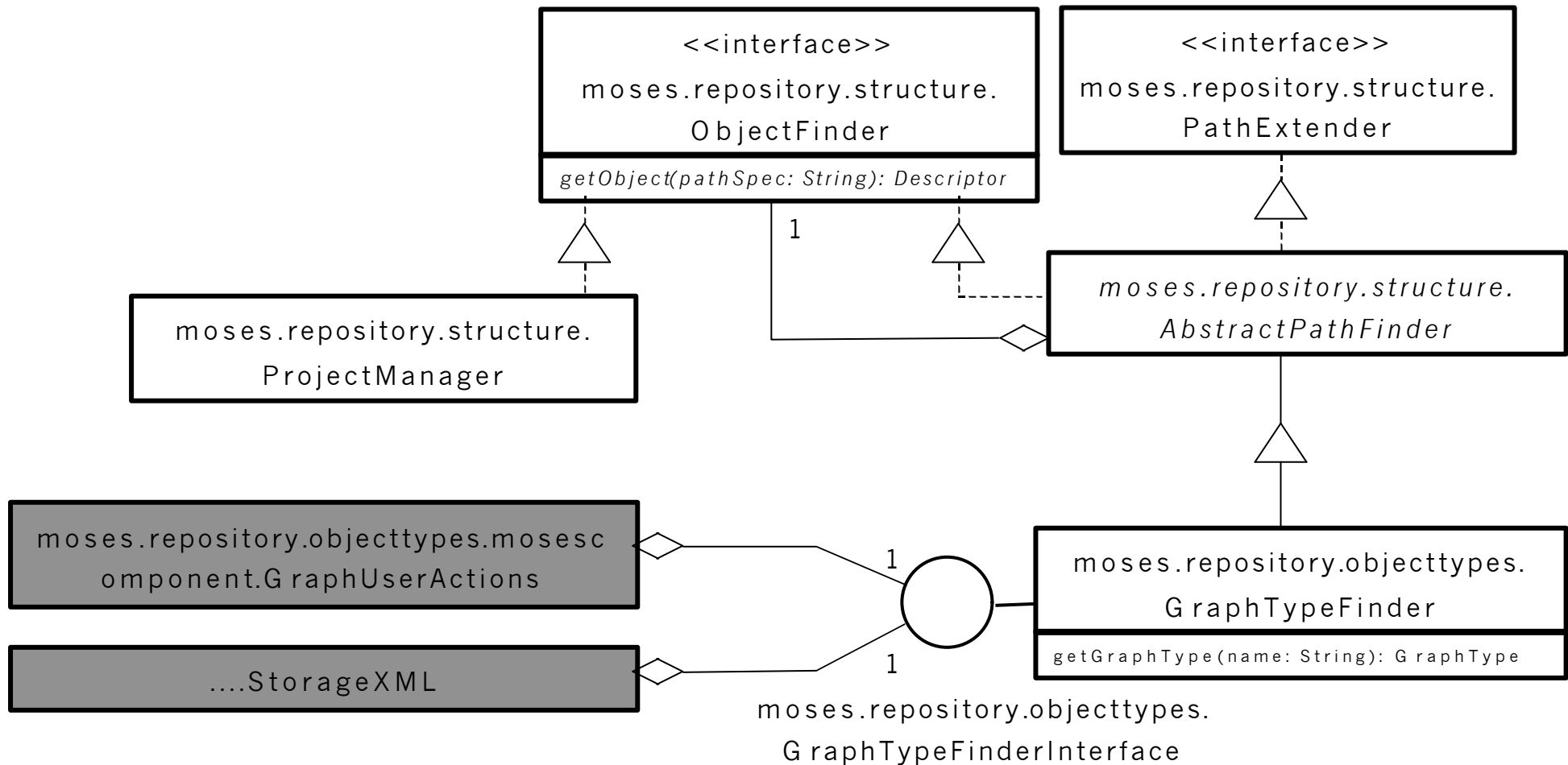
- ◆ ... define object type specific user actions
 - each entity type has two default user actions:
 - » Edit properties
 - » Remove
 - other actions must be individually implemented
 - write a class that implements `UserAction` (or `GraphUserAction` if the access to graph types from the repository is necessary)
 - the ‘business logic’ of the action should be implemented in method `actionPerformed()`
 - add a `reference` to the new class file in the ‘actions’ section of the object type `resource file`, or alternatively,
 - override the method `addActions()` of `AbstractObjectAdapter` to hard-code the action configuration
 - user actions are `responsible to lock/unlock` entities as needed

Action Class Structure



How to ...

- ◆ ... retrieve objects from the repository using program code



How to ...

- ◆ ... enable/disable user actions (1)
 - the method **checkEnabled()** of each UserAction is called each time a list of all available user actions for an entity and situation is constructed by the framework
 - if it returns **false**, the action is **disabled** and **grayed out** in the menu

How to ...

- ◆ ... enable/disable user actions (2)
 - add a new entry, e.g. `notCompiled` to the descriptors of all corresponding objects
 - add line `myactionDisableOptions=notCompiled` to the resource file to define disabling flags
 - set the value for this entry to `true` or `false` (Strings) according to the intended logic:
 - » `true`: action is disabled
 - » `false`: action is enabled
 - by default **`checkEnabled()`** calls **`checkDisableOptions()`** which evaluates the disabling flags

Summary: Repository Configuration

