EMF4CPP

Generating Ecore Models for C++

Matthias Dörfel, INCHRON GmbH
doerfel@inchron.com

MUC++
September 19, 2018

Overview

- EMF4CPP allows to use the Eclipse Modeling Framework in C++ projects
- Reuse metamodels based on ecore
- Generate a C++ class hierarchy
- Runtime support system with generic algorithms based on reflection
- Exchange of model instances serialized as XMI

Models, Metamodels and Meta-Metamodels

A model (or model instance) is formed by objects at runtime

```
auto d = new Department;
p->getEmployees().push_back(new Employee);
```

Models, Metamodels and Meta-Metamodels

The metamodel is the class definition

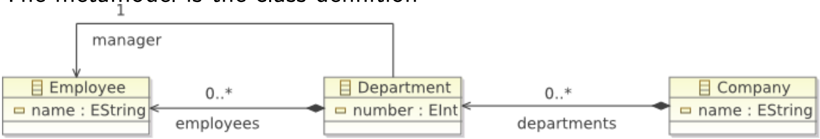
```cpp
class Employee;
class Department {
    Employee* m_manager = nullptr;
    std::vector<Employee*> m_employees;
public:
    Department() = default;

    Employee* getManager() { return m_manager; }
    void setManager(Employee* e) { m_manager = e; }

    std::vector<Employee*>& getEmployees() {
        return m_employees; }
};
```

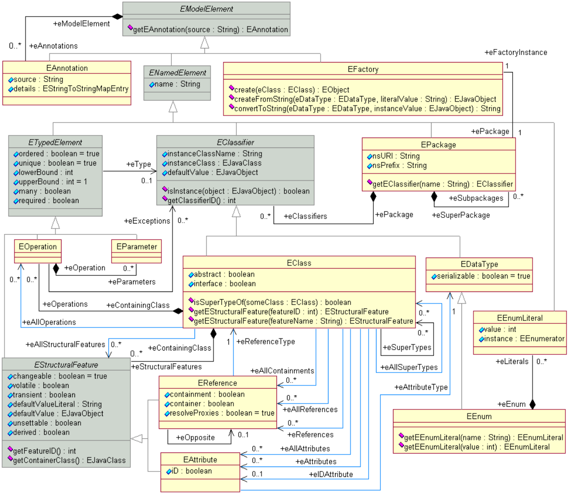
Models, Metamodels and Meta-Metamodels

The metamodel is the class definition



Models, Metamodels and Meta-Metamodels

The metamodel is an instance of the Ecore model



EModelElement
getEAnnotation(source : String) : EAnnotation

EAnnotation
source : String
details : EStringToStringMapEntry

ENamedElement
name : String

EFactory
create(eClass : EClass) : EObject
createFromString(eDataType : EDataType, literalValue : String) : EJavaObject
convertToString(eDataType : EDataType, instanceValue : EJavaObject) : String

ETypedElement
ordered : boolean = true
unique : boolean = true
lowerBound : int
upperBound : int = 1
many : boolean
required : boolean

EClassifier
instanceClassName : String
instanceClass : EJavaClass
defaultValue : EJavaObject
isInstance(object : EJavaObject) : boolean
getClassifierID() : int

EPackage
nsURI : String
nsPrefix : String
getEClassifier(name : String) : EClassifier

EOperation

EParameter

EClass
abstract : boolean
interface : boolean
isSuperTypeOf(someClass : EClass) : boolean
getEStructuralFeature(featureID : int) : EStructuralFeature
getEStructuralFeature(featureName : String) : EStructuralFeature

EDataType
serializable : boolean = true

EEnumLiteral
value : int
instance : EEnumerator

EStructuralFeature
changeable : boolean = true
volatile : boolean
transient : boolean
defaultValueLiteral : String
defaultValue : EJavaObject
unsettable : boolean
derived : boolean
getFeatureID() : int
getContainerClass() : EJavaClass

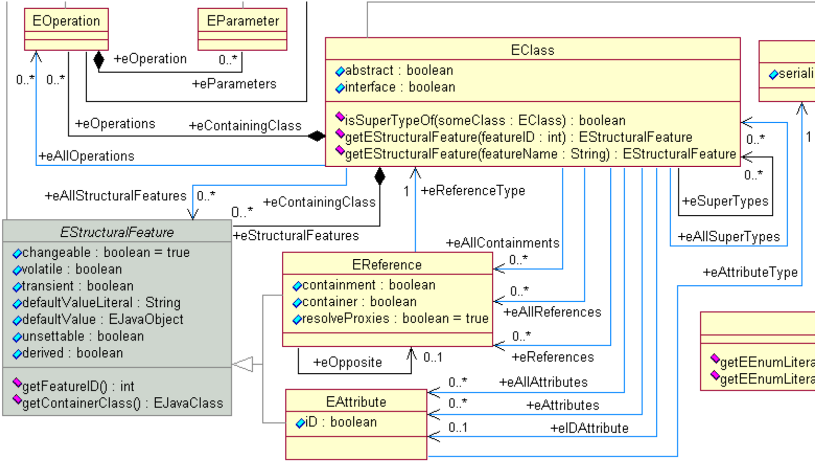
EReference
containment : boolean
container : boolean
resolveProxies : boolean = true

EAttribute
iD : boolean

EEnum
getEEnumLiteral(name : String) : EEnumLiteral
getEEnumLiteral(value : int) : EEnumLiteral

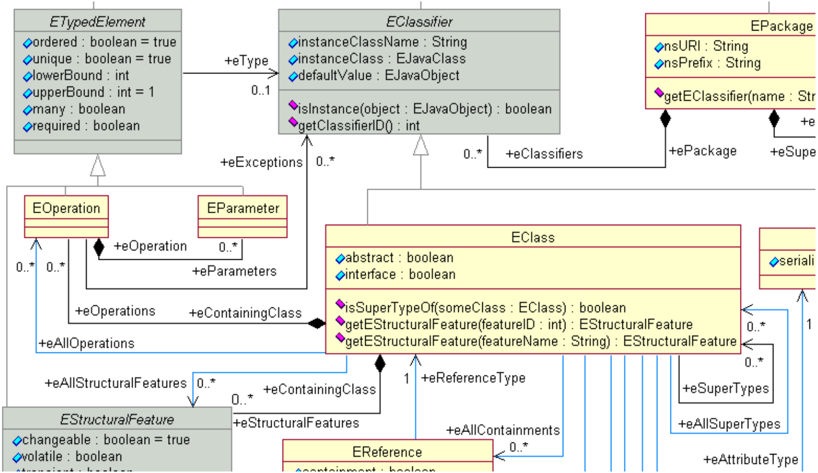
Models, Metamodels and Meta-Metamodels

At the core: EClass



Models, Metamodels and Meta-Metamodels

At the core: EClass



ETypedElement
ordered : boolean = true
unique : boolean = true
lowerBound : int
upperBound : int = 1
many : boolean
required : boolean

EClassifier
instanceClassName : String
instanceClass : EJavaClass
defaultValue : EJavaObject
isInstance(object : EJavaObject) : boolean
getClassifierID() : int

EPackage
nsURI : String
nsPrefix : String
getEClassifier(name : Str

EOperation

EParameter

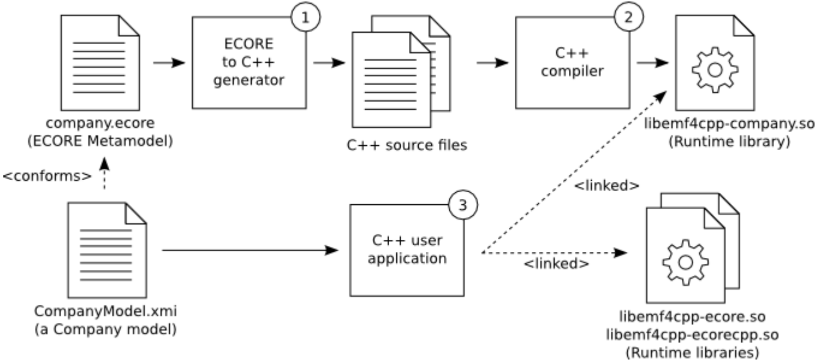
EClass
abstract : boolean
interface : boolean
isSuperTypeOf(someClass : EClass) : boolean
getEStructuralFeature(featureID : int) : EStructuralFeature
getEStructuralFeature(featureName : String) : EStructuralFeature

seriali

EStructuralFeature
changeable : boolean = true
volatile : boolean
transient : boolean

EReference
containment : boolean

Usage


company.ecore
(ECORE Metamodel)
1
ECORE
to C++
generator
C++ source files
2
C++
compiler
libemf4cpp-company.so
(Runtime library)
<conforms>
CompanyModel.xmi
(a Company model)
3
C++ user
application
<linked>
<linked>
libemf4cpp-ecore.so
libemf4cpp-ecorecpp.so
(Runtime libraries)

Codegeneration Strategy

- For each class in the ecore model, a C++ class is generated
- Objects are managed by boost::intrusive_ptr<>
- Containers are based on std::vector<>
- Classes are PODs – more or less

Codegeneration Strategy

- For each class in the ecore model, a C++ class is generated
- Objects are managed by boost::intrusive_ptr<>
- Containers are based on std::vector<>
- Classes are PODs – more or less
- Extend the generated code by
  - *PROTECTED REGION*s are kept by the code generator
  - Derive and instantiate your own classes by *factory injection*

Codegeneration Strategy

```
namespace company {

class Department : public virtual ::ecore::EObject {
public:
    Department();
    virtual ~Department();

    // Operations

    // Attributes
    virtual ::ecore::EInt getNumber () const;
    virtual void setNumber (::ecore::EInt _number);

    // References
    virtual const ::ecorecpp::mapping::EList< ::company::
        Employee_ptr >& getEmployees () const;
    virtual ::ecorecpp::mapping::EList< ::company::Employee_ptr >&
        getEmployees ();

    virtual ::company::Employee_ptr getManager () const;
    virtual void setManager (::company::Employee_ptr _manager);
```

Codegeneration Strategy

```
    /* This is the same value as getClassifierId() returns, but as a
     * static value it can be used in template expansions. */
    static const int classifierId = CompanyPackage::DEPARTMENT;

    /*PROTECTED REGION ID(Department) START*/
    // Please, enable the protected region if you add manually
        written code.
    // To do this, add the keyword ENABLED before START.
    /*PROTECTED REGION END*/

protected:
    // Attributes
    ::ecore::EInt m_number;

    // References
    std::shared_ptr<::ecorecpp::mapping::EList< ::company::
        Employee_ptr >> m_employees;
    ::company::Employee_ptr m_manager;
};

} //namespace Company
```

Reference Handling

- All references are handled as boost::intrusive_ptr<>
- Multiplicity $>$ 1 implemented by spezialized container *EList*
- Opposite relation
- Containment relation

Reference Handling

- All references are handled as boost::intrusive_ptr<>
- Multiplicity > 1 implemented by spezialized container *EList*
- Opposite relation
  - Changing a relation implicitly changes relation at referenced object
  - Opposite relations can have different multiplicity
  - Example: Employee is a manager for multiple Departments
    `Employee::getManagedDepartments()` $\Longleftrightarrow$ `Department::getManager()`
- Containment relation

Reference Handling

- All references are handled as boost::intrusive_ptr<>
- Multiplicity > 1 implemented by spezialized container *EList*
- Opposite relation
  - Changing a relation implicitly changes relation at referenced object
  - Opposite relations can have different multiplicity
  - Example: Employee is a manager for multiple Departments
    `Employee::getManagedDepartments()` $\Longleftrightarrow$ `Department::getManager()`
- Containment relation
  - Express ownership
  - Implicit opposite relation: eContainer()
  - Removing a child **must not** implicitly delete the child

Reference Handling

- All references are handled as boost::intrusive_ptr<>
- Multiplicity > 1 implemented by spezialized container *EList*
- Opposite relation
  - Changing a relation implicitly changes relation at referenced object
  - Opposite relations can have different multiplicity
  - Example: Employee is a manager for multiple Departments `Employee::getManagedDepartments()` ⟺ `Department::getManager()`
- Containment relation
  - Express ownership
  - Implicit opposite relation: eContainer()
  - Removing a child **must not** implicitly delete the child
- Implemented by generated code in combination with EList container

libemf4cpp-ecorecpp

- Serialization / deserialization as XML Metadata Interchange (XMI)
- Resources and ResourceSets
    - Resources reference URIs, e.g. files
    - Instances can be split over Resources
- Tree traversal
- Notification framework
    - Callbacks for modifications of the model
    - Useful for Model-View-Controller UIs

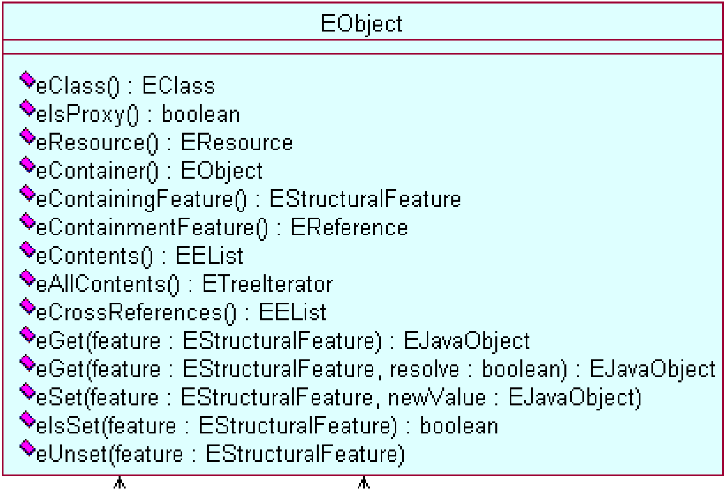
libemf4cpp-ecorecpp

- Serialization / deserialization as XML Metadata Interchange (XMI)
- Resources and ResourceSets
  - Resources reference URIs, e.g. files
  - Instances can be split over Resources
- Tree traversal
- Notification framework
  - Callbacks for modifications of the model
  - Useful for Model-View-Controller UIs
- Generic, model-agnostic implementation

Reflection API

- libemf4cpp-ecore
  - C++ implementation of the ecore metamodel
  - Created during bootstrap build of the codegenerator
- Every generated class is derived from ecore::EObject

Operations Defined for ecore::EObject



EObject

eClass() : EClass
eIsProxy() : boolean
eResource() : EResource
eContainer() : EObject
eContainingFeature() : EStructuralFeature
eContainmentFeature() : EReference
eContents() : EEList
eAllContents() : ETreeIterator
eCrossReferences() : EEList
eGet(feature : EStructuralFeature) : EJavaObject
eGet(feature : EStructuralFeature, resolve : boolean) : EJavaObject
eSet(feature : EStructuralFeature, newValue : EJavaObject)
eIsSet(feature : EStructuralFeature) : boolean
eUnset(feature : EStructuralFeature)

Know Your Metaclass

- For each generated class, the runtime system initializes an instance of ecore::EClass
- Instances know their ecore::EClass: `employee->eClass()`

Know Your Metaclass

- For each generated class, the runtime system initializes an instance of ecore::EClass
- Instances know their ecore::EClass: `employee->eClass()`
- Examine the class definition by
  - Class hierarchy: eSuperTypes(), eAllSuperTypes()
  - Members: eStructuralFeatures(), eAllStructuralFeatures() as well as eAttributes(), eReferences(), ...

Know Your Metaclass

- For each generated class, the runtime system initializes an instance of ecore::EClass
- Instances know their ecore::EClass: `employee->eClass()`
- Examine the class definition by
  - Class hierarchy: eSuperTypes(), eAllSuperTypes()
  - Members: eStructuralFeatures(), eAllStructuralFeatures() as well as eAttributes(), eReferences(), ...
- Access instances by generic APIs
  - Access to members: eGet(someFeature), eSet(someFeature, newValue)
  - The parent: eContainer(), eContainingFeature()
  - All children: eContents(), eAllContents()

Licensing: LGPL

- The codegenerator, the runtime libraries and the generated ecore implementation: published under LGPL
- Code generated from an ecore model: It's yours!
  (Actually most lawyers think, it belongs to the owner of the model)

Project URL

First release in 2010
Research project at University of Murcia, Spain
https://github.com/catedrasaes-umu/emf4cpp

Presented features are implemented in a fork
https://github.com/mdoerfel/emf4cpp

Participation is welcome!

Contact me for questions: doerfel@inchron.com