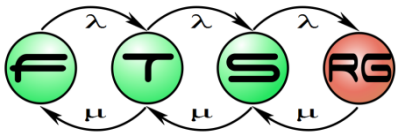
Domain-specific modeling based on the Eclipse Modeling Framework



Motivation

- Modern software/system development
  - High complexity
    - Increasing development time
    - Increasing cost
  - Diversity
    - Domain
    - Requirements
    - Implementation tools and techniques
  - Changes
    - Environment
    - Requirements
    - Bug fixes

General Purpose Languages

- Multiple languages
  - C, Java, Javascript, Go, Ruby, Python, …
- Capable of solving all problems (Turing-complete)
- Domain expert cannot efficiently use it
  - Programming knowledge needed!

Domain-specific language

A DSL is a **focused, processable** language for describing a specific concern when building a system in a specific domain. The **abstractions** and **notations** used are natural/suitable for the **stakeholders** who specify that particular concern.

Markus Voelter: DSL Design

	GPL	DSL
Domain	Large and complex	Smaller and well-defined
Language Size	Large	Small
Turing completeness	Always	Often not
User-defined abstractions	Sophisticated	Limited
Execution	Via intermediate GPL	Native
Lifespan	Years to decades	Months to years
Designed by	Guru or committee	Few enginers, domain experts
User community	Large, anonymous and widespread	Small, accessible and local
Evolution	Slow, often standardized	Fast-paced
Deprecation, incompatible changes	Almost impossible	Feasible

	GPL	DSL
Do		Smaller and well-defined
L		Small
T		Often not
U a		Limited
E		Native
Lif		Months to years
Designed by	Guru or co	Few enginers, domain experts
User community	Large, anonymous and widespread	all, accessible and local
Evolution	Slow, often standardized	Fast-paced
Deprecation, incompatible changes	Almost impossible	Feasible

- More properties on the DSL side
  - Easier to develop language
  - More focused
- No clear boundary between GPL and DSL

Example: Test Language for VIATRA2

- Goal: test suite definition
  - Parser tests – code and error messages
  - Interpreter tests – code, input and expected textual output

Example – DSL 1.

- Fixed directory structure
  - **desc** folder: Test case description files
  - **out** folder: Test case output files
  - **vpml** folder: Input model
  - **vtcl** folder: Folder containing control script files
  - **.suite** file: The test suite descriptor file
- Is this structure ok?
- It depends
  - Hard to use
  - Easy to parse/process

Example – DSL 1.

- Fixed directory structure
  - **desc** folder: Test
  - **out** folder: Test case output
  - **vpml** folder: Input model
  - **vtcl** folder: Folder containing
  - **.suite** file: The test suite desc
- Is this structure ok?
- It depends
  - Hard to use
  - Easy to parse/process

Example – DSL 1.

- Fixed directory structure
  - **desc** folder: Test case description files
  - **out** folder: Test case output files
  - **vpml** folder: Input model
  - **vtcl** folder: Folder containing control script files
  - **.suite** file: The test suite descriptor file
- Is this structure ok?
- It depends
  - Hard to use
  - Easy to parse/process

Example – DSL 1.

- Fixed directory structure
  - **desc** folder: Test case description files
  - **out** folder: Test case output files
  - **vpml** f
  - **vtcl** fo
  - **.suite**
- Is this st
- It depends
  - Hard to use
  - Easy to parse/process

out: tab-separated file

6 ERROR PARSER_ERROR expected instead of this input

Example – DSL 1.

- Fixed directory structure
  - **desc** folder: Test case description files
  - **out** folder: Test case output files
  - **vpml** folder: Input model
  - **vtcl** folder: Folder containing control script files
  - **.suite** file: The test suite descriptor file
- Is this structure ok?
- It depends
  - Hard to use
  - Easy to parse/process

Example – DSL 1.

- Fixed directory structure
  - **desc** folder: Test case description files
  - **out** folder: Test case output files
  - **vpml** folder: Input
  - **vtcl** folder: Folder
  - **.suite** file: The test suite descriptor file
- Is this structure ok?
- It depends
  - Hard to use
  - Easy to parse/process

vpml and vtcl: VIATRA-specific input files

Example – DSL 1.

- Fixed directory structure
  - **desc** folder: Test case description files
  - **out** folder: Test case output files
  - **vpml** folder: Input model
  - **vtcl** folder: Folder containing control script files
  - **.suite** file: The test suite descriptor file
- Is this structure ok?
- It depends
  - Hard to use
  - Easy to parse/process

Example – DSL 1.

- Fixed directory structure
  - **desc** folder: Test case description files
  - **out** folder: Test case output files
  - **vpml** folder: Input model
  - **vtcl**
  - **.s**
- Is thi
- It de
  - Ha
  - Eas

.suite file: tab-separated values

```
PARSER HorriblyWrongDefSyntax asmfunction1.desc
PARSER WrongDefSyntax asmfunction2.desc
PARSER DuplicateSameArity asmfunction3.desc
```

Example – DSL 1.

- Fixed directory structure
  - **desc** folder: Test case description files
  - **out** folder: Test case output files
  - **vpml** folder: Input model
  - **vtcl** folder: Folder containing control script files
  - **.suite** file: The test suite descriptor file
- Is this structure ok?
- It depends
  - Hard to use
  - Easy to parse/process

Example – DSL 1.

- Fixed directory structure
  - **desc** folder: Test case description files
  - **out** folder: Test case output files
  - **vpml** folder: Input model
  - **vtcl** folder: Folder containing control script files
  - **.suite** file: The test suite descriptor file
- Is this structure ok?
- It depends
  - Hard to use
  - Easy to parse/process

Example – DSL 1.

- Fixed directory structure
  - **desc** folder: Test case description files
  - **out** folder: Test case output files
  - **vpml** folder: Input model
  - **vtcl** folder: Folder containing control script files
  - **.suite** file: The test suite descriptor file
- Is this structure ok?
- It depends
  - Hard to use
  - Easy to parse/process

Example – Java API

```
public class MissingSeqTest extends ParserTest {

  @Override
  protected String getVtcl() {
    return "/suites/asmrules/asmrule1.vtcl";
  }

  @Override
  protected void buildReferenceOutput() {
    ErrorSeverity es = ErrorSeverity.ERROR;
    ErrorKind ek = ErrorKind.PARSE_ERROR;
    Location l = new Location(3, -1, -1, -1);
    String message = "expected after this token";
    addOutput(new ErrorInformation(message,
"org.eclipse.viatra2.marker", ek, l, es));
  }

}
```

Example – Java API

```
public class MissingSeqTest extends Pars
  @Override
  protected String getVtcl() {
    return "/suites/asmrules/asmrule1.vtcl";
  }

  @Override
  protected void buildReferenceOutput() {
    ErrorSeverity es = ErrorSeverity.ERROR;
    ErrorKind ek = ErrorKind.PARSE_ERROR;
    Location l = new Location(3, -1, -1, -1);
    String message = "expected after this token";
    addOutput(new ErrorInformation(message,
"org.eclipse.viatra2.marker", ek, l, es));
  }

}
```

Test base class

Example – Java API

```
public class MissingSeqTest extends ParserTest {

  @Override
  protected String getVtcl() {
    return "/suites/asmrules/asmrule1.vtcl";
  }

  @Override
  protected void buildReferenceOutput() {
    ErrorSeverity es = ErrorSeverity.ERROR;
    ErrorKind ek = ErrorKind.PARSE_ERROR;
    Location l = new Location(3, -1, -1, -1);
    String message = "expected after this token";
    addOutput(new ErrorInformation(message,
"org.eclipse.viatra2.marker", ek, l, es));
  }

}
```

Example – Java API

```
public class MissingSeqTest extends ParserTest {

  @Override
  protected String getVtcl() {
    return "/suites/asmrules/asmrule1.vtcl"
  }

  @Override
  protected void buildReferenceOutput() {
    ErrorSeverity es = ErrorSeverity.ERROR;
    ErrorKind ek = ErrorKind.PARSE_ERROR;
    Location l = new Location(3, -1, -1, -1);
    String message = "expected after this token";
    addOutput(new ErrorInformation(message,
"org.eclipse.viatra2.marker", ek, l, es));
  }

}
```

Test input

Example – Java API

```
public class MissingSeqTest extends ParserTest {

  @Override
  protected String getVtcl() {
    return "/suites/asmrules/asmrule1.vtcl";
  }

  @Override
  protected void buildReferenceOutput() {
    ErrorSeverity es = ErrorSeverity.ERROR;
    ErrorKind ek = ErrorKind.PARSE_ERROR;
    Location l = new Location(3, -1, -1, -1);
    String message = "expected after this token";
    addOutput(new ErrorInformation(message,
"org.eclipse.viatra2.marker", ek, l, es));
  }

}
```

Example – Java API

```
public class MissingSeqTest extends ParserTest {

  @Override
  protected String getVtcl() {
    return "/suites/asmrules/asmrule1.vtcl";
  }

  @Override
  protected void buildReferenceOutput() {
    ErrorSeverity es = ErrorSeverity.ERROR
    ErrorKind ek = ErrorKind.PARSE_ERROR;
    Location l = new Location(3, -1, -1,
    String message = "expected after this token";
    addOutput(new ErrorInformation(message,
"org.eclipse.viatra2.marker", ek, l, es));
  }

}
```

Test output

Example – Java API

```
public class MissingSeqTest extends ParserTest {

  @Override
  protected String getVtcl() {
    return "/suites/asmrules/asmrule1.vtcl";
  }

  @Override
  protected void buildReferenceOutput() {
    ErrorSeverity es = ErrorSeverity.ERROR;
    ErrorKind ek = ErrorKind.PARSE_ERROR;
    Location l = new Location(3, -1, -1, -1);
    String message = "expected after this token";
    addOutput(new ErrorInformation(message,
"org.eclipse.viatra2.marker", ek, l, es));
  }

}
```

Example – DSL 2.

```
suite ASM project "org.eclipse.viatra2.parser.tests {
  PARSER MissingSeq {
    vtcl "/suites/asmrules/asmrule1.vtcl”
    output error {
      PARSE ERROR 3 "expected after this token”
    }
  }
  PARSER DuplicateNested {
    vtcl "/suites/asmrules/asmrule2.vtcl”
    output error {
      VALIDATION ERROR 3 "Duplicate definition”,
      VALIDATION WARNING 3 "is not currently in use"
      }
  }
…
}
```

Example – DSL 2.

```
suite ASM project "org.eclipse.viatra2.parser.tests {
  PARSER M
    vtcl "/su                    smrule1.vtcl”
    output er
      PARSE ERROR 3 "expected after this token”
    }
  }
  PARSER DuplicateNested {
    vtcl "/suites/asmrules/asmrule2.vtcl”
    output error {
      VALIDATION ERROR 3 "Duplicate definition”,
      VALIDATION WARNING 3 "is not currently in use"
      }
  }
…
}
```

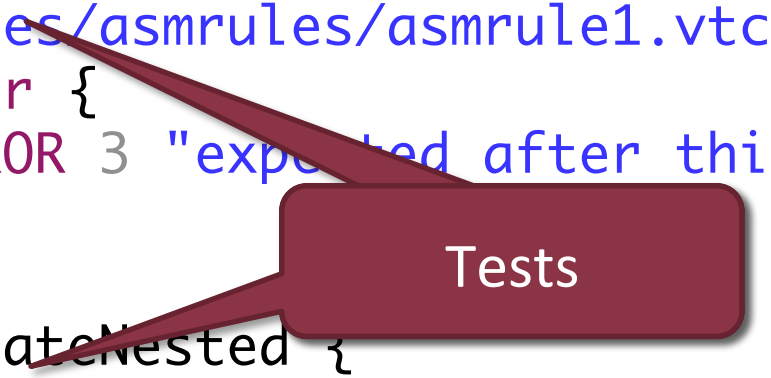
Suite definition

Example – DSL 2.

```
suite ASM project "org.eclipse.viatra2.parser.tests {
  PARSER MissingSeq {
    vtcl "/suites/asmrules/asmrule1.vtcl”
    output error {
      PARSE ERROR 3 "expected after this token”
    }
  }
  PARSER DuplicateNested {
    vtcl "/suites/asmrules/asmrule2.vtcl”
    output error {
      VALIDATION ERROR 3 "Duplicate definition”,
      VALIDATION WARNING 3 "is not currently in use"
      }
  }
…
}
```

Example – DSL 2.

```
suite ASM project "org.eclipse.viatra2.parser.tests {
  PARSER MissingSeq {
    vtcl "/suites/asmrules/asmrule1.vtcl"
    output error {
      PARSE ERROR 3 "expected after this token"
    }
  }
  PARSER DuplicateNested {
    vtcl "/suites/asmrules/asmrule2.vtcl"
    output error {
      VALIDATION ERROR 3 "Duplicate definition",
      VALIDATION WARNING 3 "is not currently in use"
      }
  }
…
}
```



Example – DSL 2.

```
suite ASM project "org.eclipse.viatra2.parser.tests {
  PARSER MissingSeq {
    vtcl "/suites/asmrules/asmrule1.vtcl”
    output error {
      PARSE ERROR 3 "expected after this token”
    }
  }
  PARSER DuplicateNested {
    vtcl "/suites/asmrules/asmrule2.vtcl”
    output error {
      VALIDATION ERROR 3 "Duplicate definition”,
      VALIDATION WARNING 3 "is not currently in use"
      }
  }
…
}
```

Example – DSL 2.

```
suite ASM project "org.eclipse.viatra2.pc
  PARSER MissingSeq {
    vtcl "/suites/asmrules/asmrule1.vtcl"
    output error {
      PARSE ERROR 3 "expected after this token"
    }
  }
  PARSER DuplicateNested {
    vtcl "/suites/asmrules/asmrule2.vtcl"
    output error {
      VALIDATION ERROR 3 "Duplicate definition",
      VALIDATION WARNING 3 "is not currently in use"
      }
  }
…
}
```

Input model

Example – DSL 2.

```
suite ASM project "org.eclipse.viatra2.parser.tests {
  PARSER MissingSeq {
    vtcl "/suites/asmrules/asmrule1.vtcl”
    output error {
      PARSE ERROR 3 "expected after this token”
    }
  }
  PARSER DuplicateNested {
    vtcl "/suites/asmrules/asmrule2.vtcl”
    output error {
      VALIDATION ERROR 3 "Duplicate definition”,
      VALIDATION WARNING 3 "is not currently in use"
      }
  }
…
}
```

Example – DSL 2.

```
suite ASM project "org.eclipse.viatra2.parser.tests {
  PARSER MissingSeq {
    vtcl "/suites/asmrules/asmrule1.vtcl”
    output error {
      PARSE ERROR 3 "expected after this t
    }
  }
  PARSER DuplicateNested {
    vtcl "/suites/asmrules/asmrule2.vtcl”
    output error {
      VALIDATION ERROR 3 "Duplicate definition”,
      VALIDATION WARNING 3 "is not currently in use"
      }
  }
…
}
```

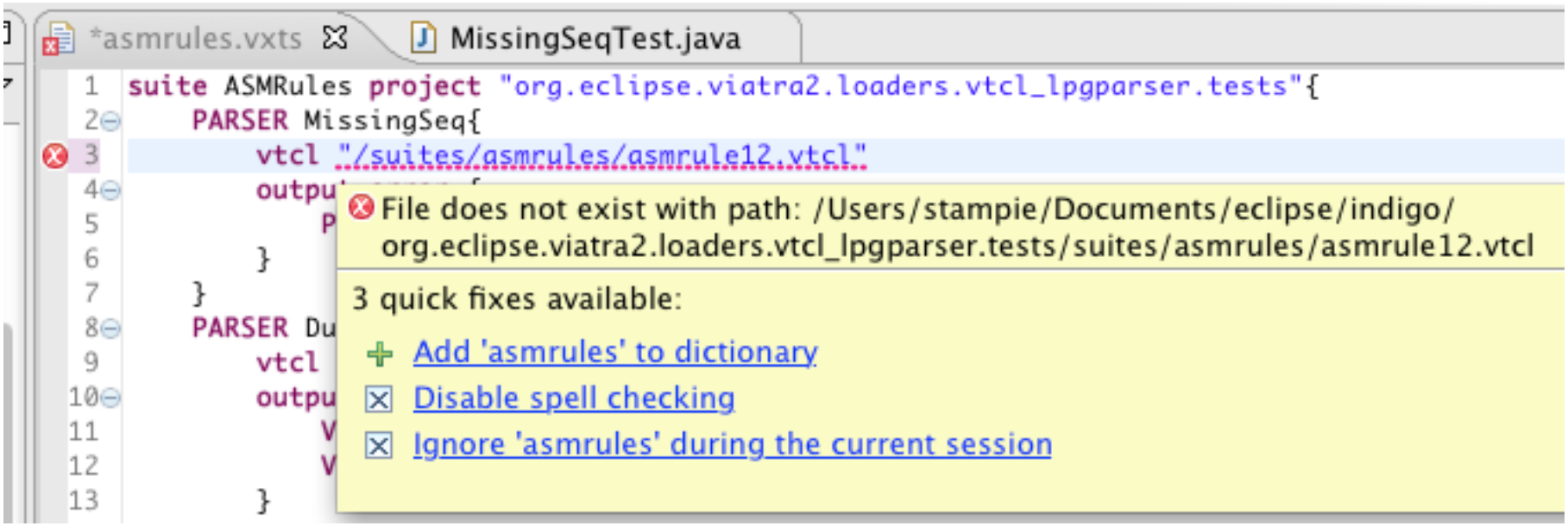
Output

Example – DSL 2.

```
suite ASM project "org.eclipse.viatra2.parser.tests {
  PARSER MissingSeq {
    vtcl "/suites/asmrules/asmrule1.vtcl”
    output error {
      PARSE ERROR 3 "expected after this token”
    }
  }
  PARSER DuplicateNested {
    vtcl "/suites/asmrules/asmrule2.vtcl”
    output error {
      VALIDATION ERROR 3 "Duplicate definition”,
      VALIDATION WARNING 3 "is not currently in use"
      }
  }
…
}
```

Example: Is DSL 2 better than Java API?

- Better error reporting
- Generated Java classes for test execution



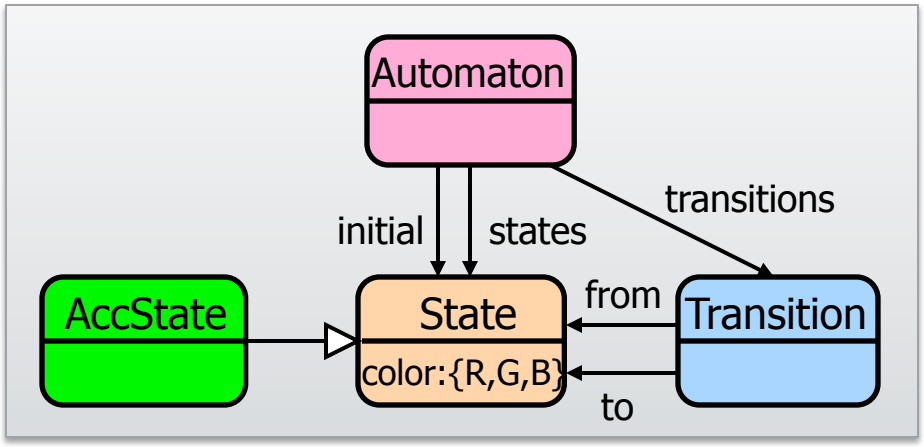
Metamodeling

Design paradigm for modeling languages

Designing modeling languages

- Metamodel: a model of models
  - Abstract syntax
  - Concrete syntax
  - Well-formedness rules
  - Behavioral (dynamic) semantics
  - Translation to other languages

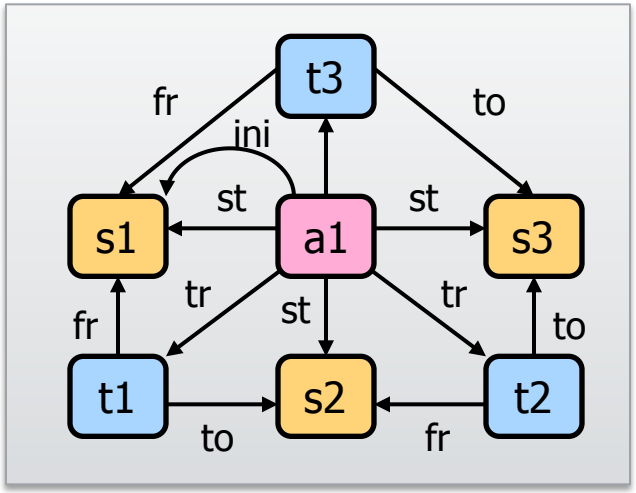
Meta- and instance models

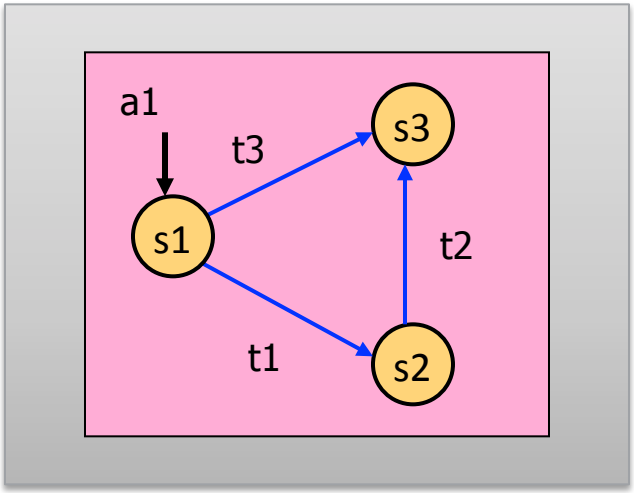


Metamodel level

Metamodel

Model level

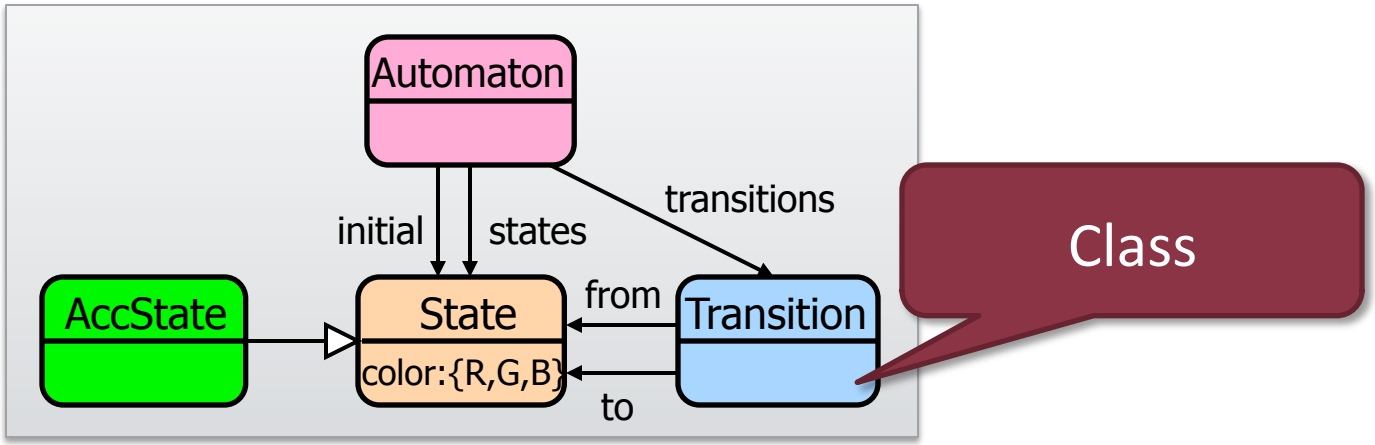




Abstract Syntax

Concrete syntax

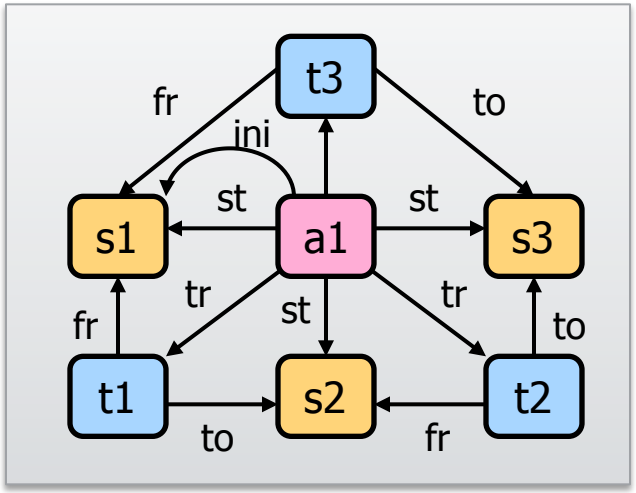
Meta- and instance models

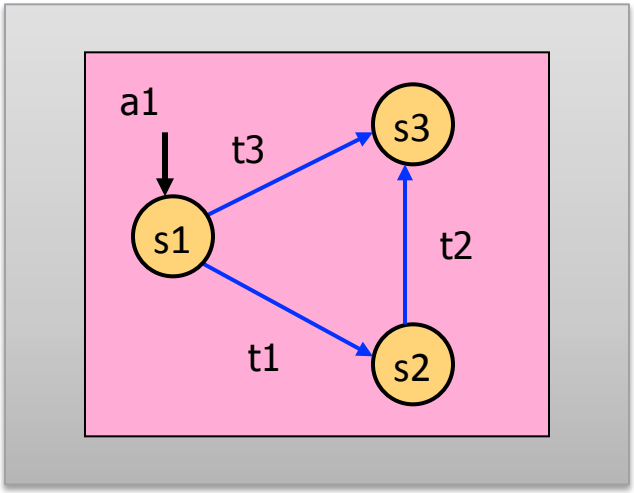


Metamodel level

Metamodel

Model level

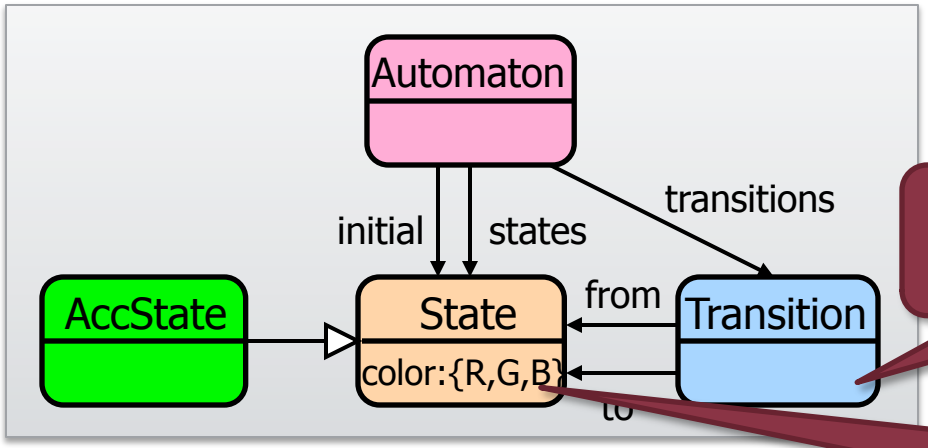




Abstract Syntax

Concrete syntax

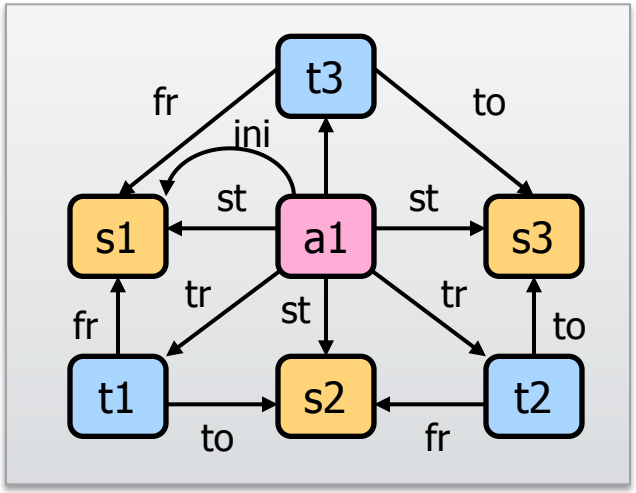
Meta- and instance models



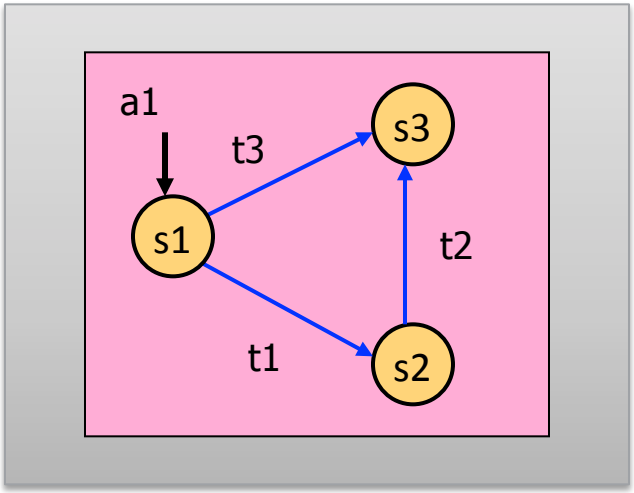
Metamodel

Metamodel level

Model level

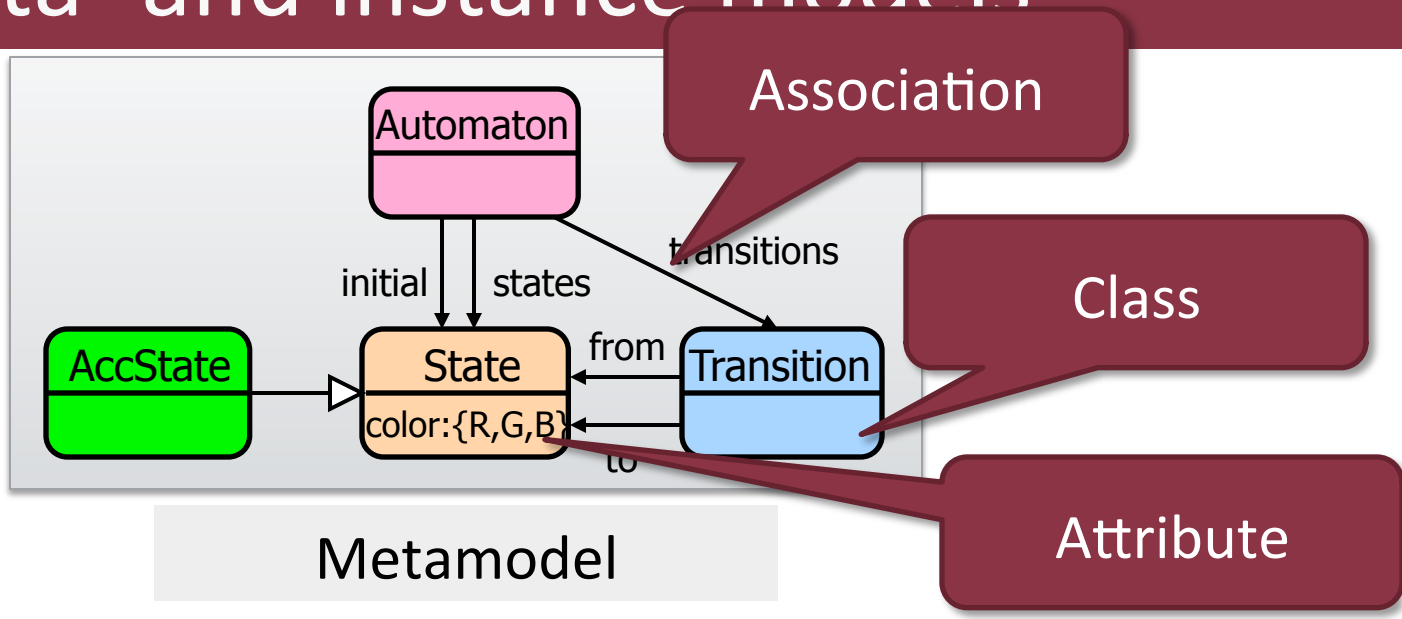


Abstract Syntax



Concrete syntax

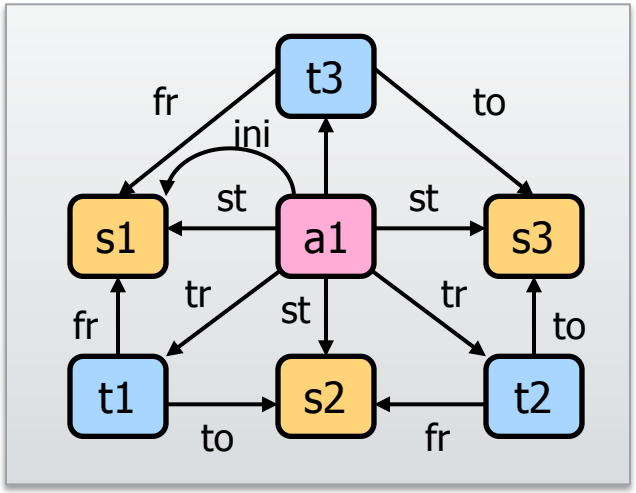
Meta- and instance models



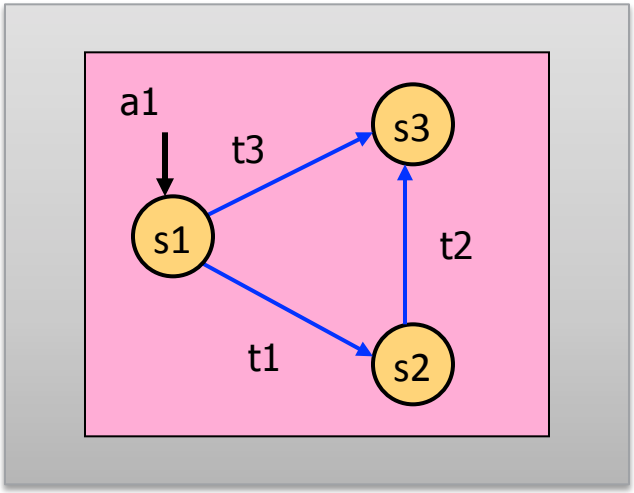
Metamodel level

Metamodel

Model level

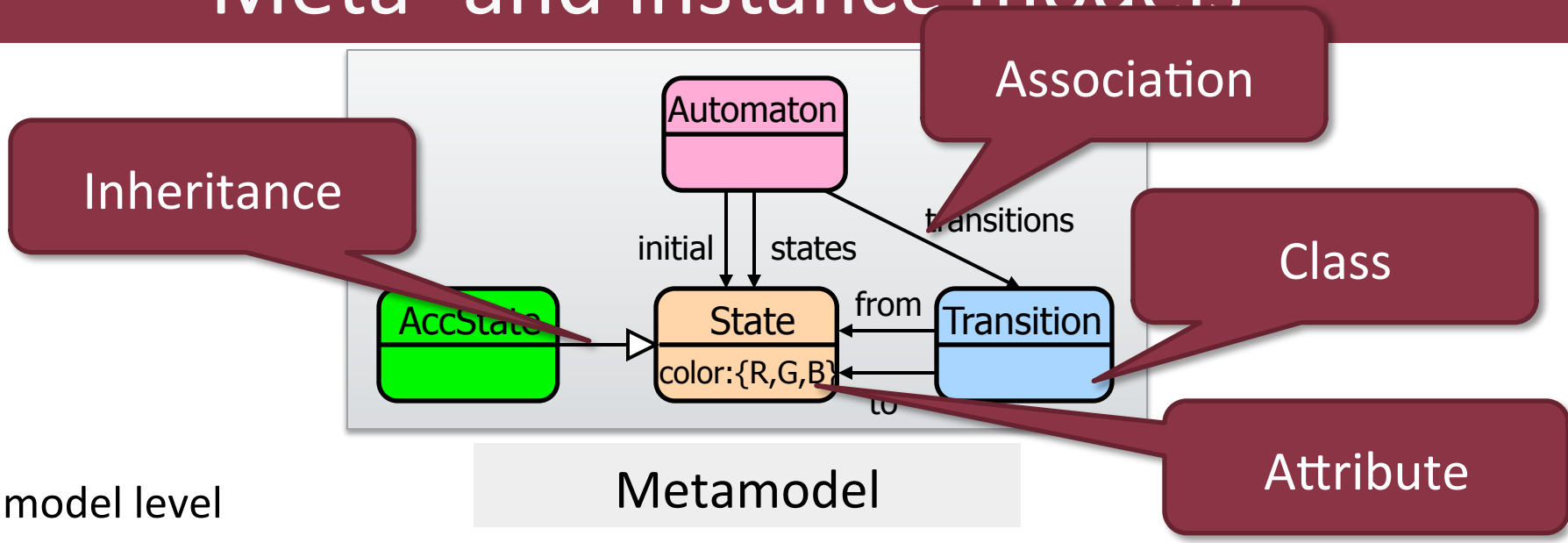


Abstract Syntax

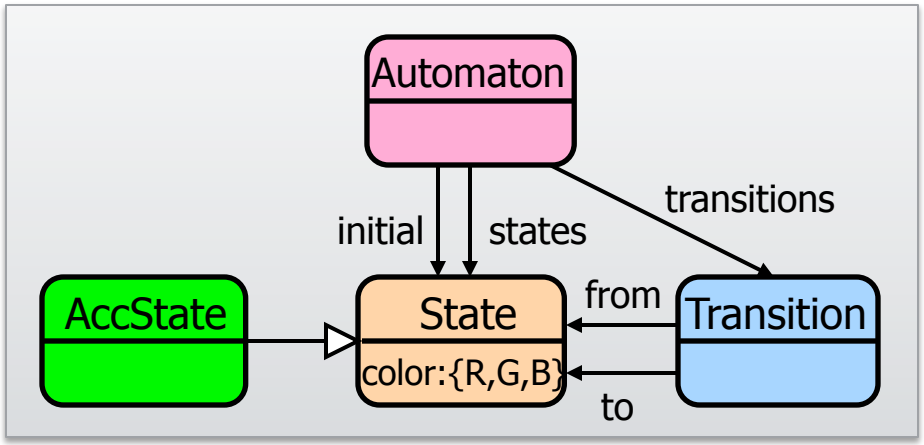


Concrete syntax

Meta- and instance models



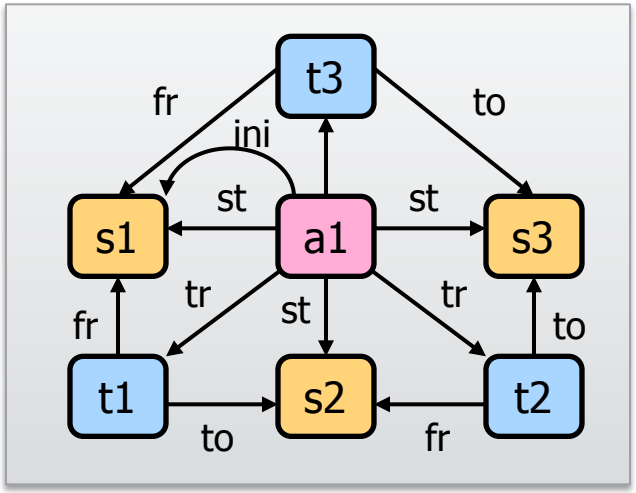
Meta- and instance models



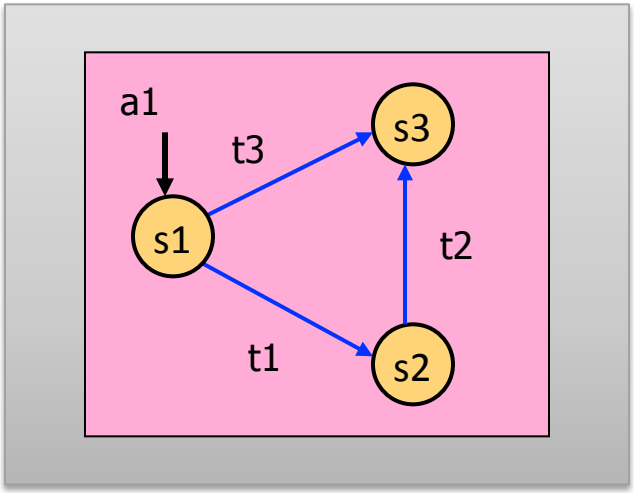
Metamodel level

Metamodel

Model level

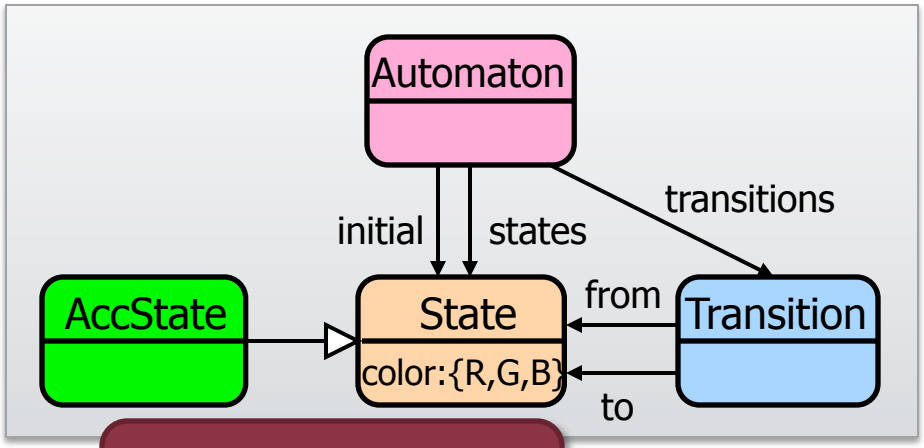


Abstract Syntax



Concrete syntax

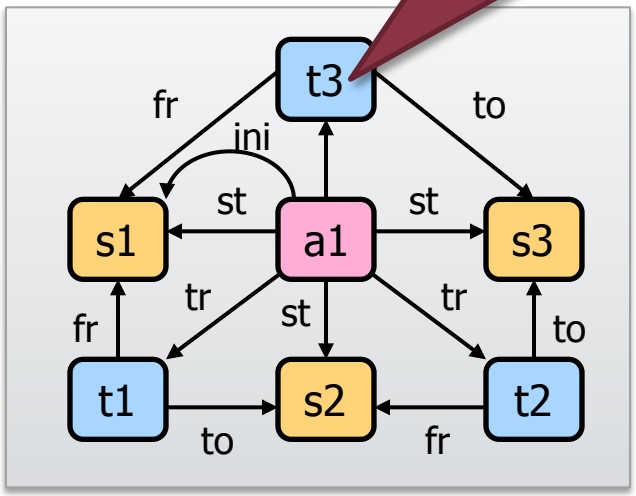
Meta- and instance models

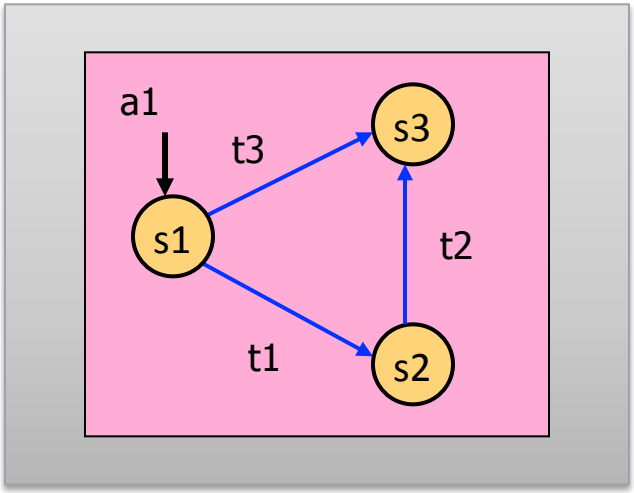


Object

Metamodel level

Model level

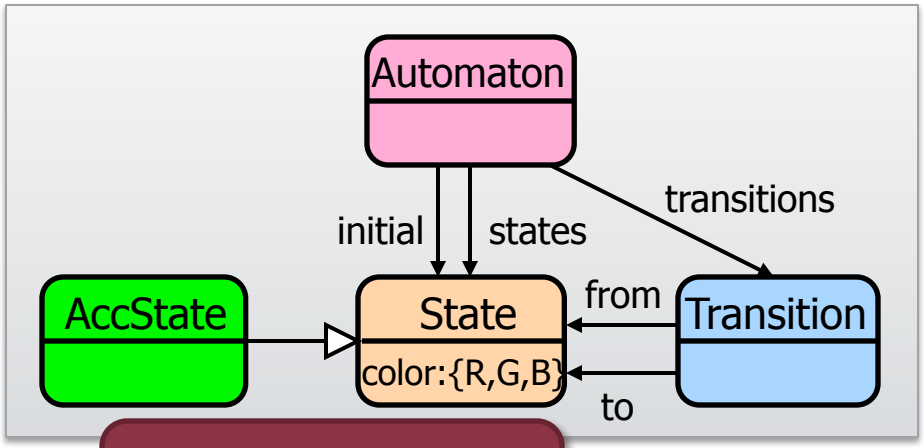




Abstract Syntax

Concrete syntax

Meta- and instance models

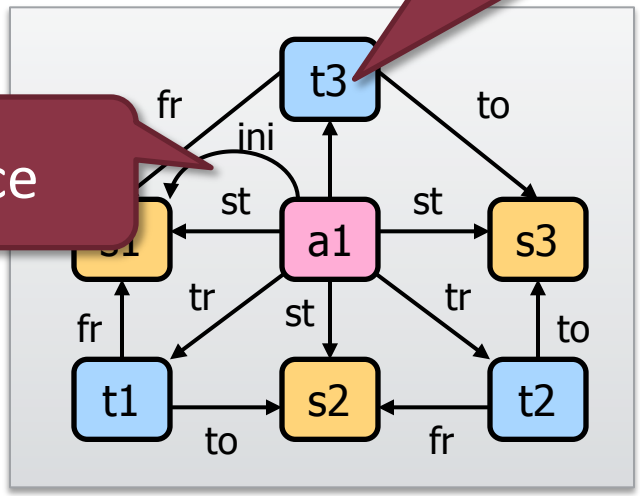


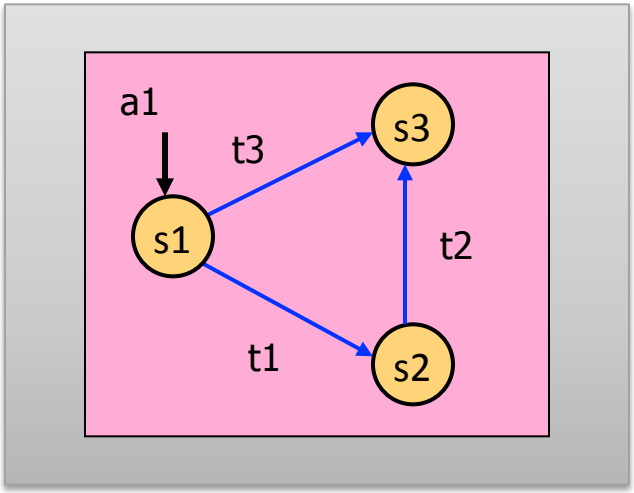
Metamodel level

Object

Model level

Reference

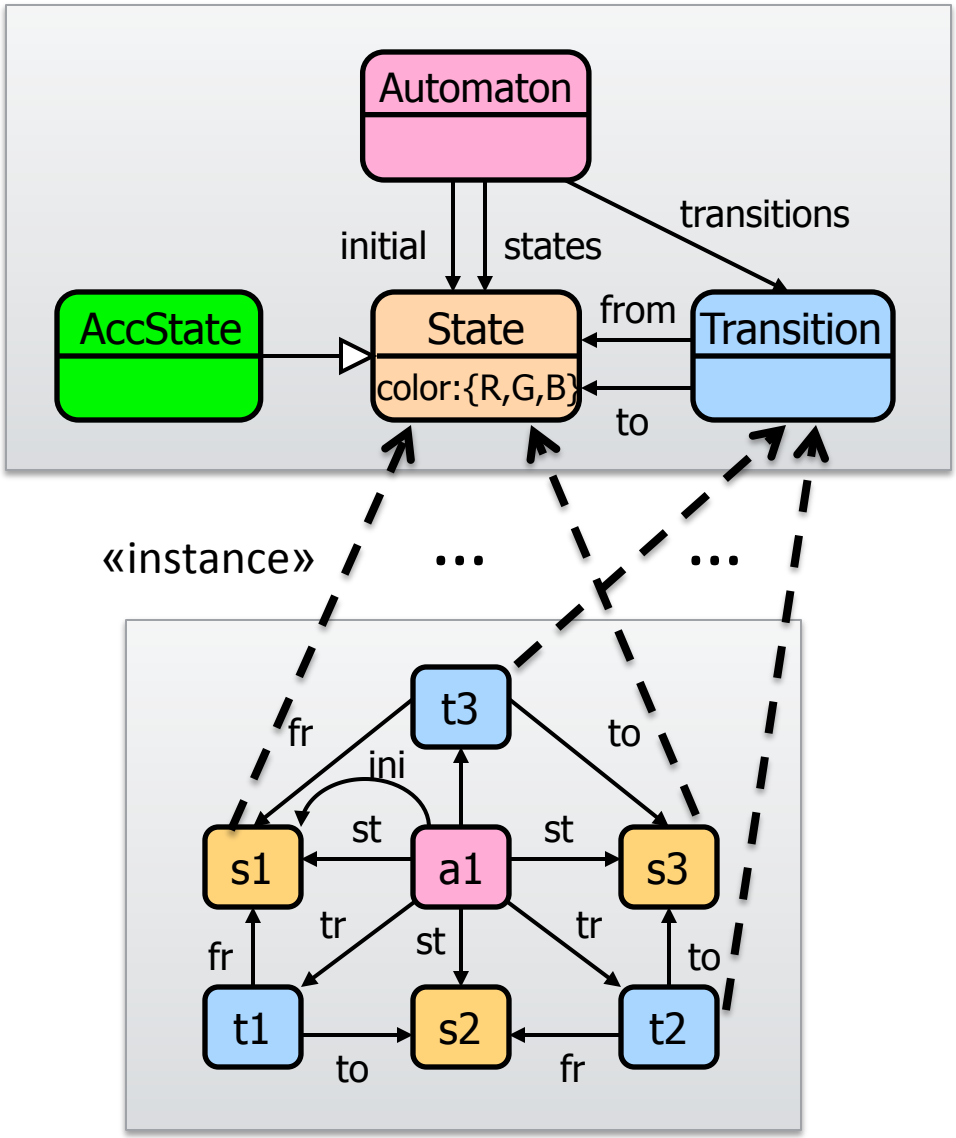




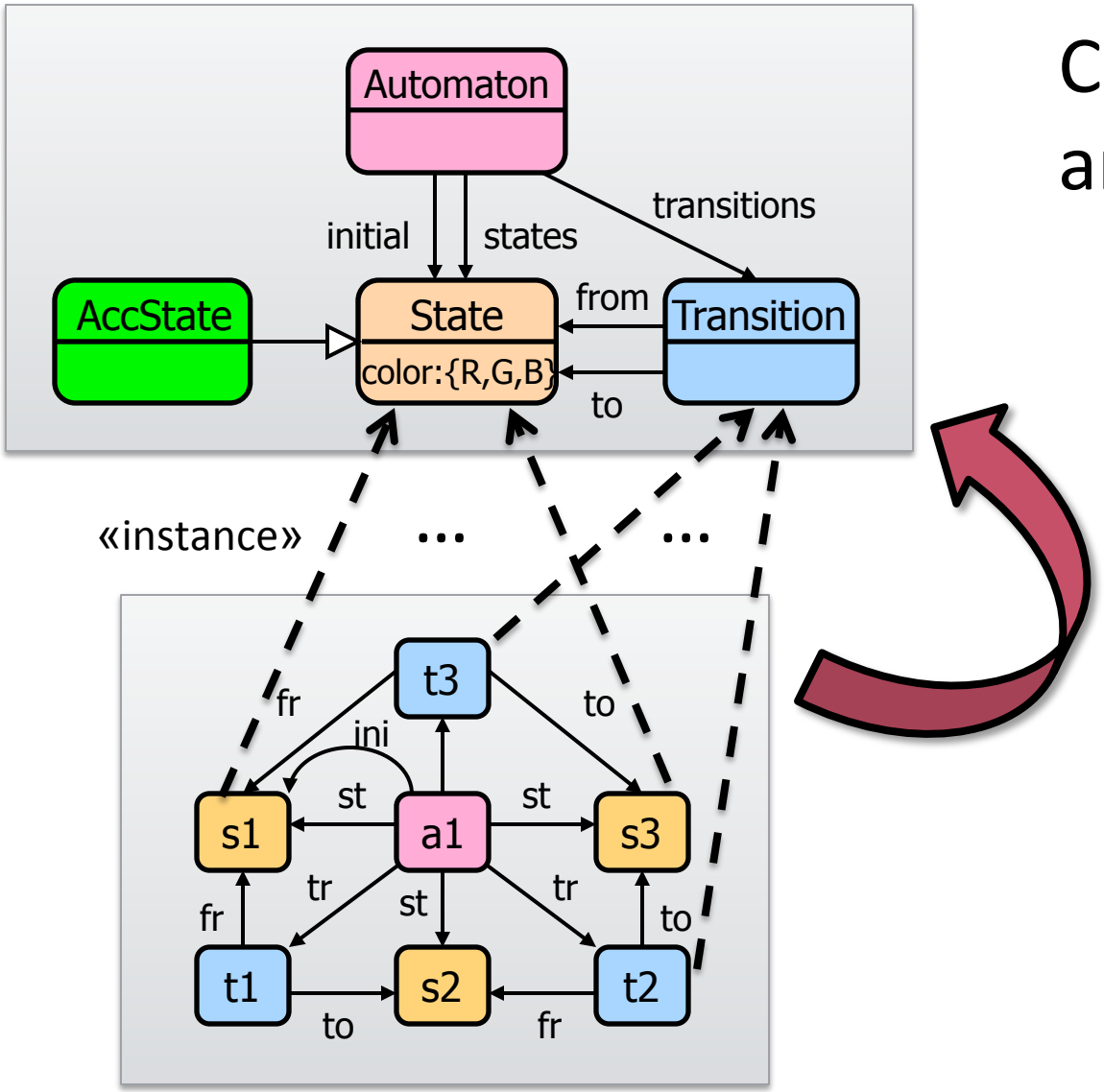
Abstract Syntax

Concrete syntax

Instantiation



Instantiation



Connects instance and metamodel

Concrete syntax

- A notation for the users
  - As the user “sees” the models
- Multiple concrete syntaxes for every language
  - Either different notations
  - Or different views

Graphical concrete syntax

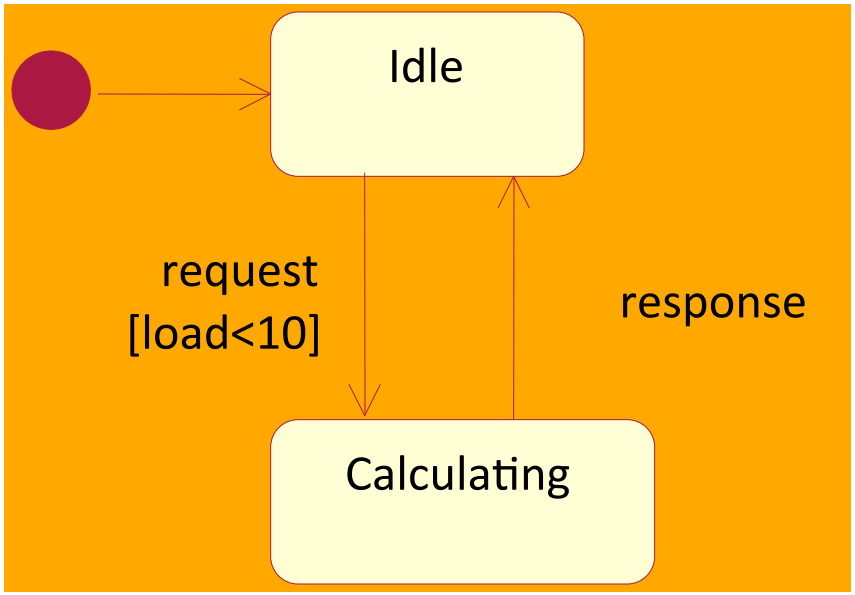
+ Easy to read
  + Intuitive, understandable notation
+ Safe(r) to write
  + Only syntactictically concrete models can be created
- Hard to write
  - Graphical editing is slower
- Scalability issues
  - Usability issues
  - Sometimes technological issues as well

Textual concrete syntax

+ Easy to write
  - Even complex expressions can be written quickly
+ Scales well
  - Files with 10k lines can be managed
– Hard to read
  - Understandability
  - Does not visualize connections
– Maintenance problematic
  - E.g. reference by name

Example concrete syntax for state matchines

- Graphical notation



- Textual notation

```
void request() {
 if(state == IDLE &&
     this.load < 10)
        state =
  CALCULATING;
}

void response() {
 if (state ==
  CALCULATING)
        state = IDLE;
}
```

Abstract syntax

- Defines the vocabulary of the languages
  - Terms
  - Relations between terms
    - Associations/references
    - Attributes
    - Abstractions/refinements (see taxonomies, ontologies)

Well-formedness rules

- Multiplicity constrants
  - One: 1
  - At most one: 0..1
  - Many: *
- Aggregation/Containment references
  - Specifies a containment hierarchy
  - At most one parent for each model element
- Language-specific constraints
  - E.g., unique names

Dynamic semantics

- Semantics:
  - Meaning of the terms of the language
  - How to understand the models
- Static semantics
  - What are the valid models?
  - See previous slides
- Dynamic semantics
  - Possible behaviour
  - State changes

Dynamic semantics – Main approaches

- Denotational semantics
  - Translating terms to another language
  - Similar to compilers
- Operation semantics
  - Modeling the behavior of terms
  - Similar to interpreters
- ~~Axiomatic semantics~~
  - ~~Based on logical formulaes~~
  - ~~Hard to understand/implement~~

Related Technologies

- Abstract syntax
  - EMF
- Well-formedness rules
  - EMF Validation, OCL
- Concrete syntax
  - Graphical editors: GEF/GMF/Graphiti
  - Textual editors: Xtext, EMFText
- Semantics, translation to other languages
  - Code generators: Acceleo, Xtend, (JET)
  - Model transformation: ATL, ETL, VIATRA2, ...

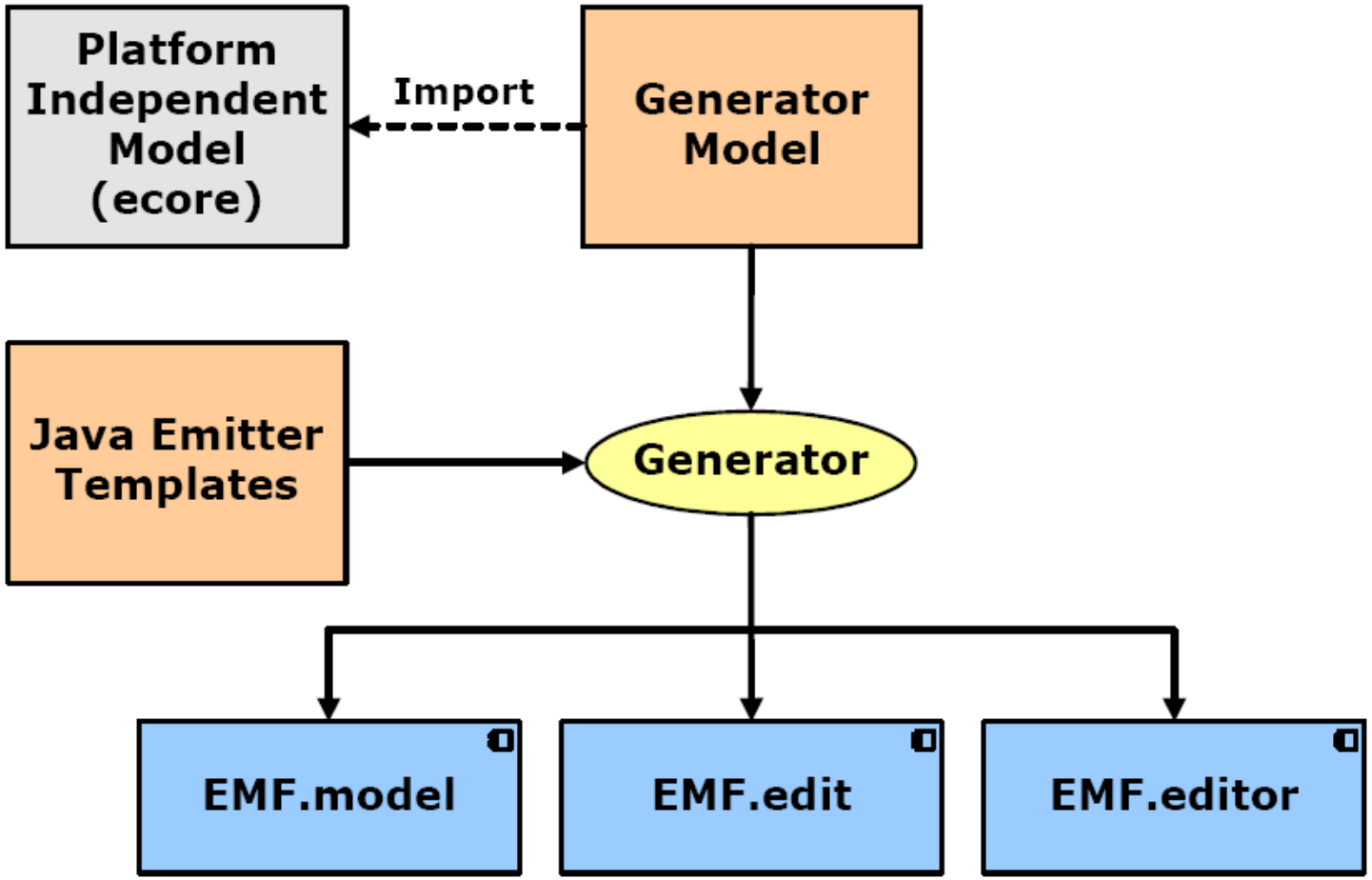
Eclipse Modeling Framework (EMF)

Metamodeling in Eclipse

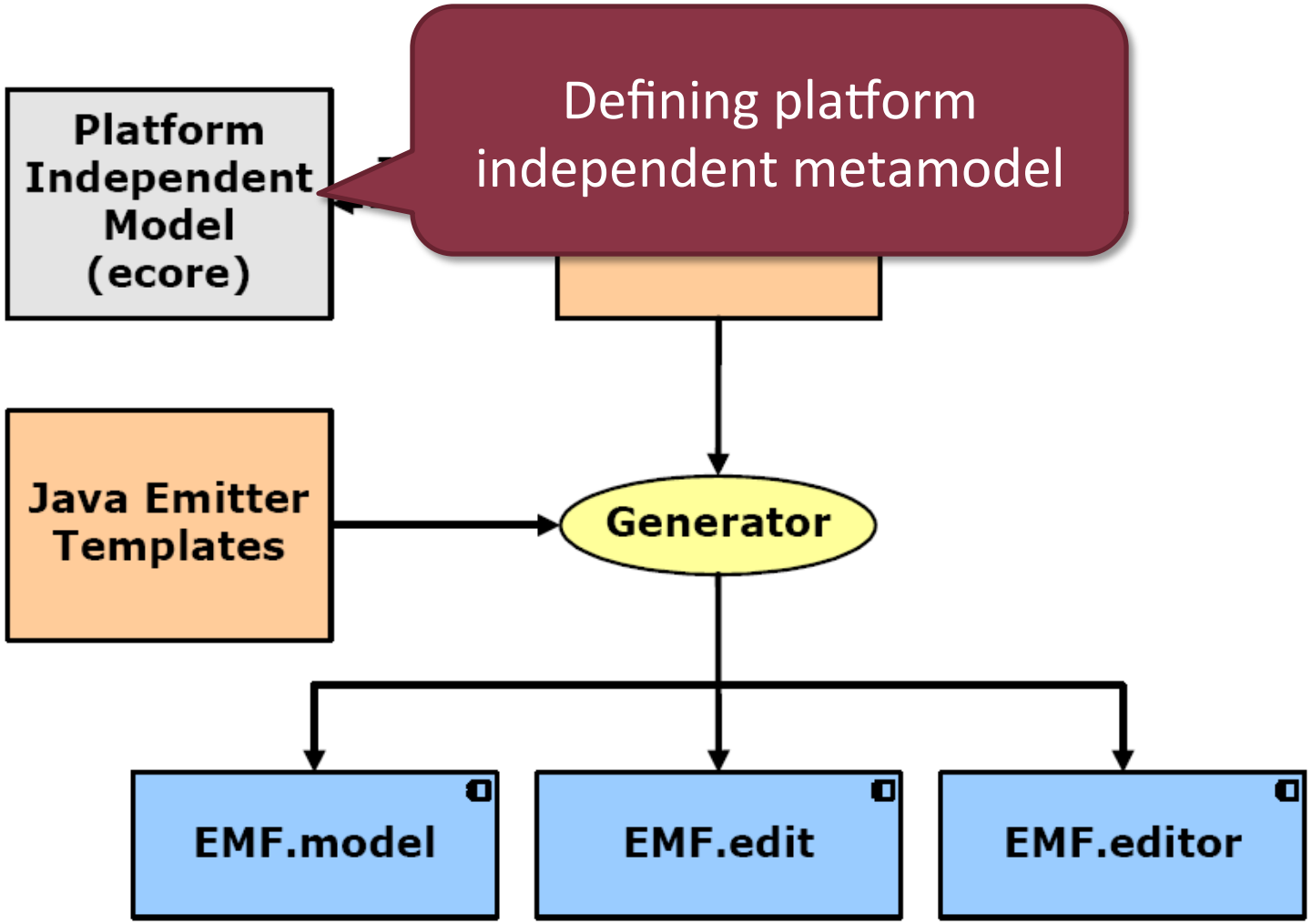
Eclipse Modeling Framework

- Modeling component for Eclipse
  - Supports the definition of DSLs
- Supports
  - Basic editing commands
  - Change notification
  - Undo/redo support
  - XML/XMI export-import
- Uses a simple metamodeling core
  - Called Ecore
  - Similar to MOF (used for UML)

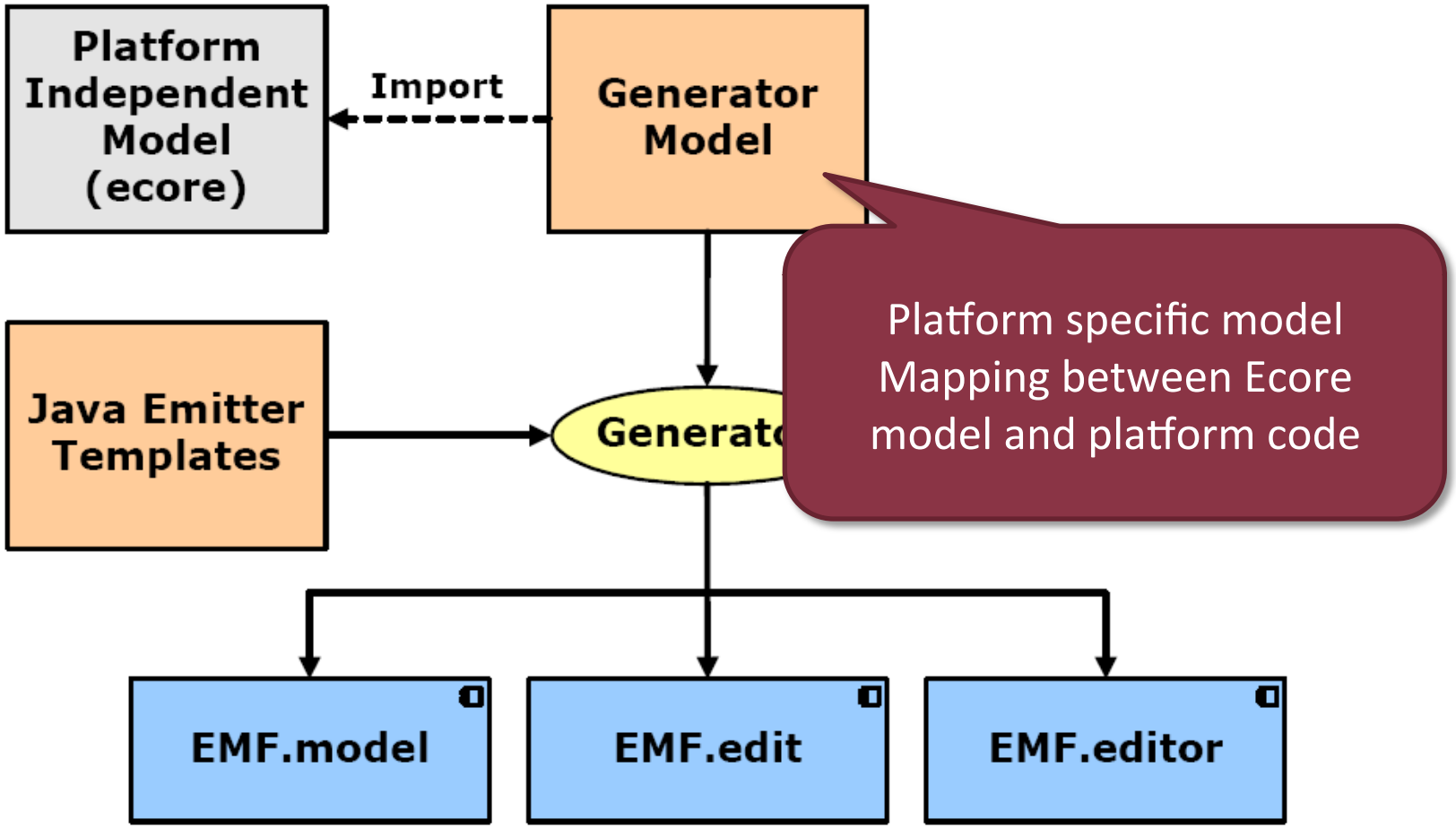
The EMF tooling



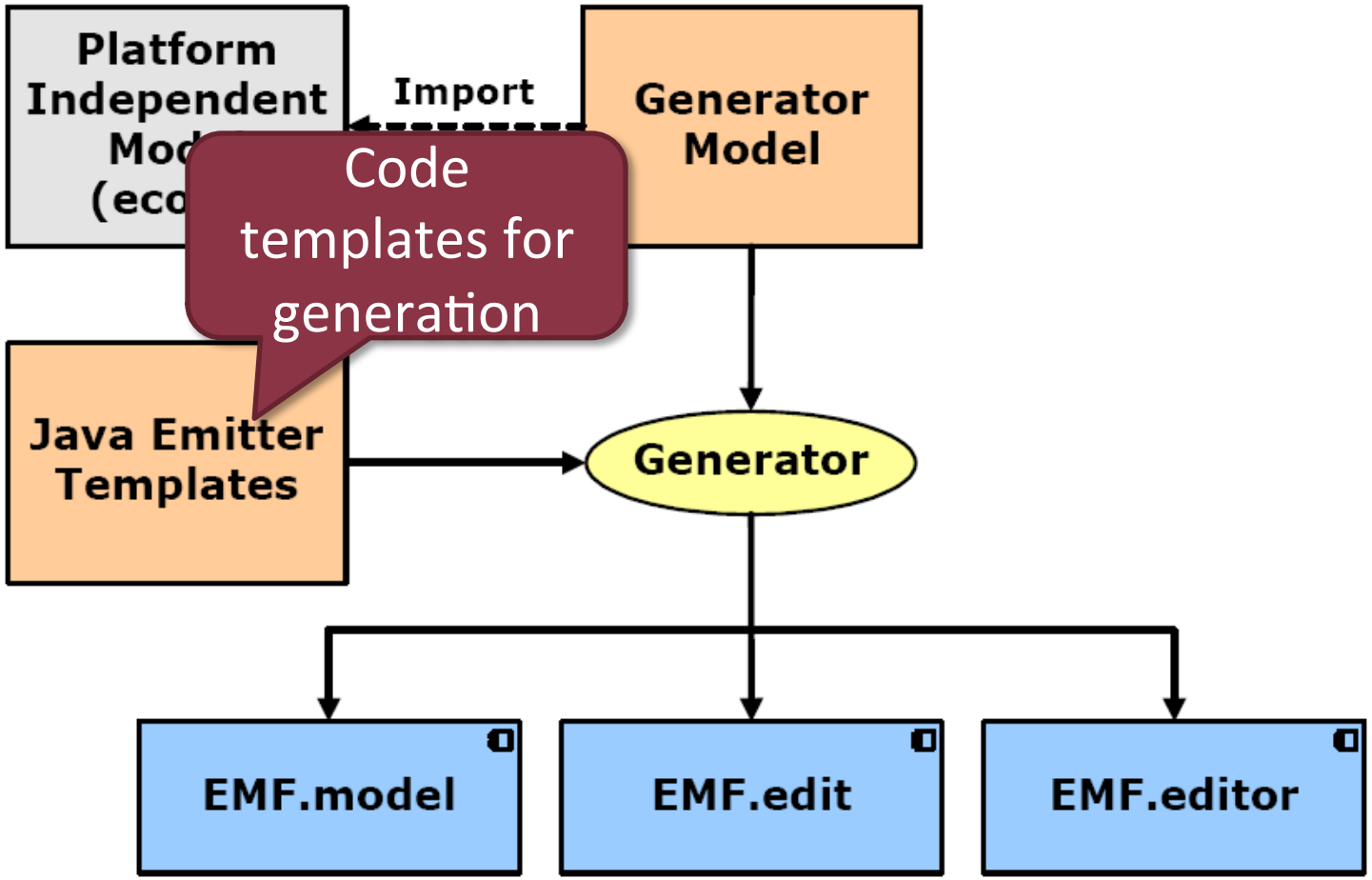
The EMF tooling



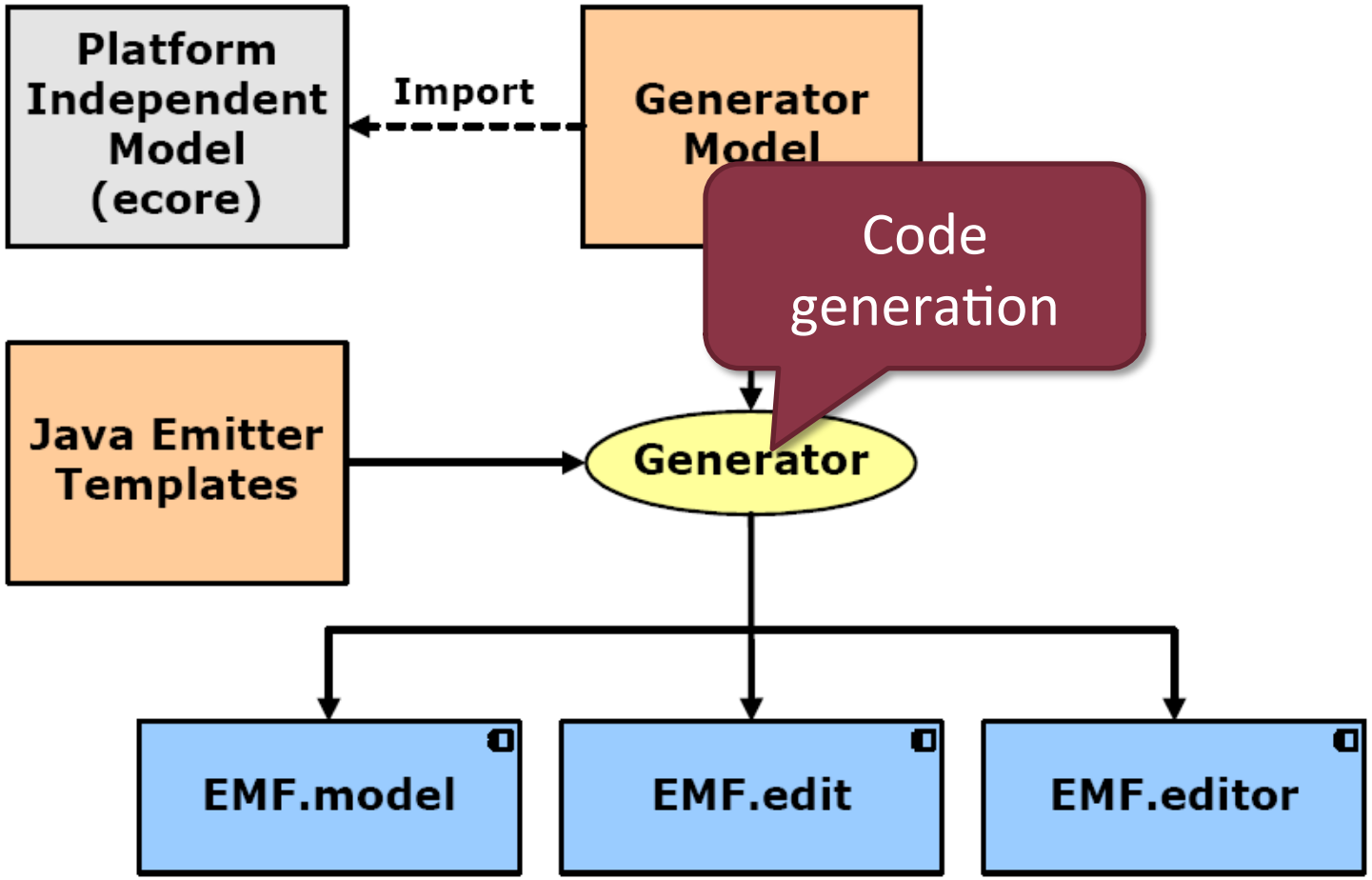
The EMF tooling



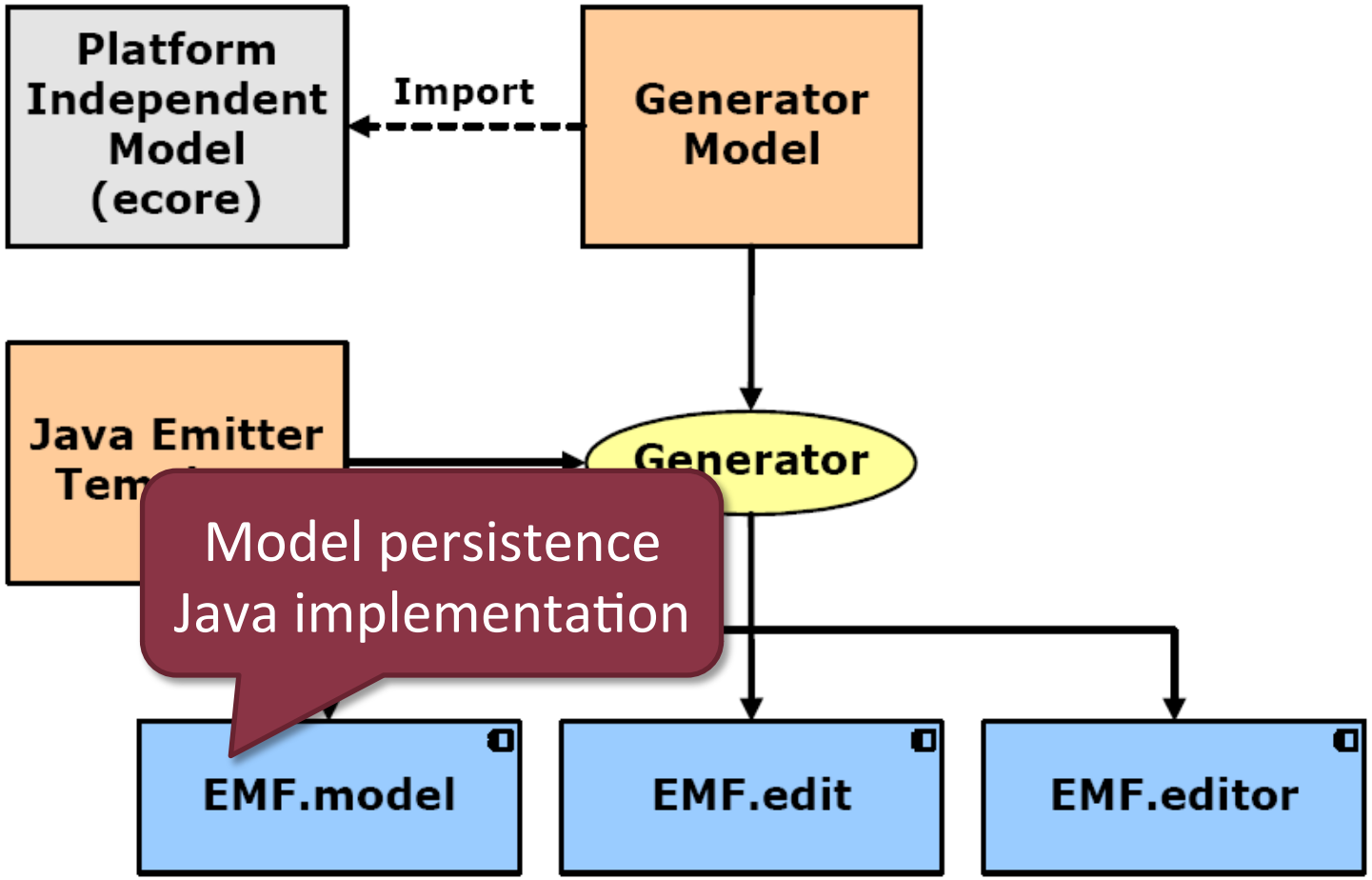
The EMF tooling



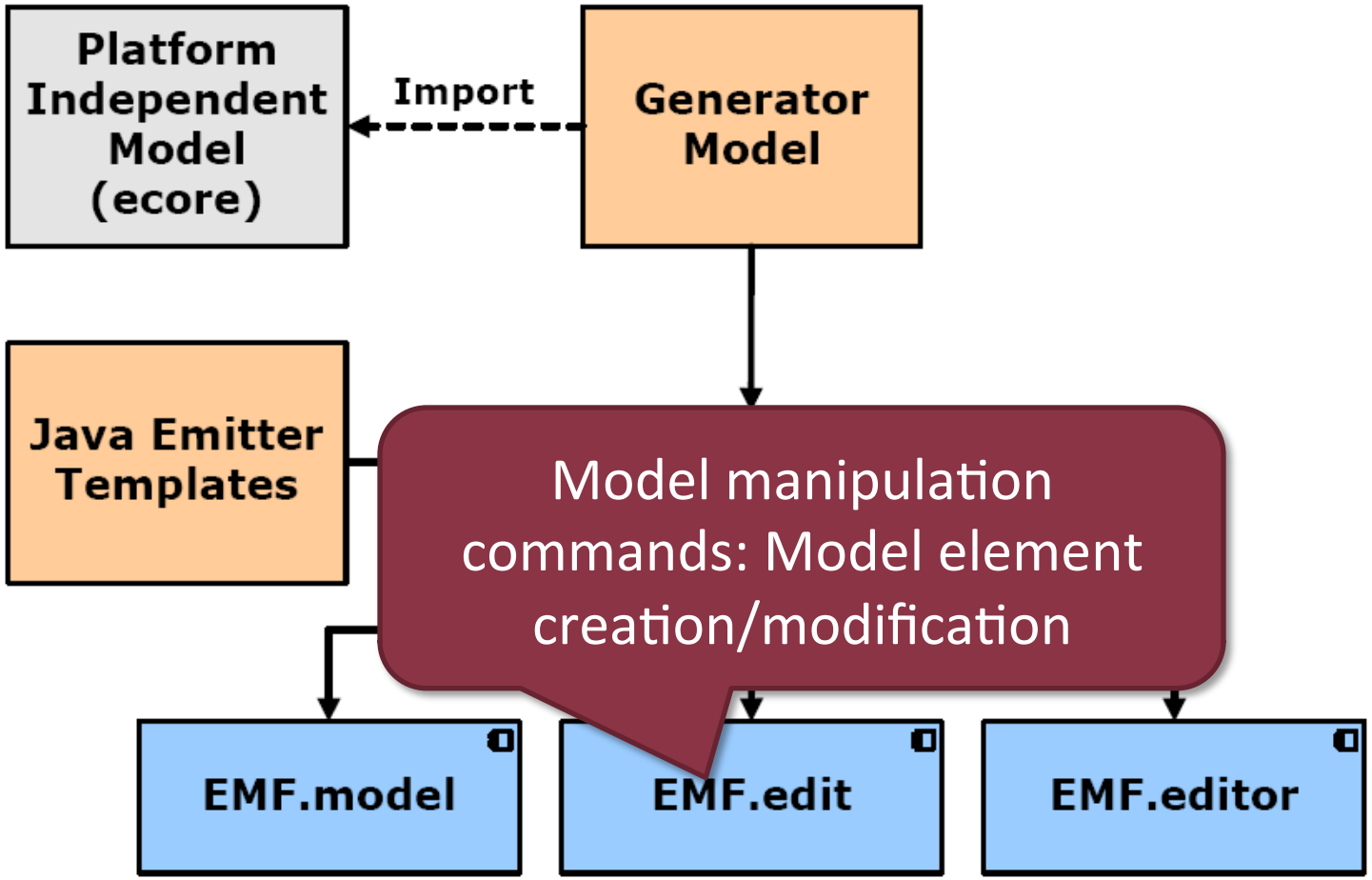
The EMF tooling



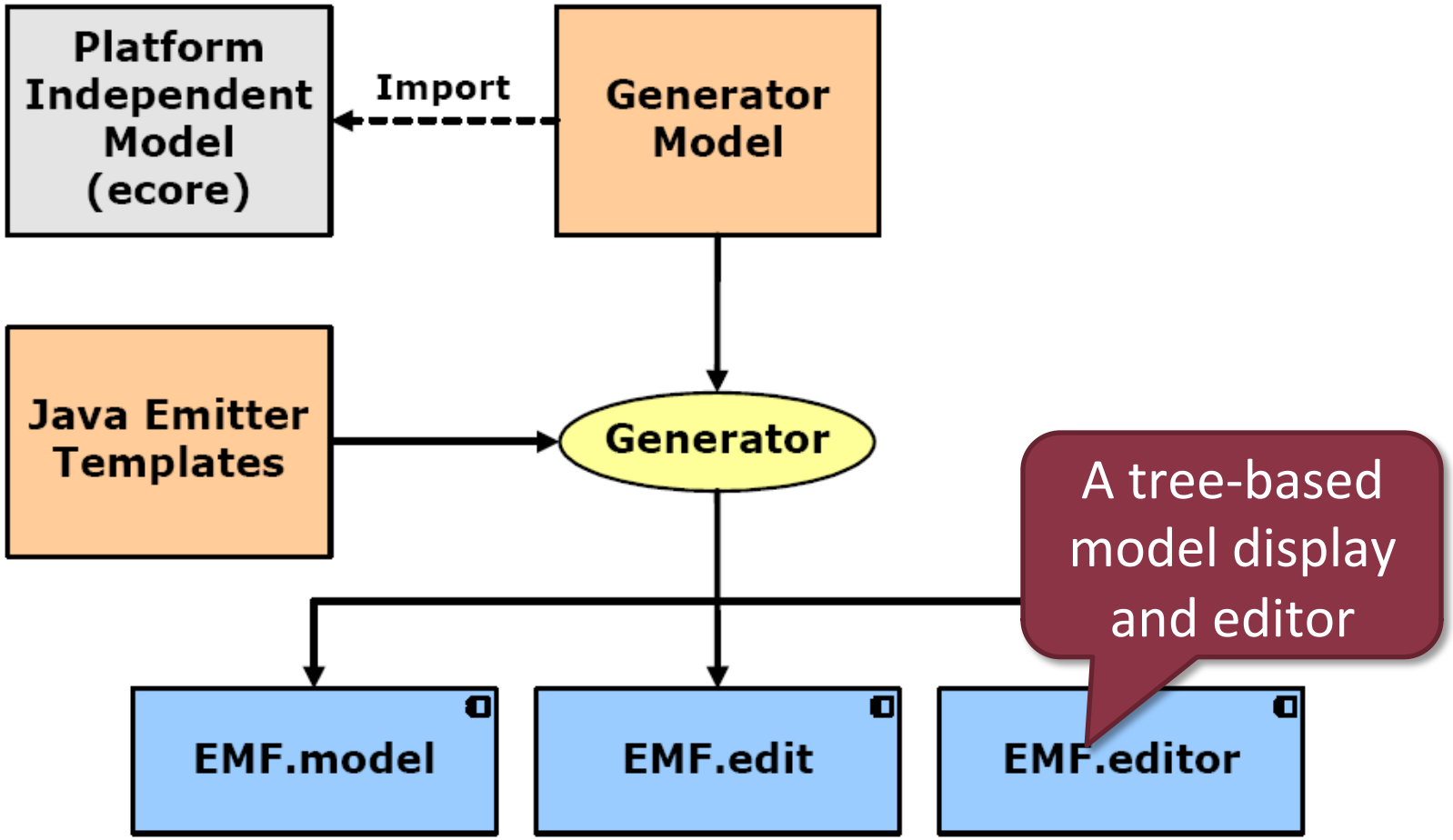
The EMF tooling



The EMF tooling



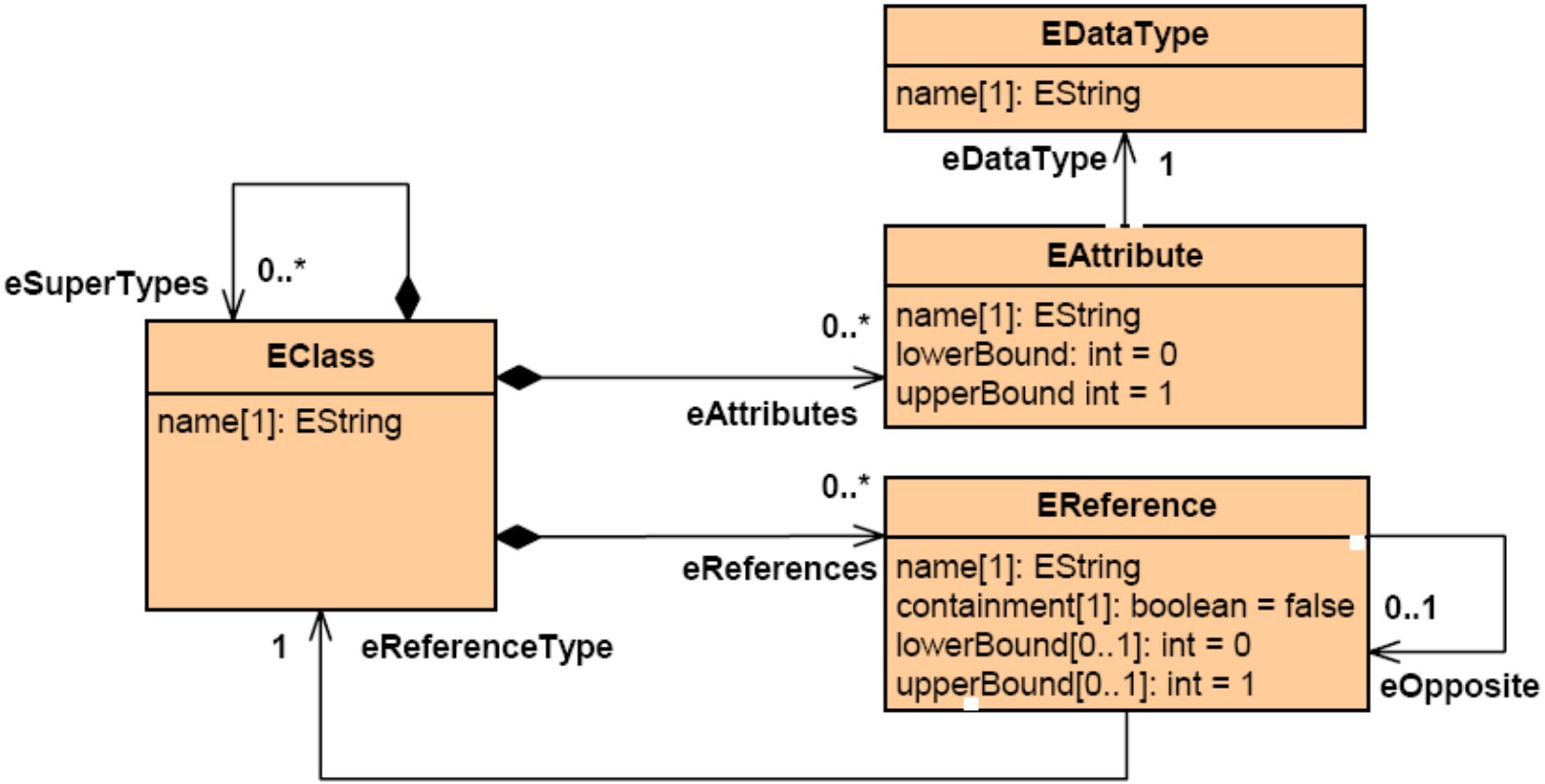
The EMF tooling



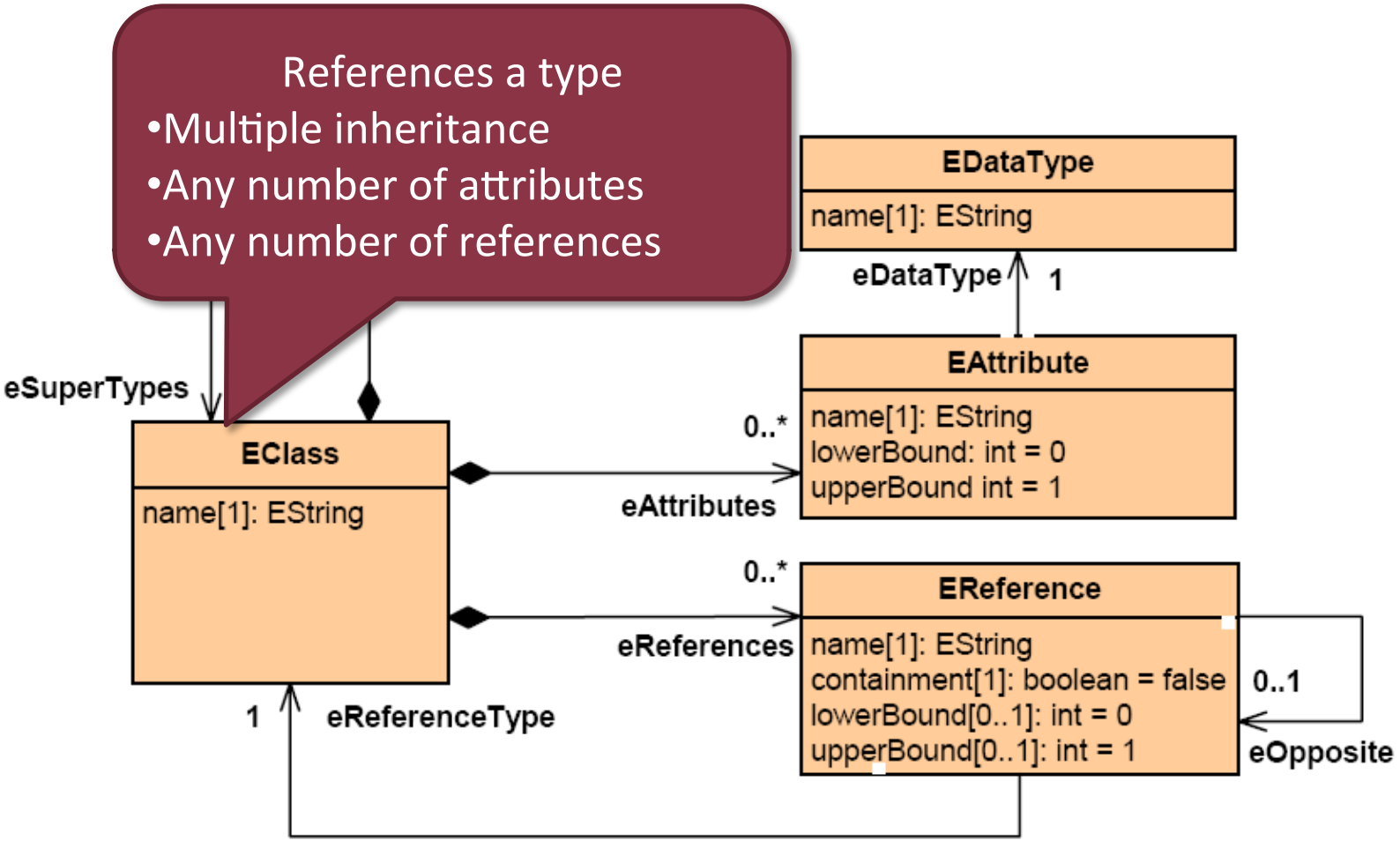
Ecore language

- Metamodeling language of EMF
  - Meta-language
- Platform independent metamodels
  - Additional models for platform-specific modeling (see genmodel)
- Defines structural information

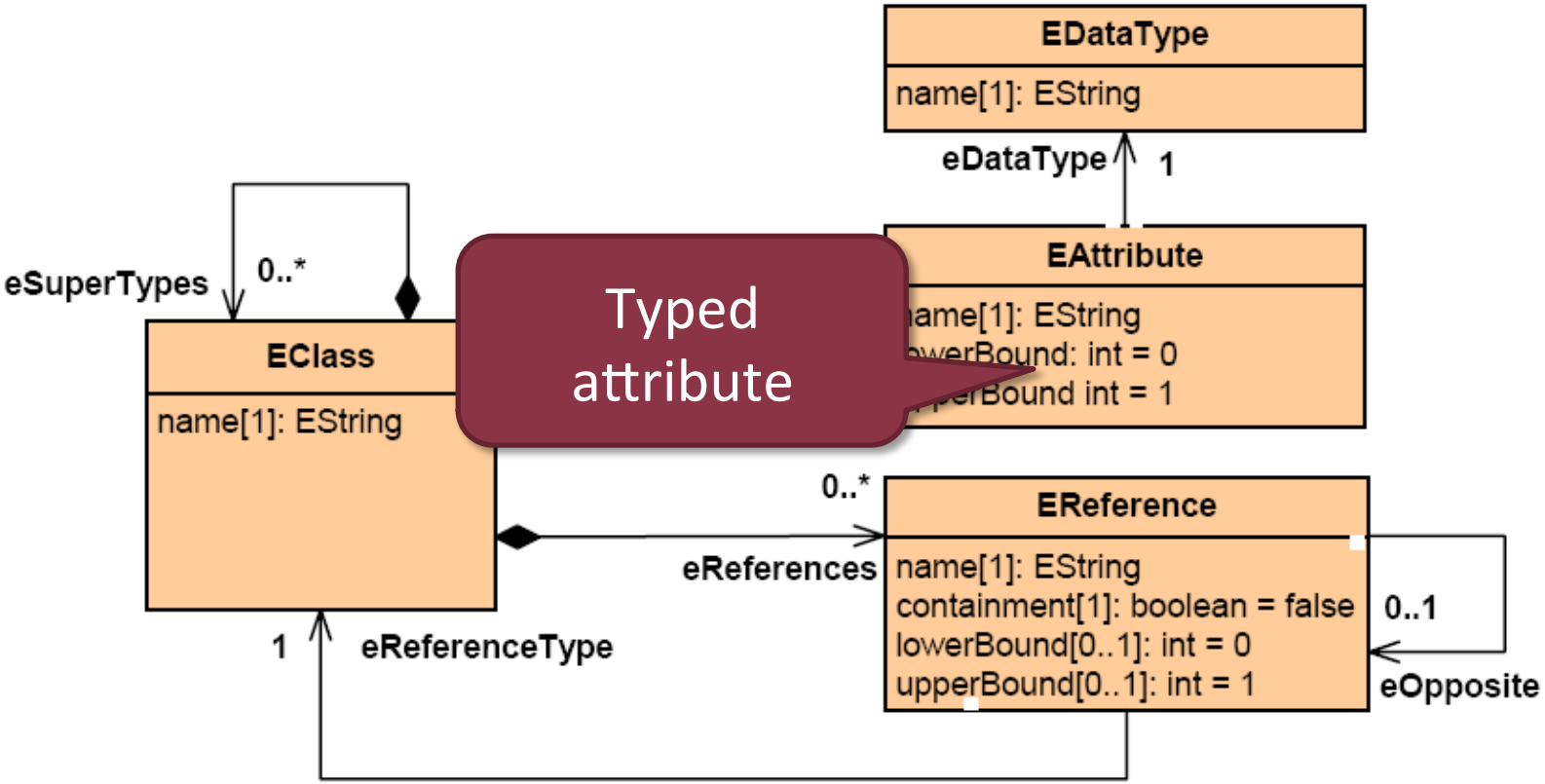
Ecore – Most important concepts



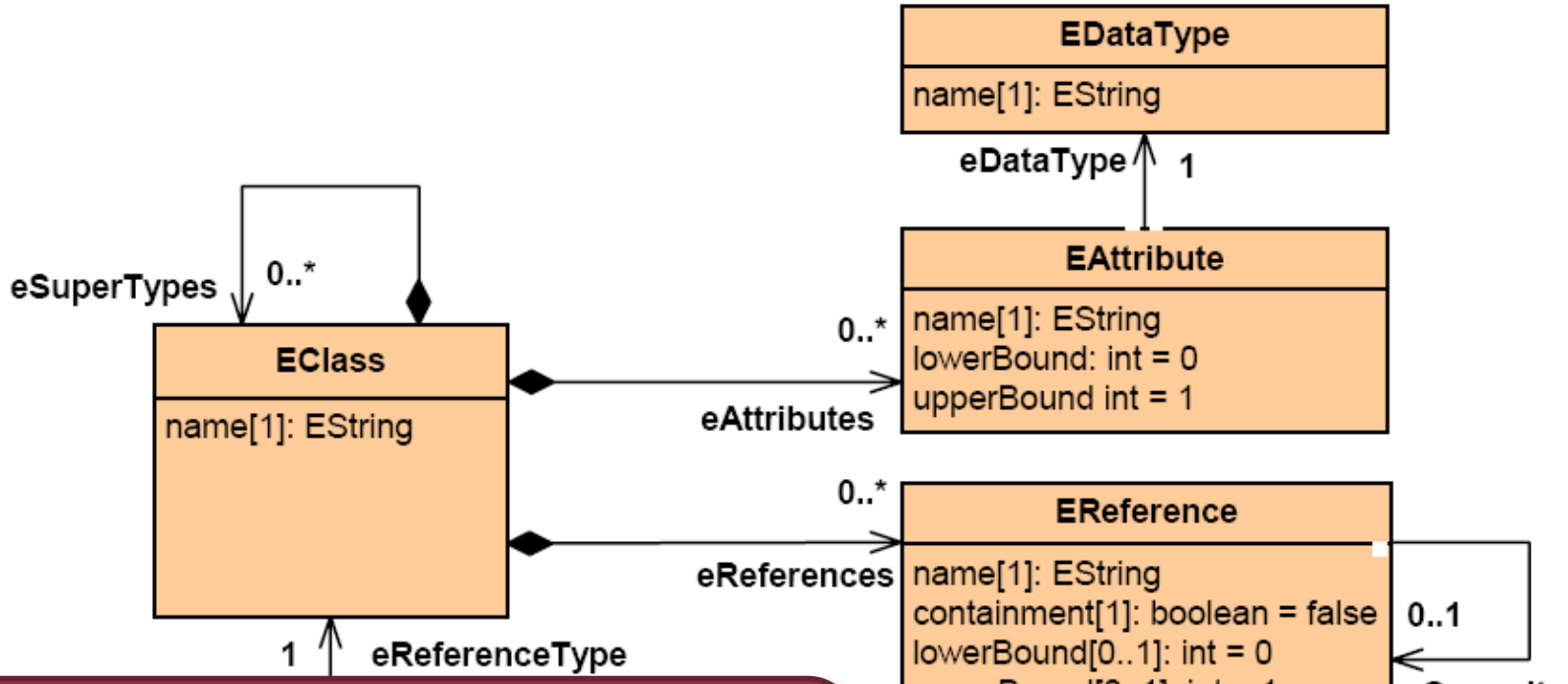
Ecore – Most important concepts



Ecore – Most important concepts



Ecore – Most important concepts

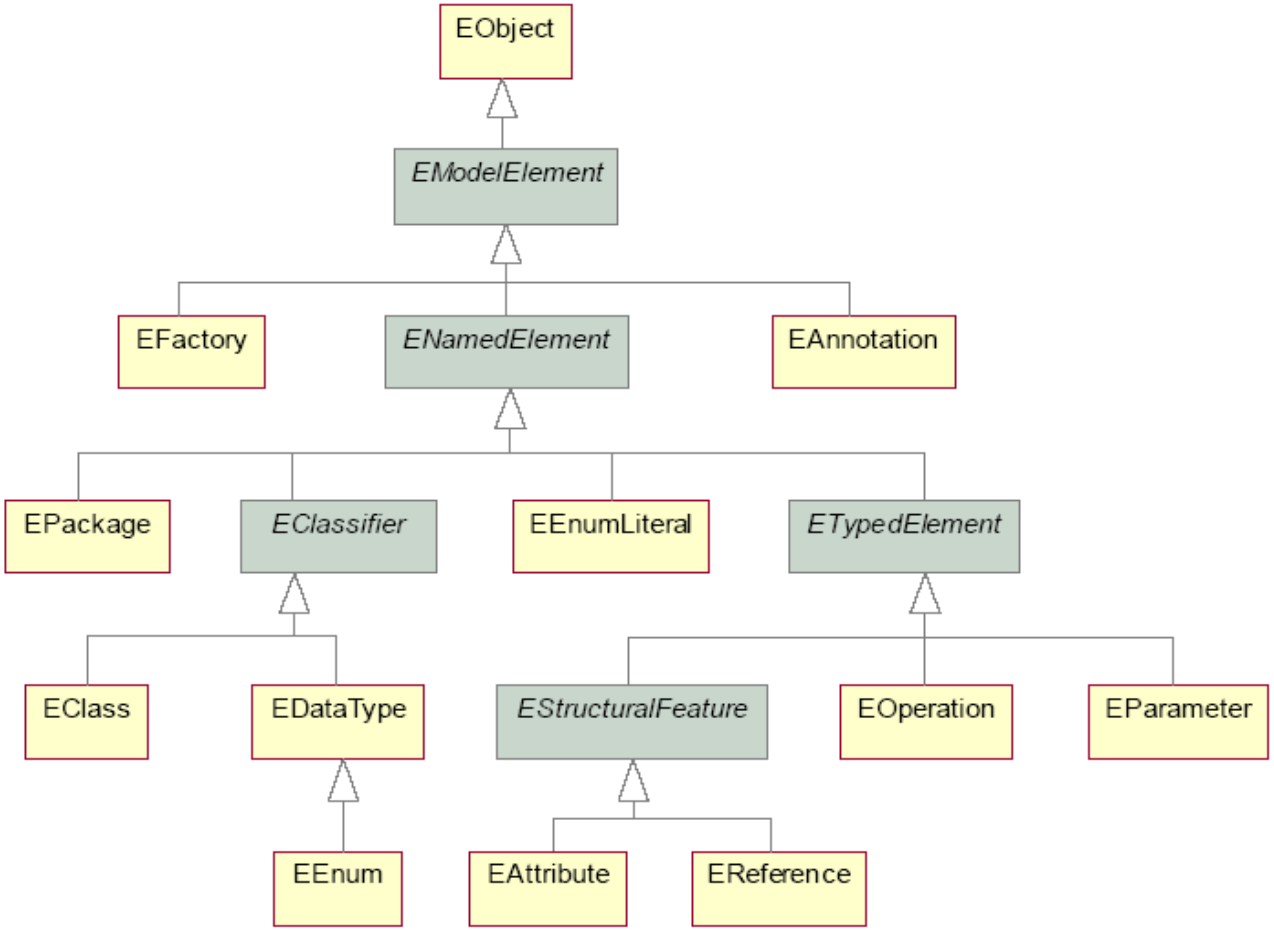


Unidirectonal, binary connection
- Optionally inverse reference
- Defines referenced type

Ecore – Most important concepts

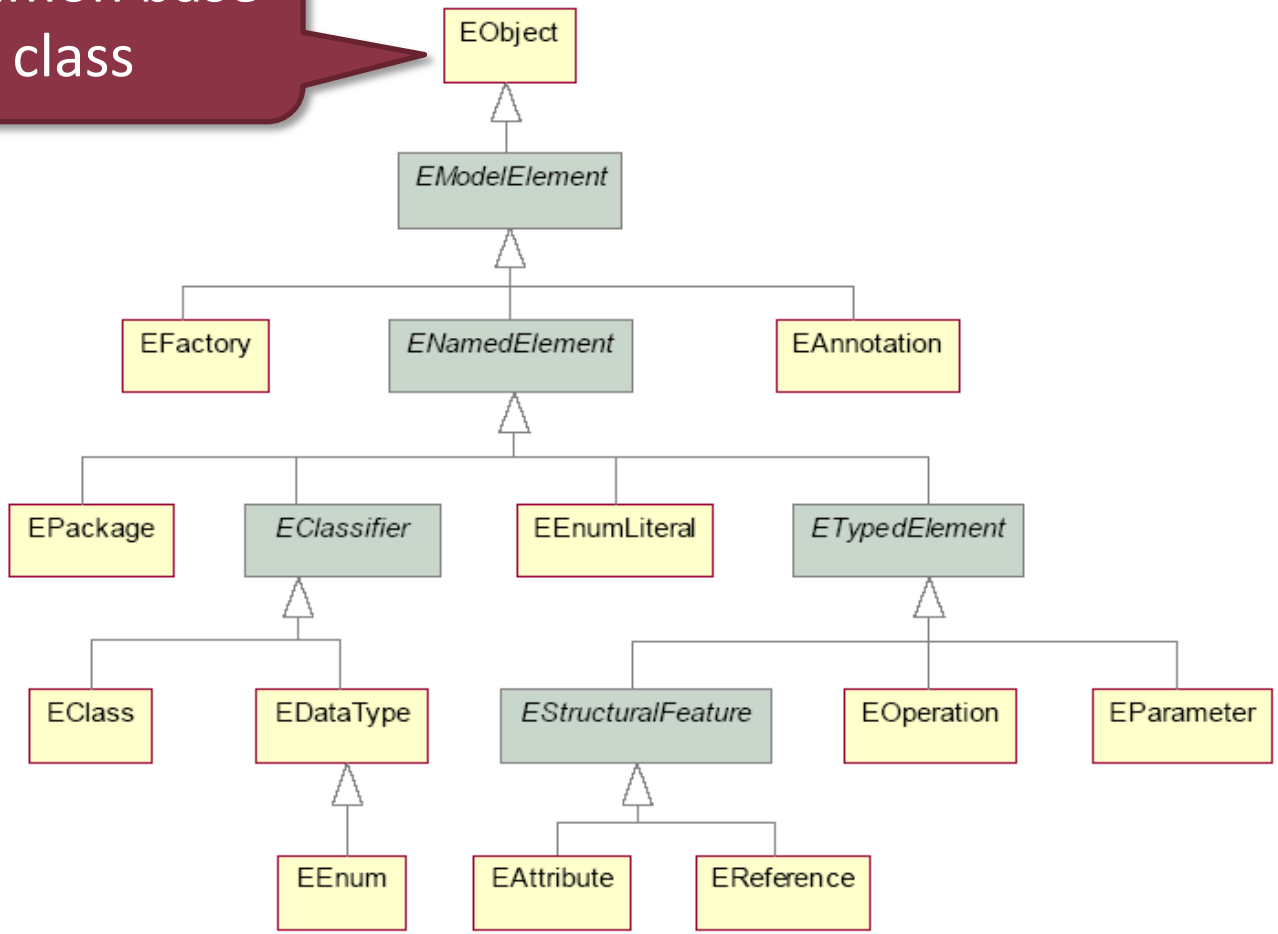
- EPackage
  - Contains and manages a set of classes
  - Compile/code generation time
    - Builds together
  - In runtime
    - Are registered together
    - Provides API for
      - Factory methods
      - Reflective access

Full Ecore hierarchy

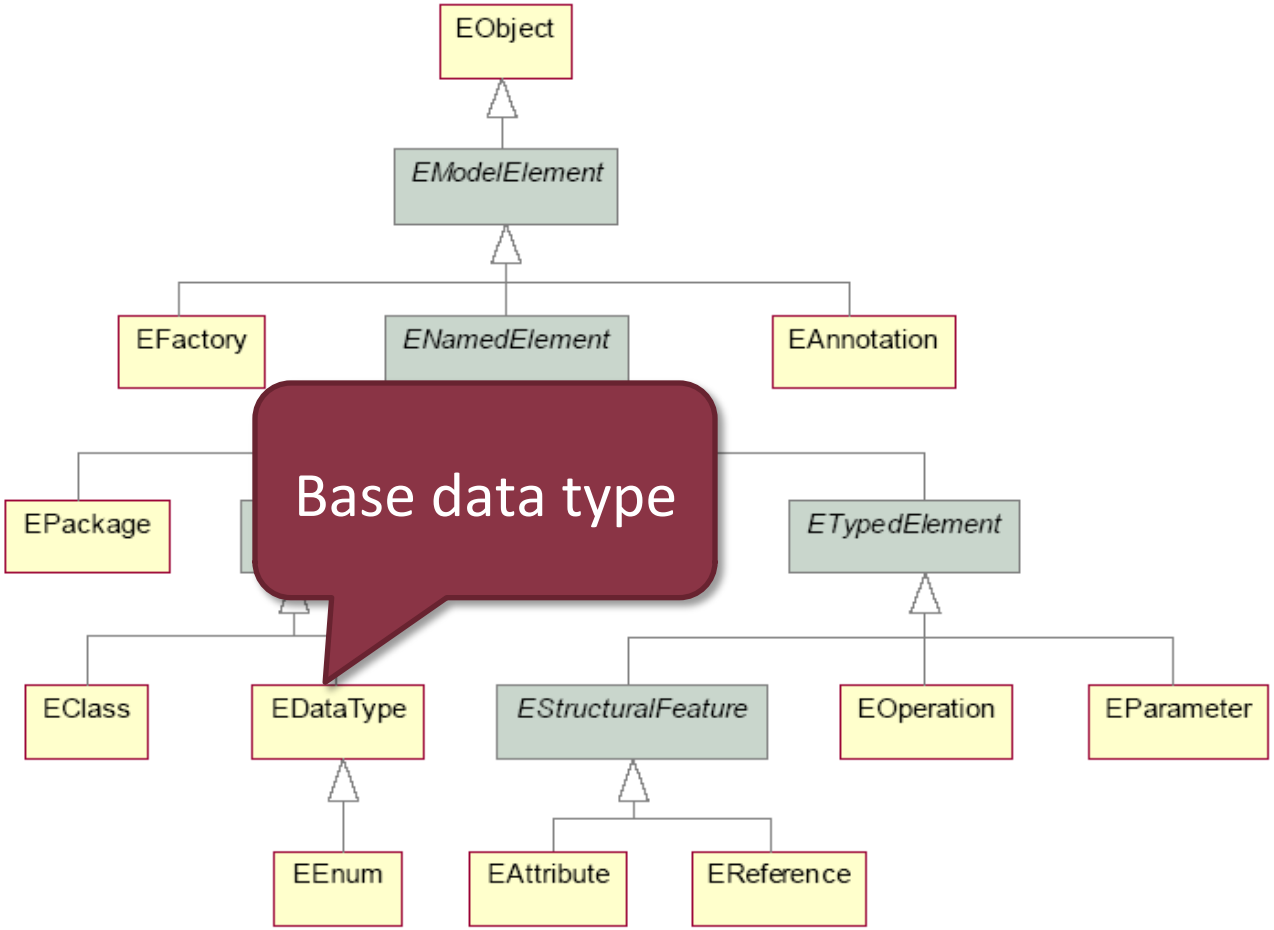


Full Ecore hierarchy

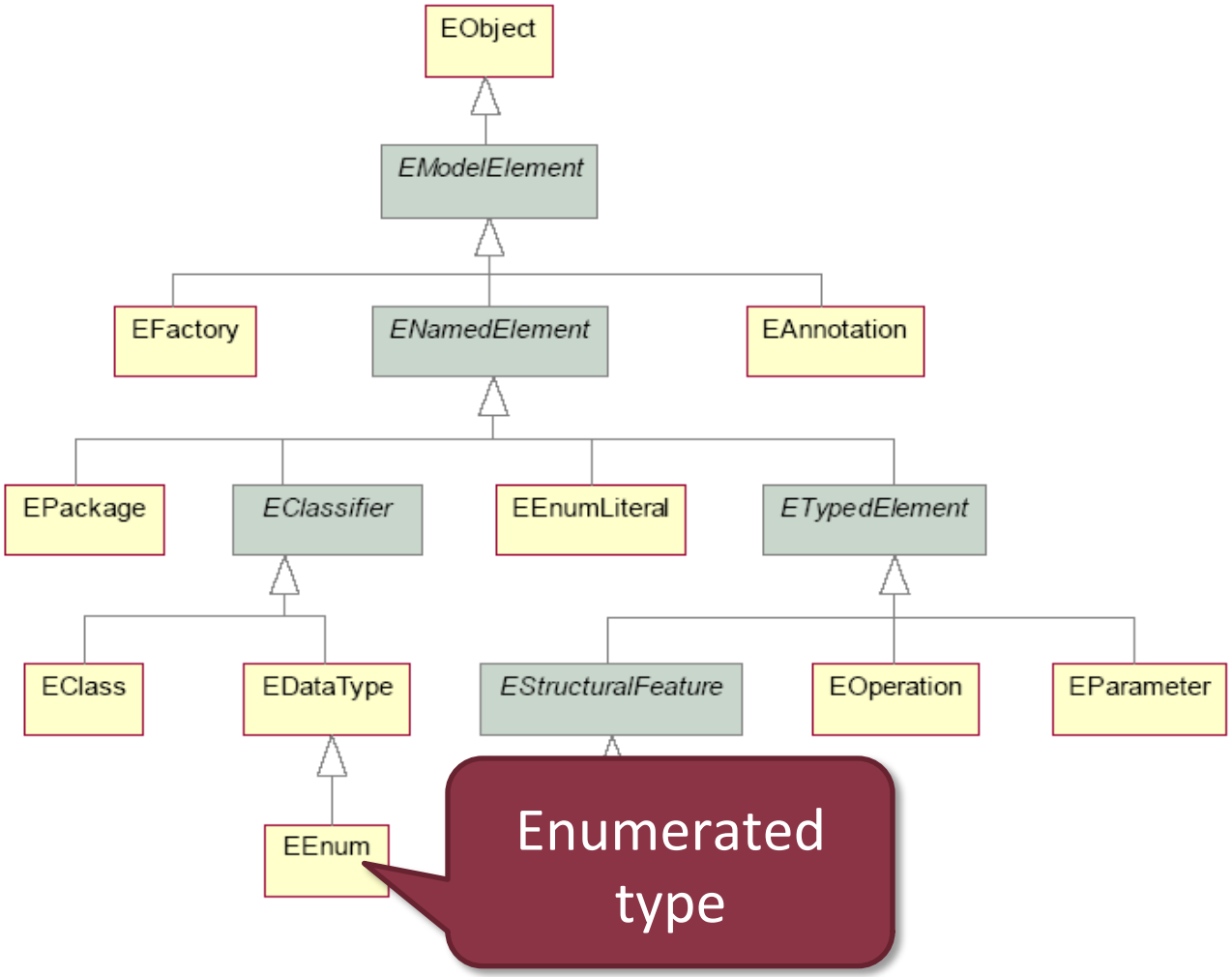
Common base class



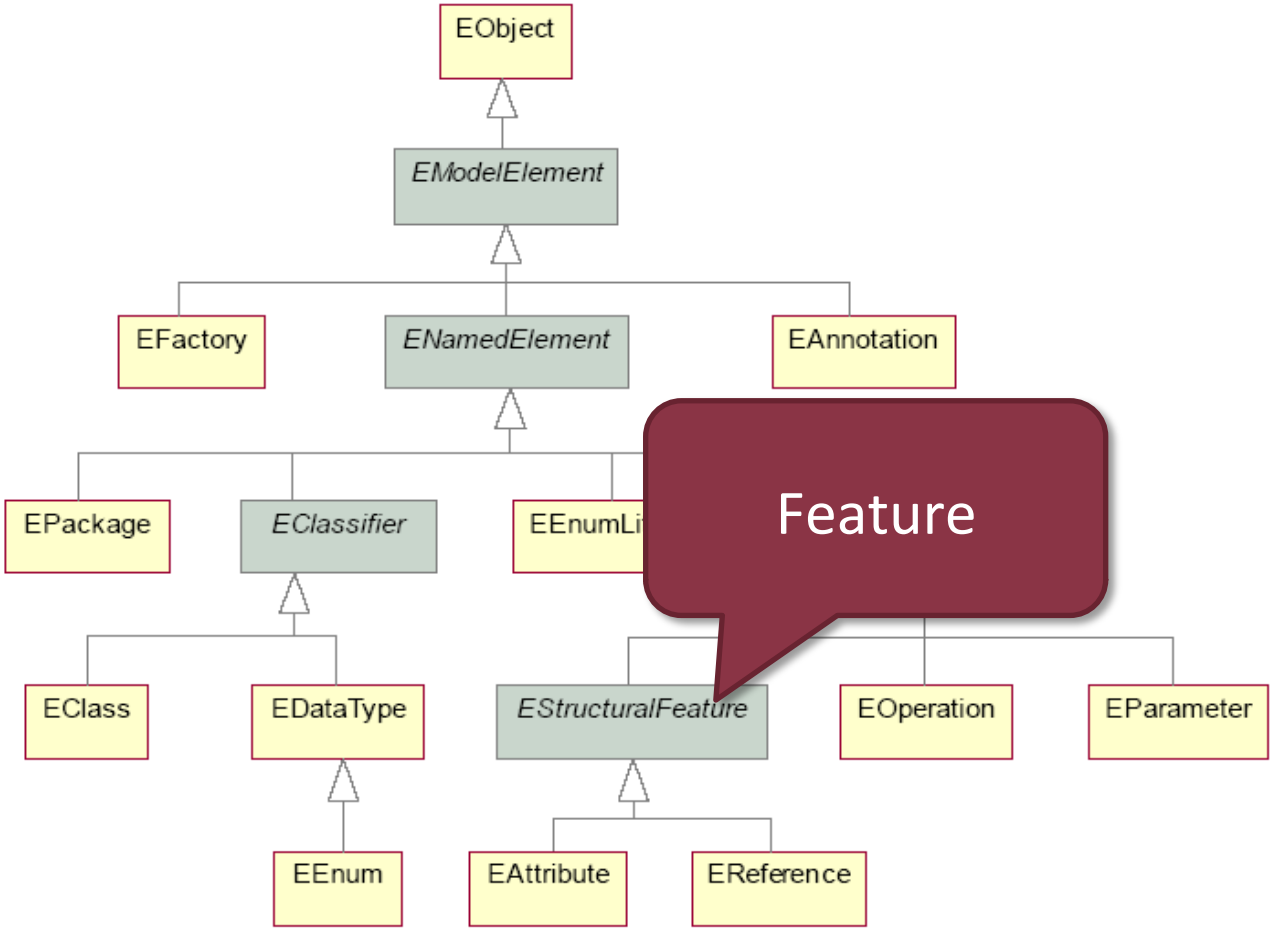
Full Ecore hierarchy



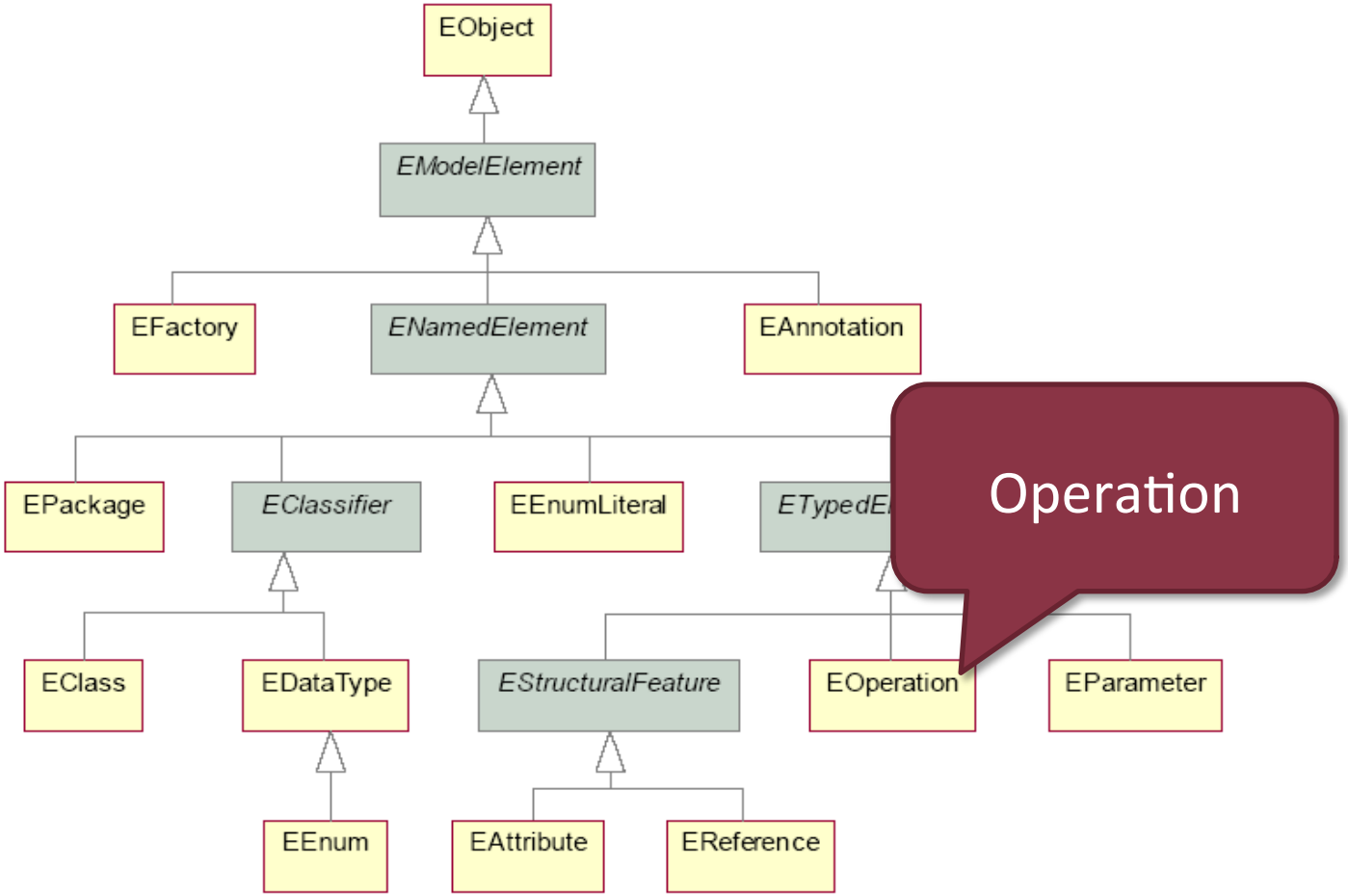
Full Ecore hierarchy



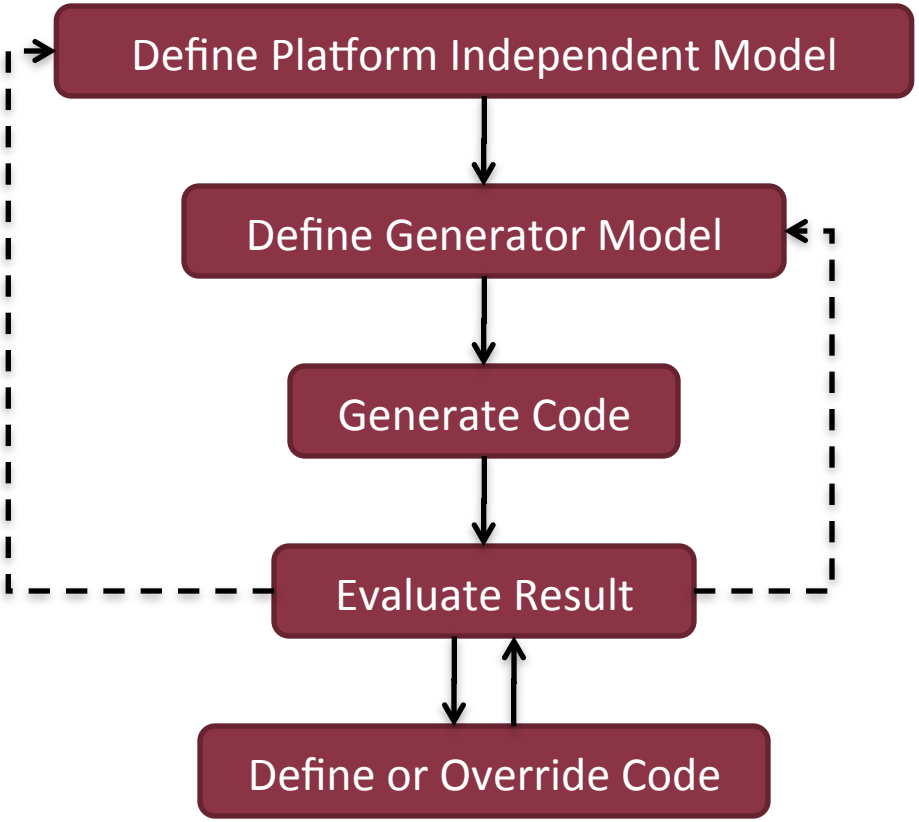
Full Ecore hierarchy



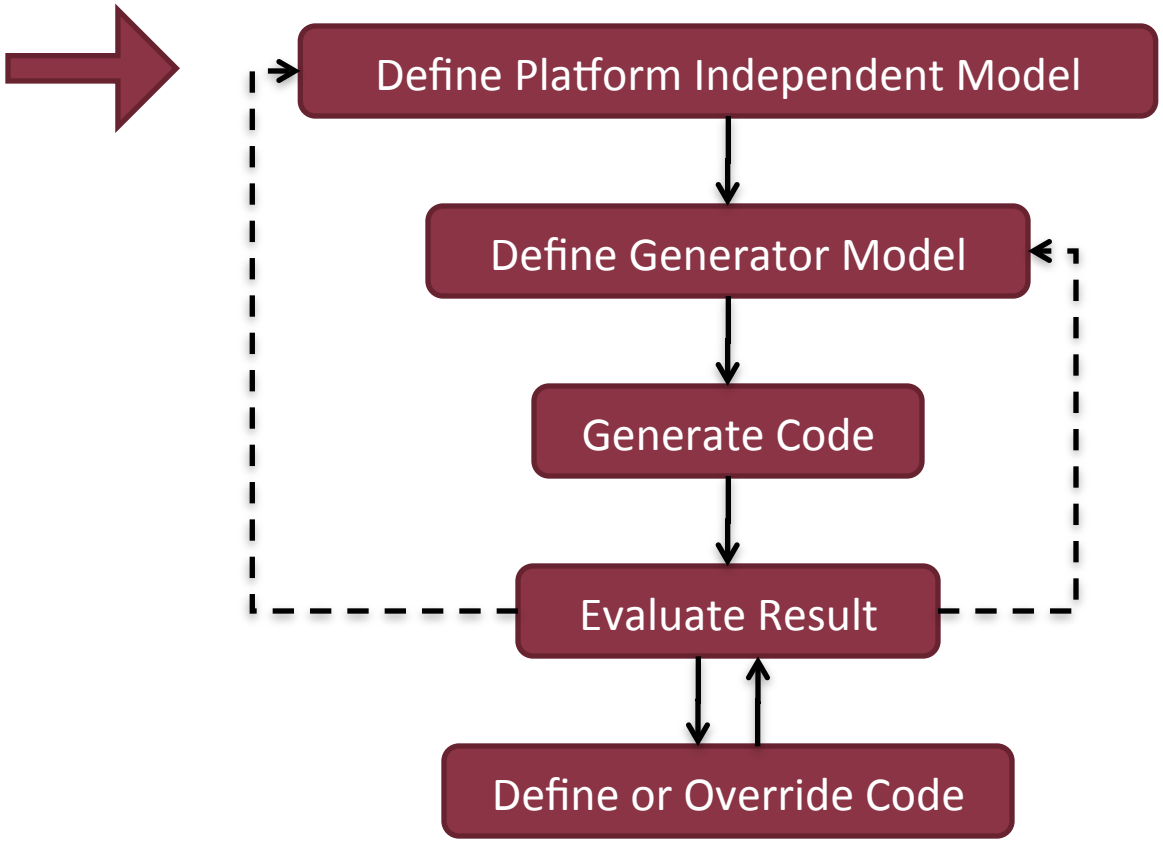
Full Ecore hierarchy



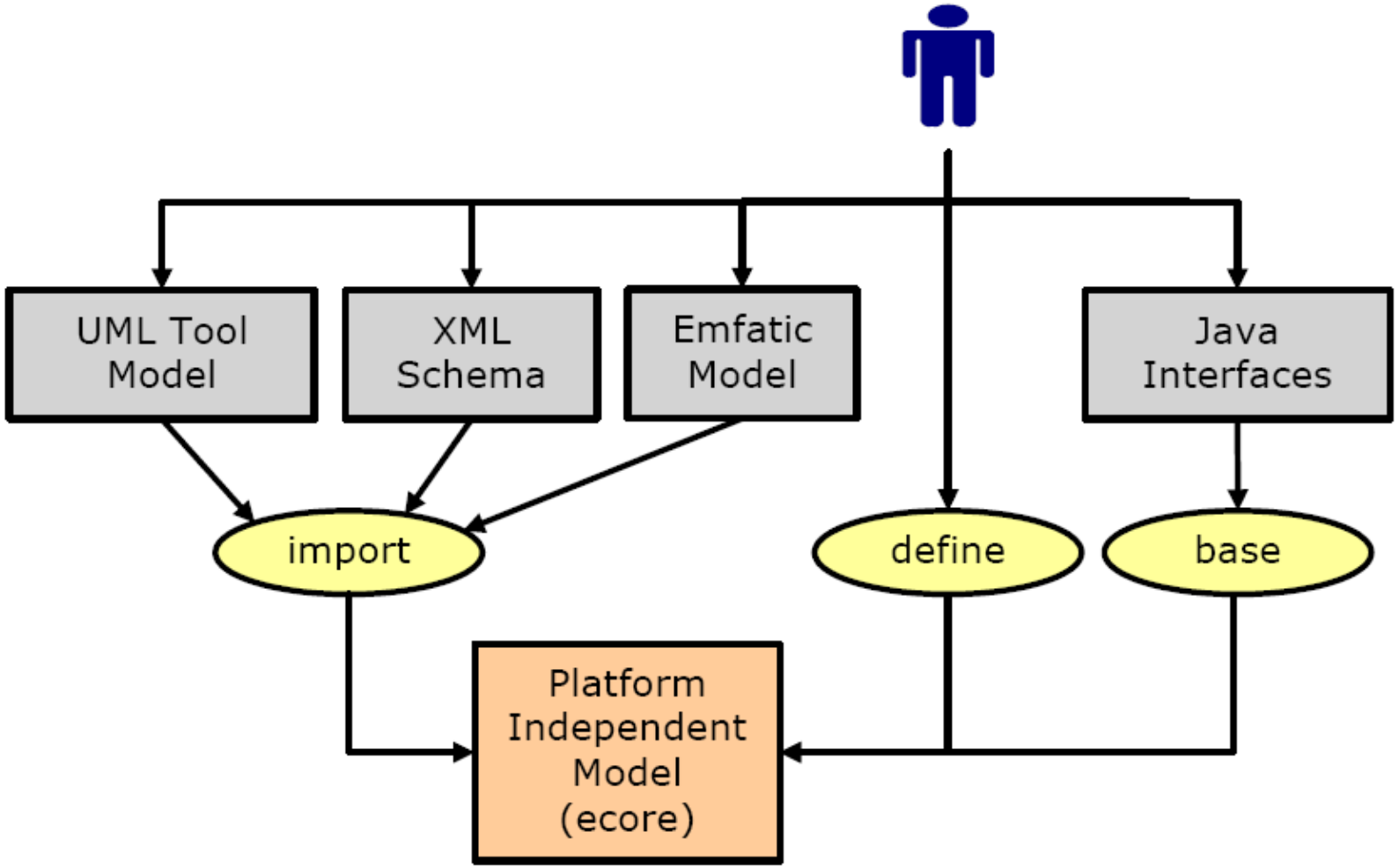
Using EMF


Define Platform Independent Model
Define Generator Model
Generate Code
Evaluate Result
Define or Override Code

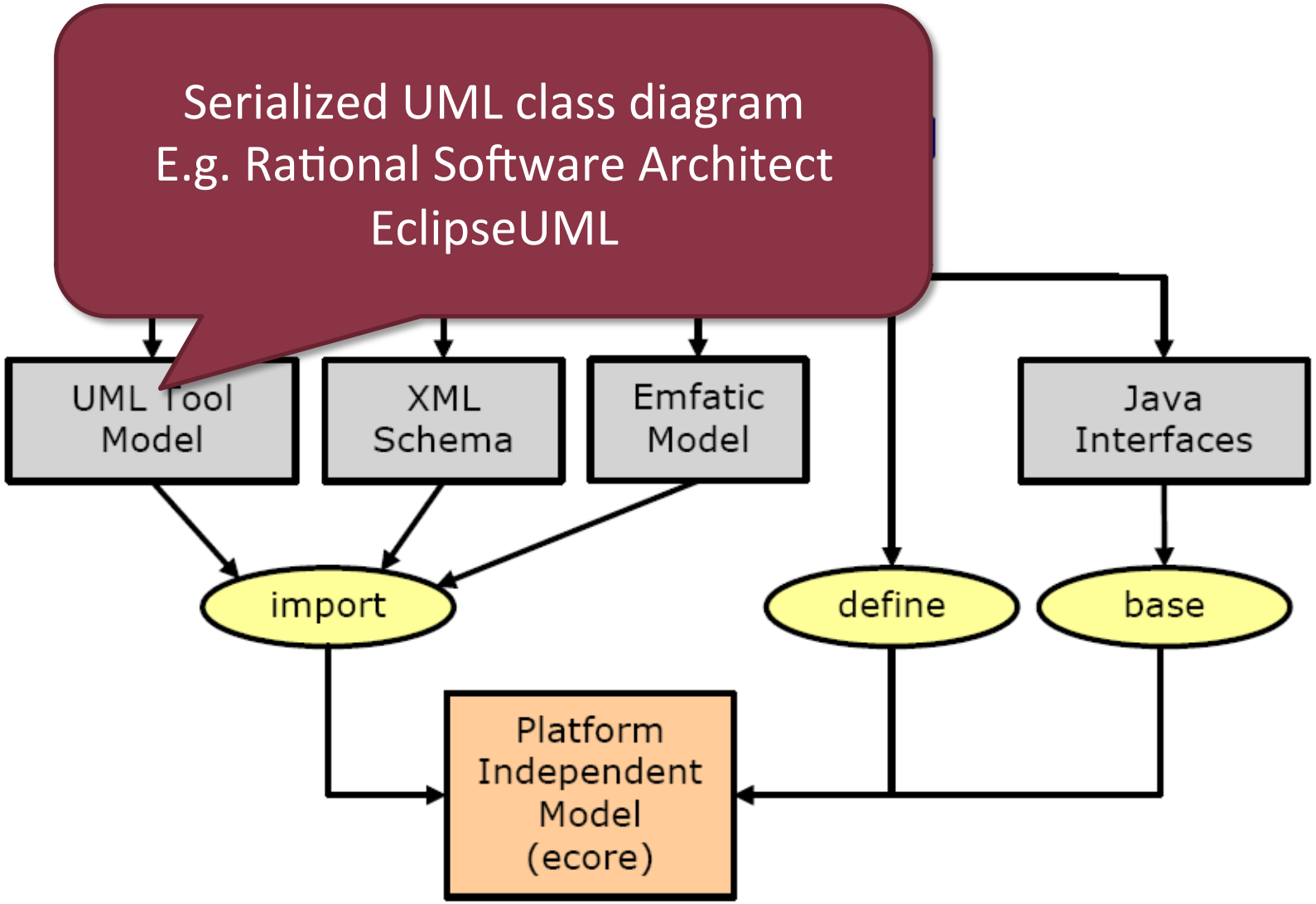
Using EMF


Define Platform Independent Model
Define Generator Model
Generate Code
Evaluate Result
Define or Override Code

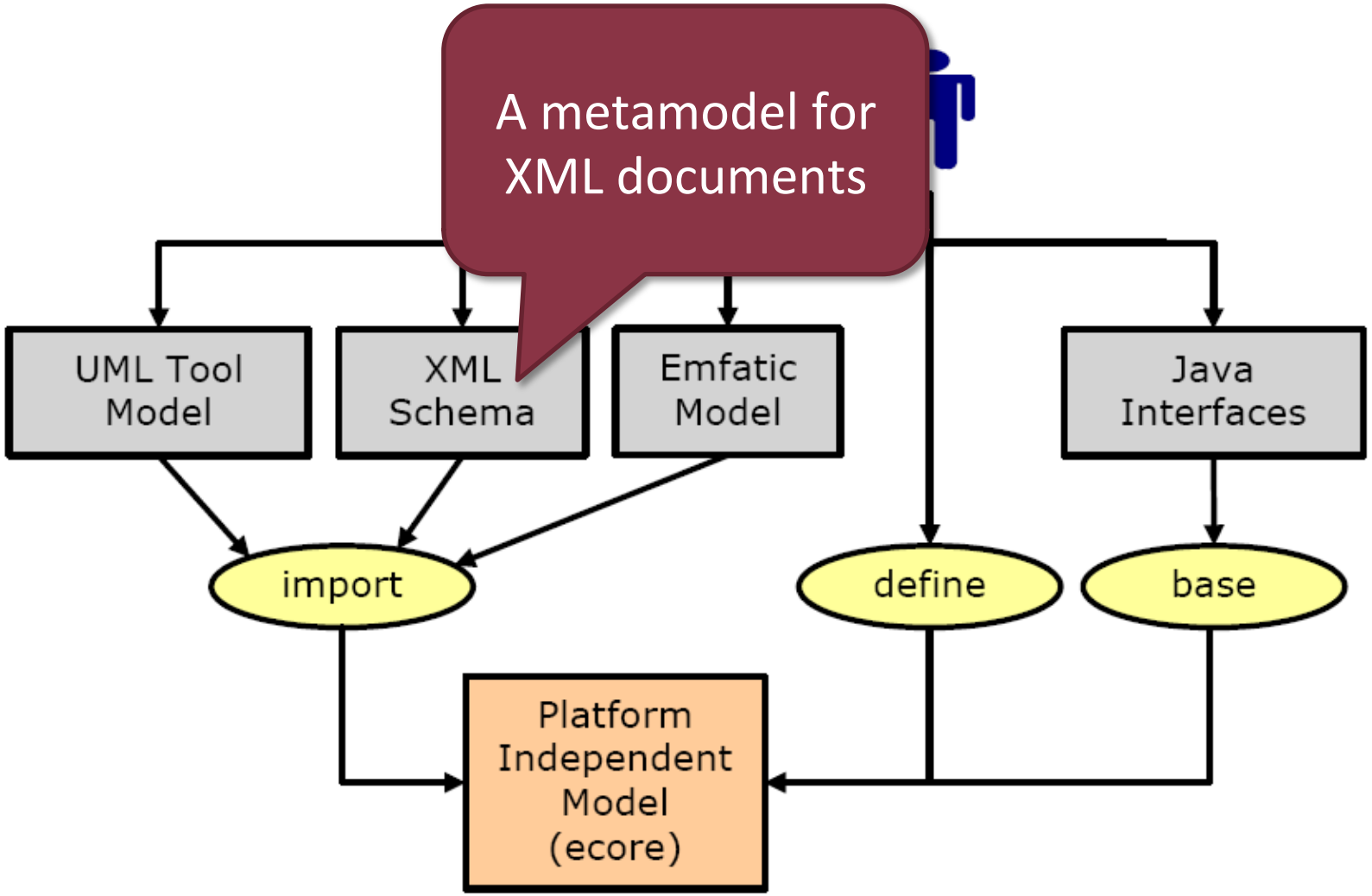
Defining Ecore Models



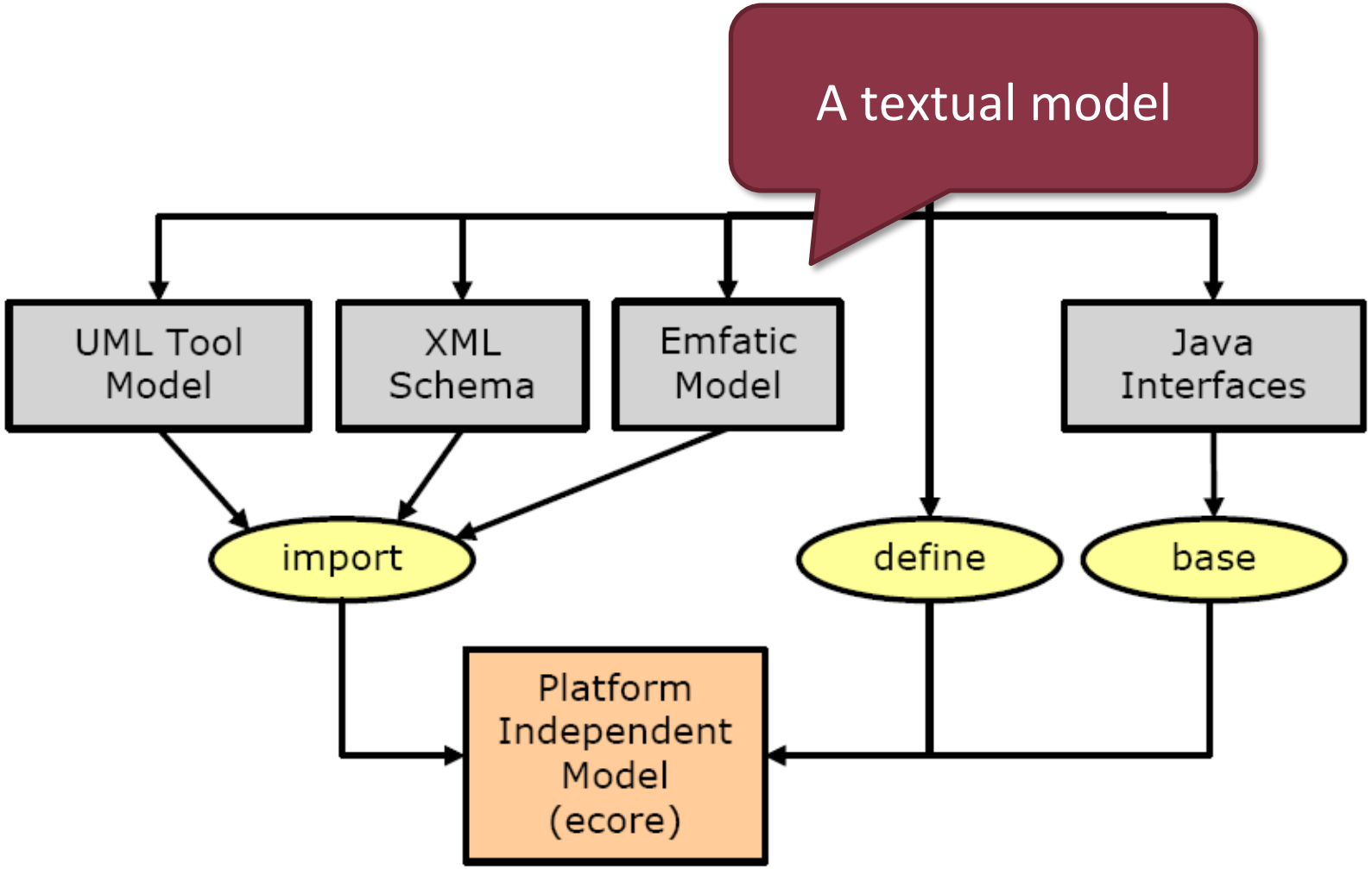
Defining Ecore Models



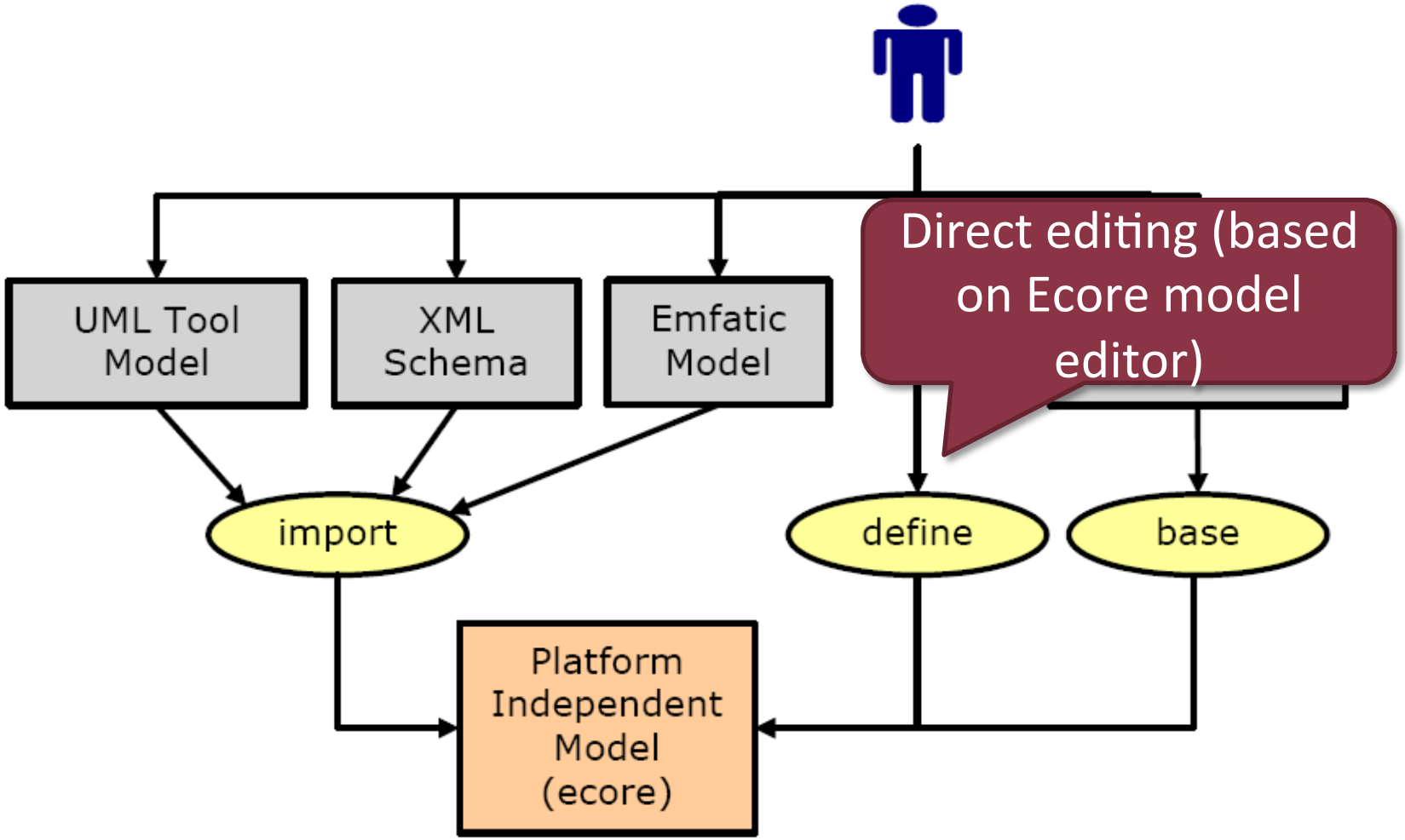
Defining Ecore Models



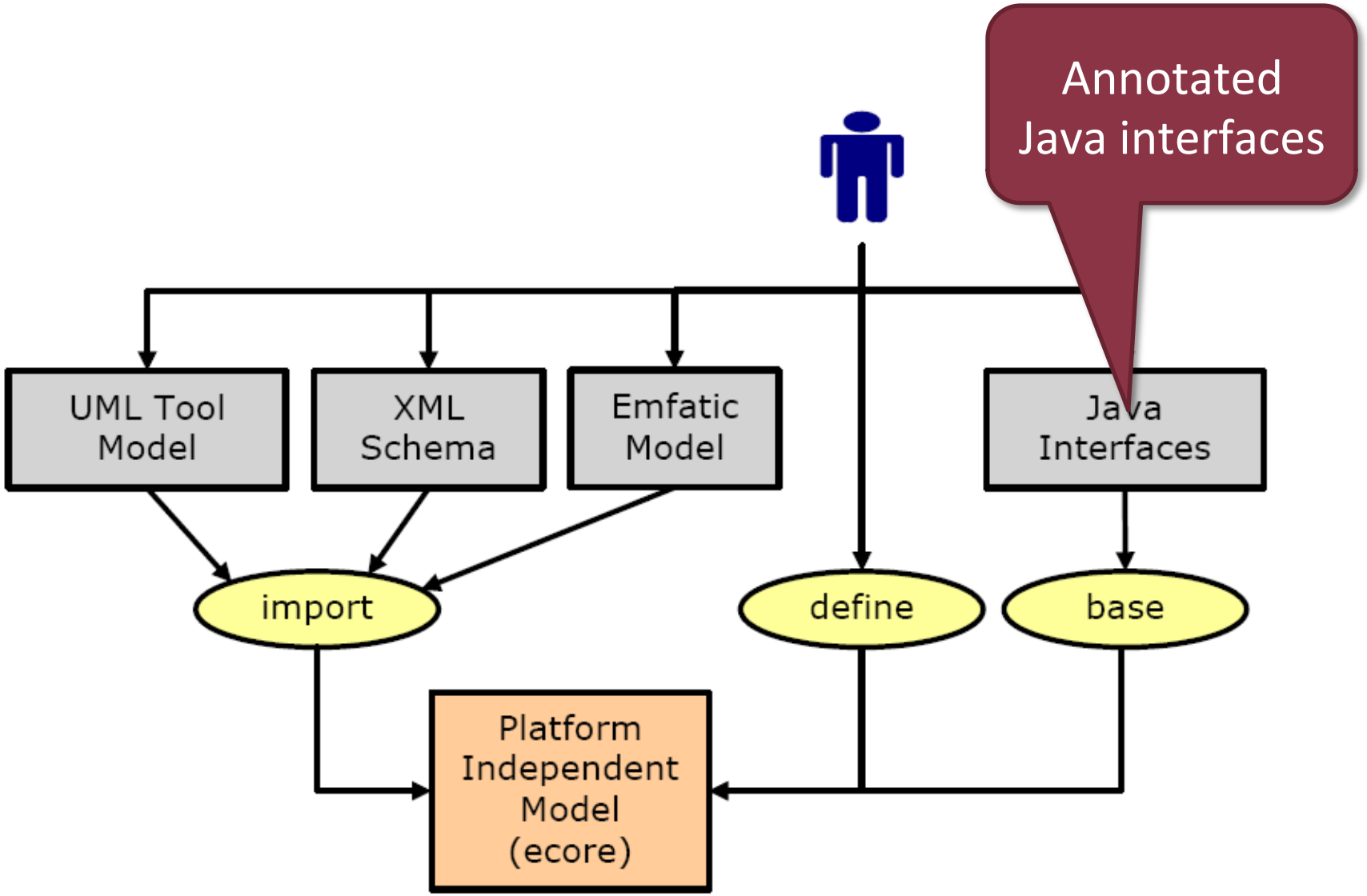
Defining Ecore Models



Defining Ecore Models

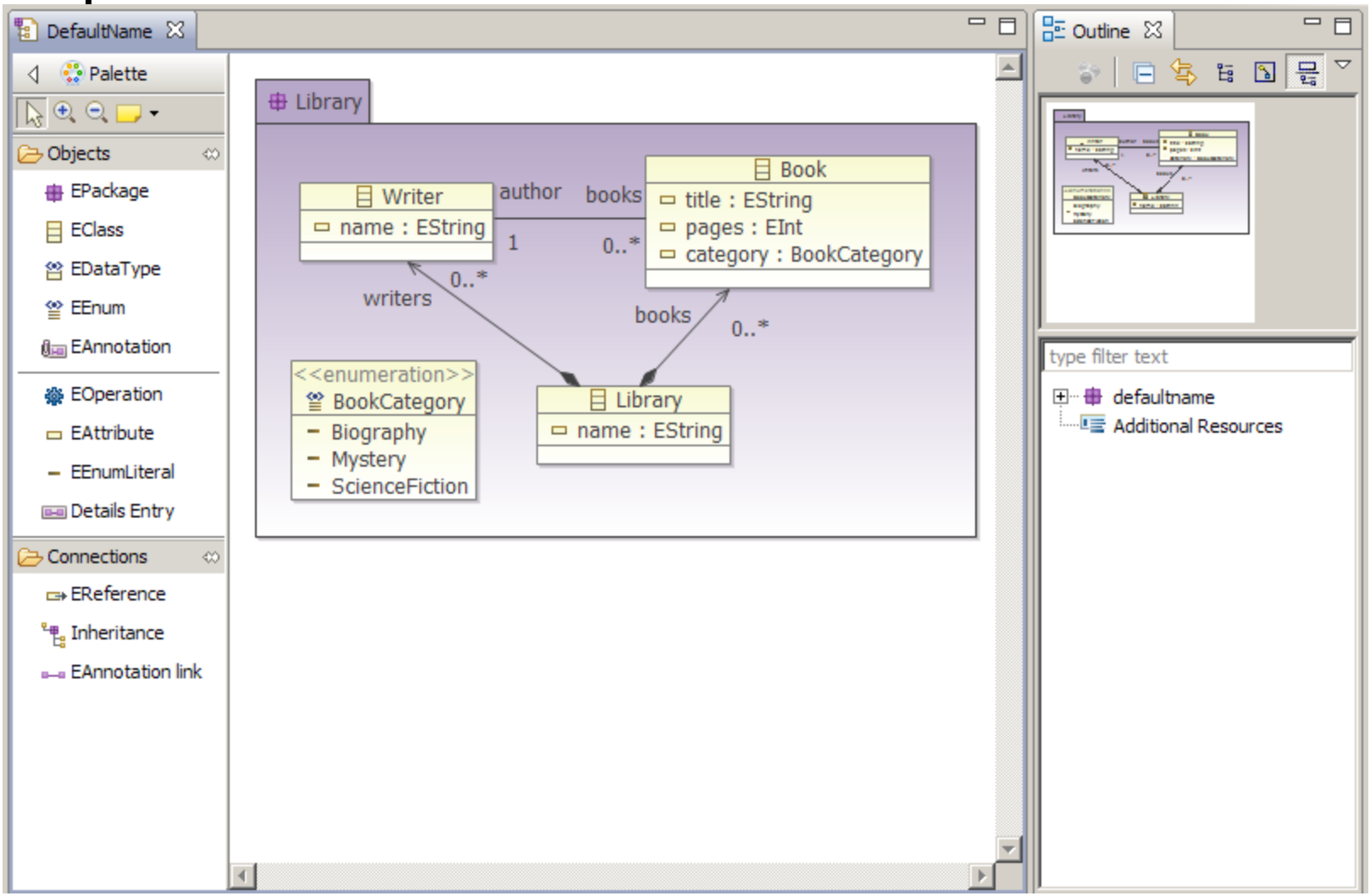


Defining Ecore Models

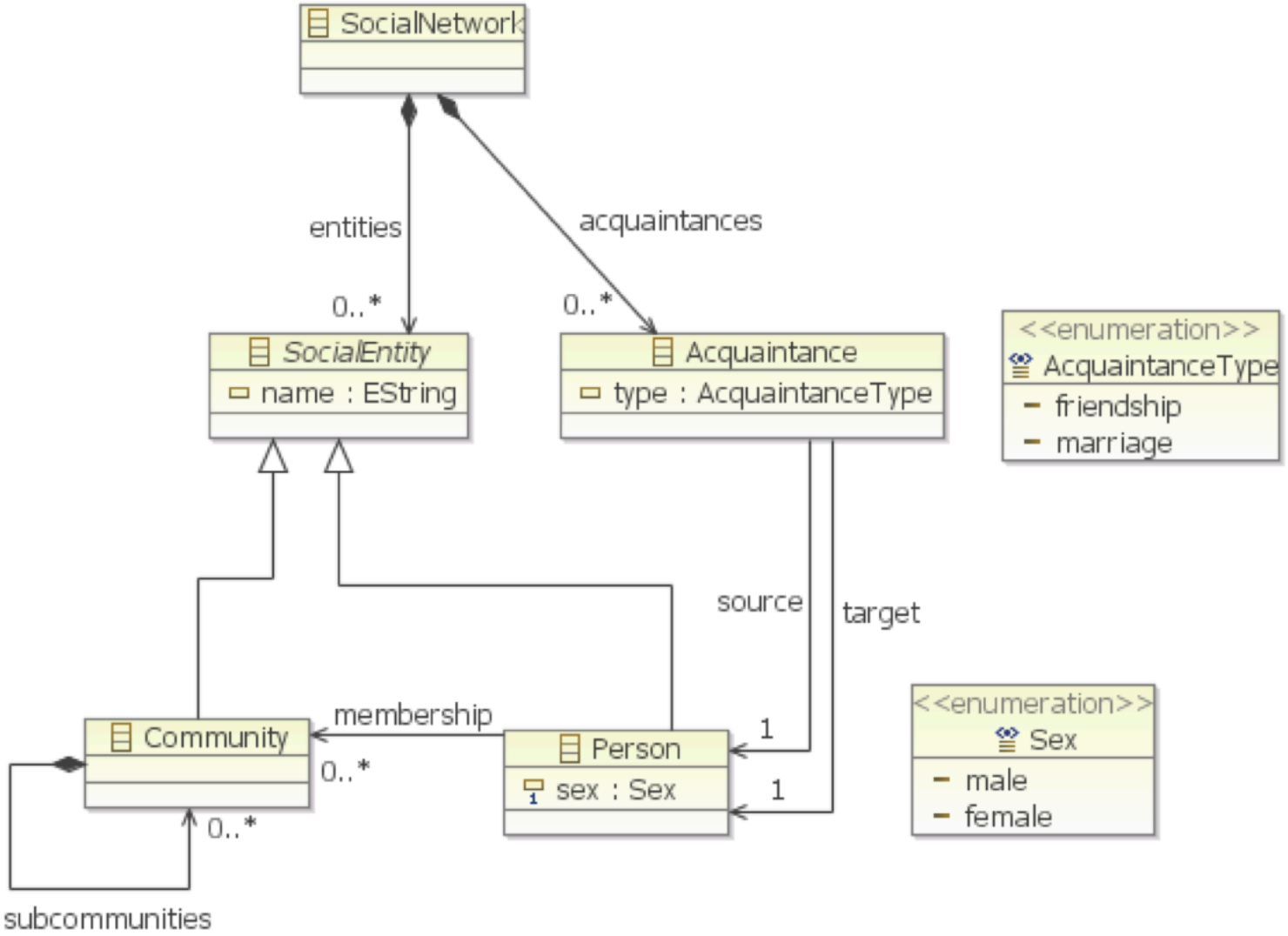


Ecore Tools (1.x): Ecore Diagram Editor

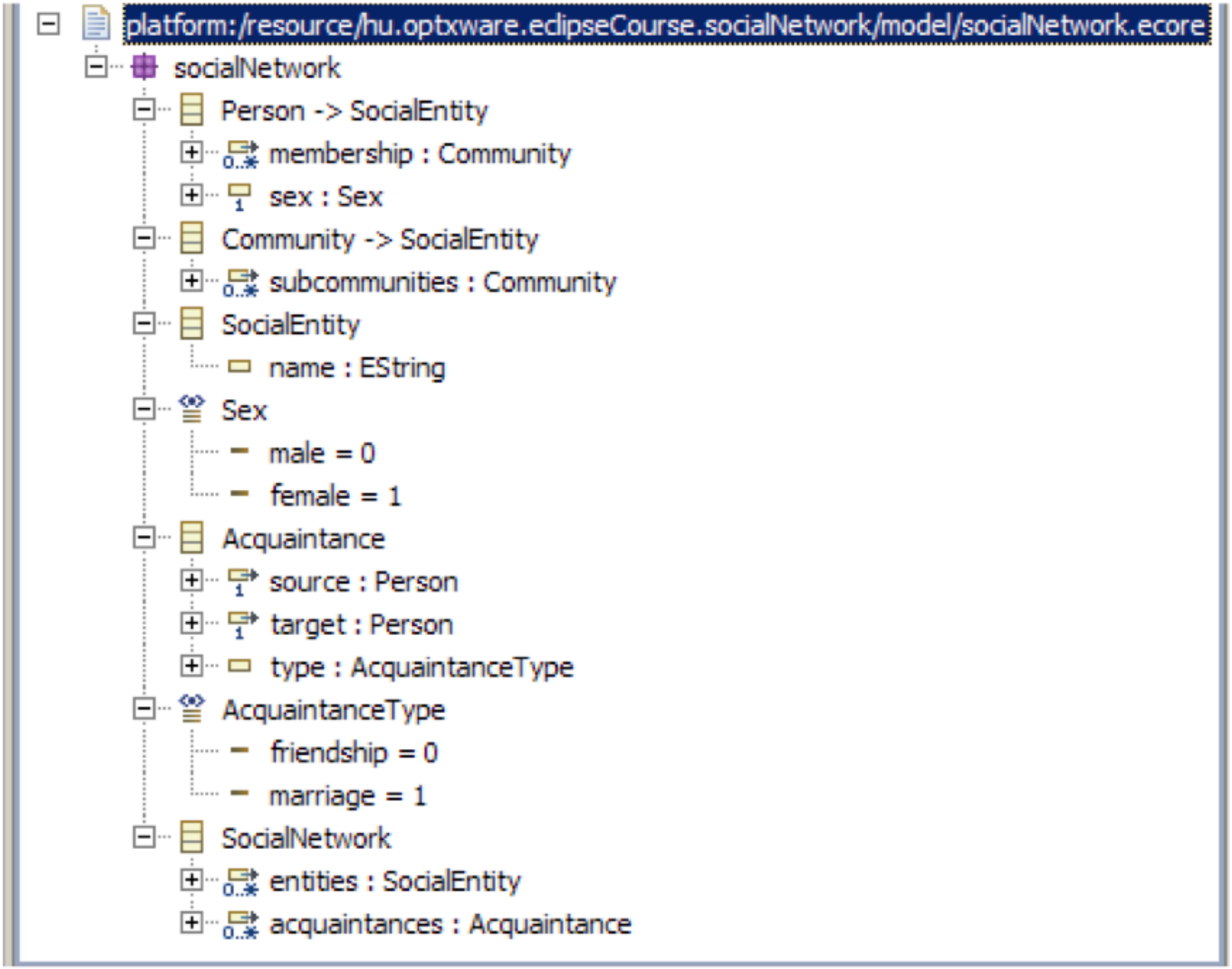
- Graphical DSL for EMF metamodels



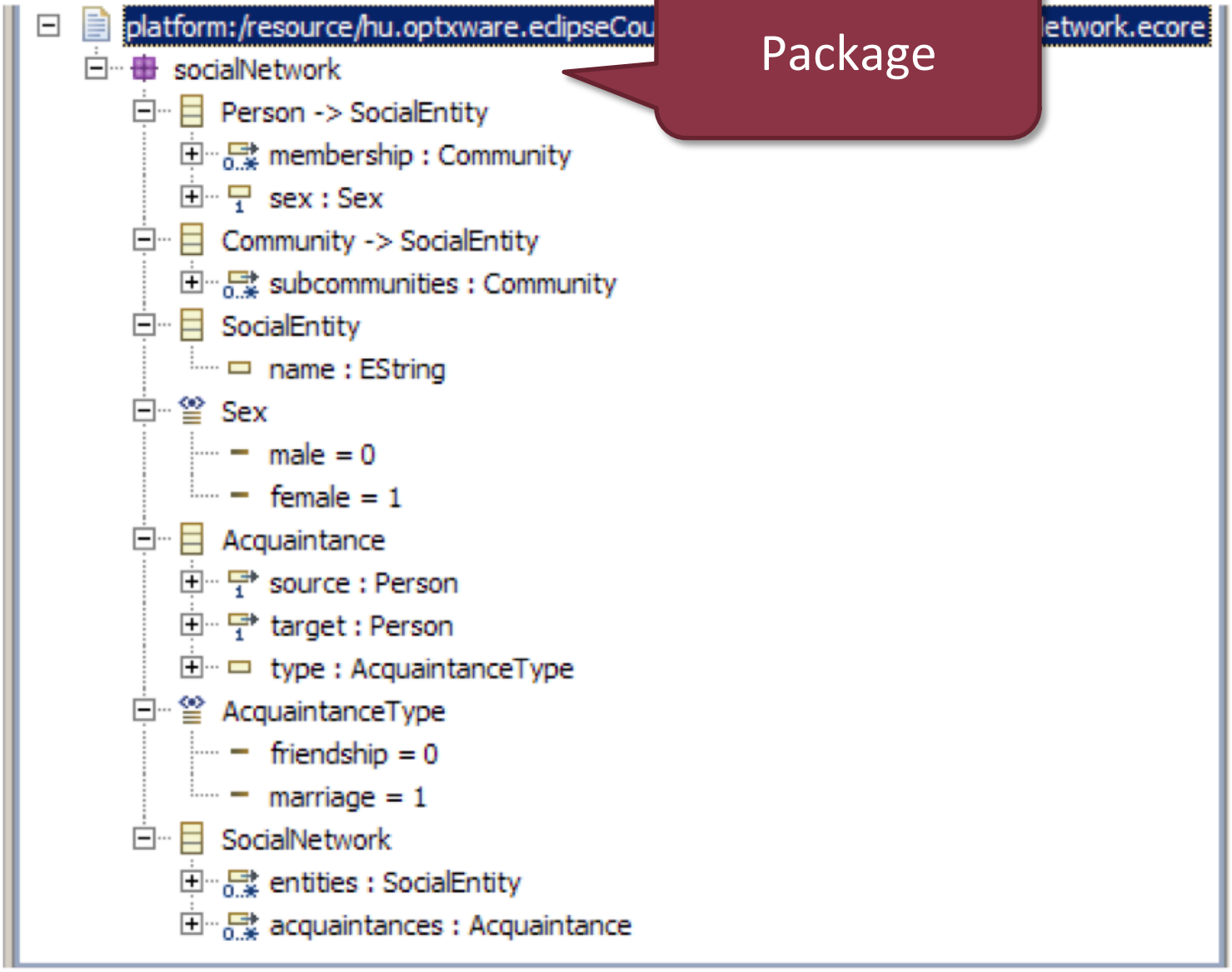
Example: Social Network



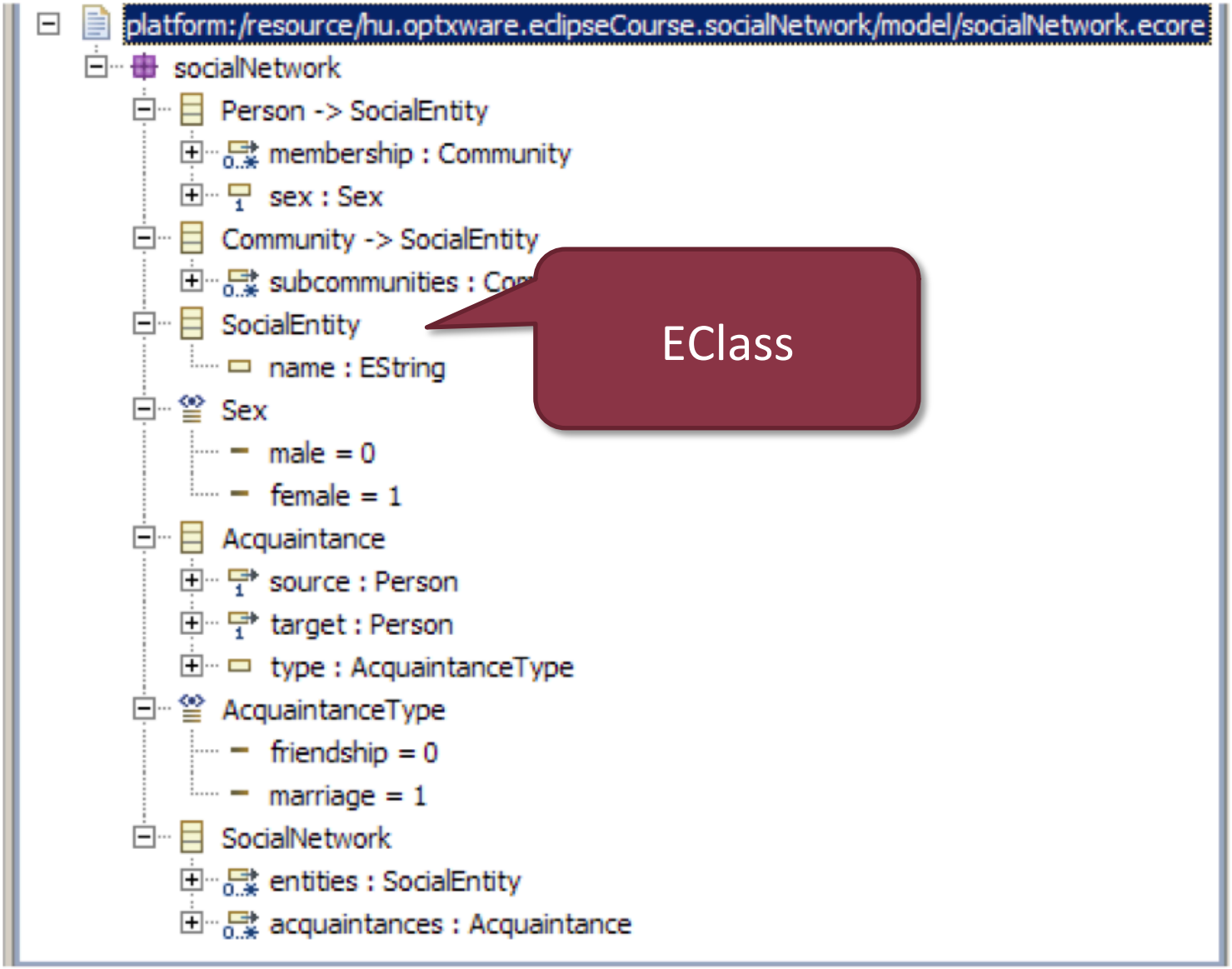
Example: Social network



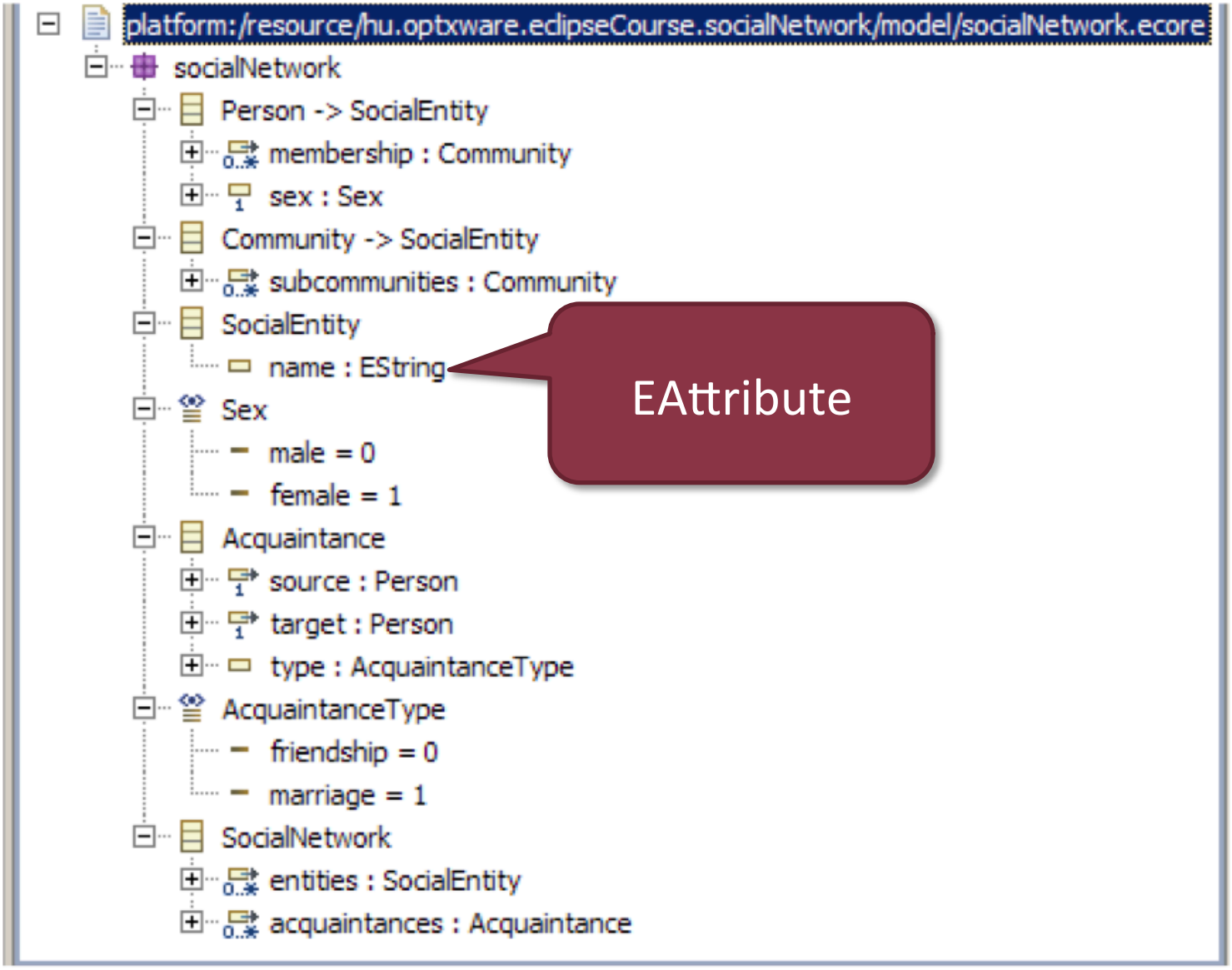
Example: Social network



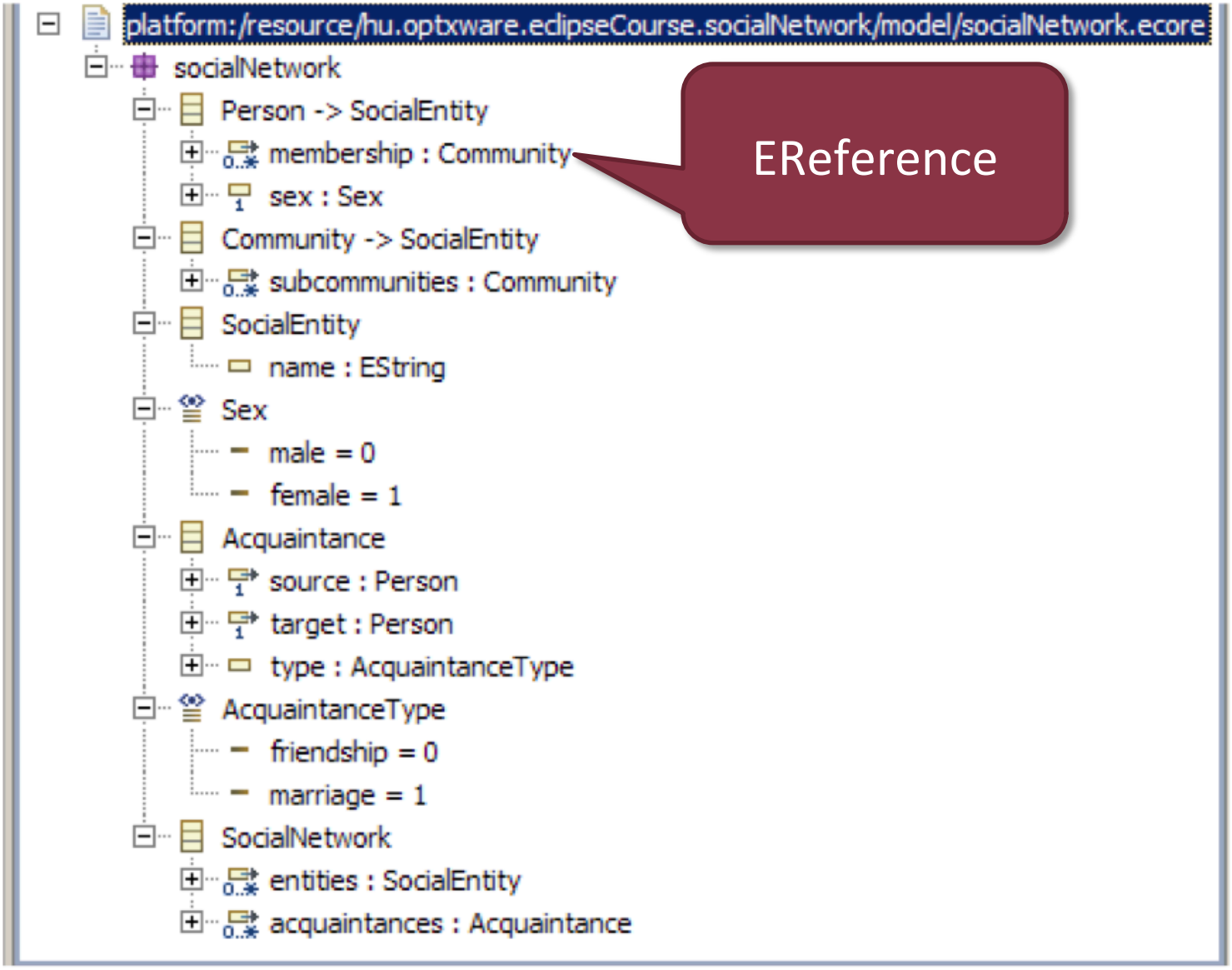
Example: Social network



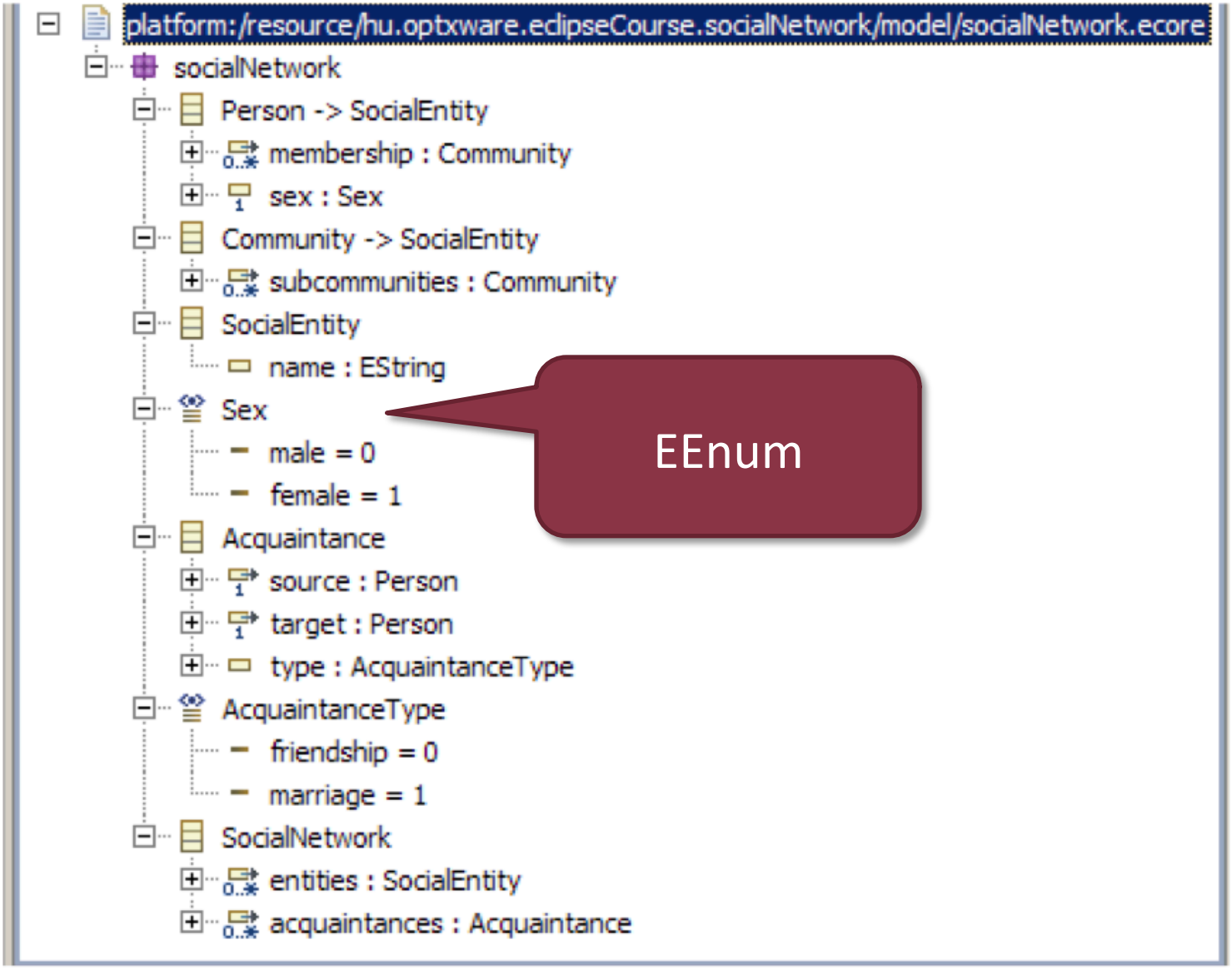
Example: Social network



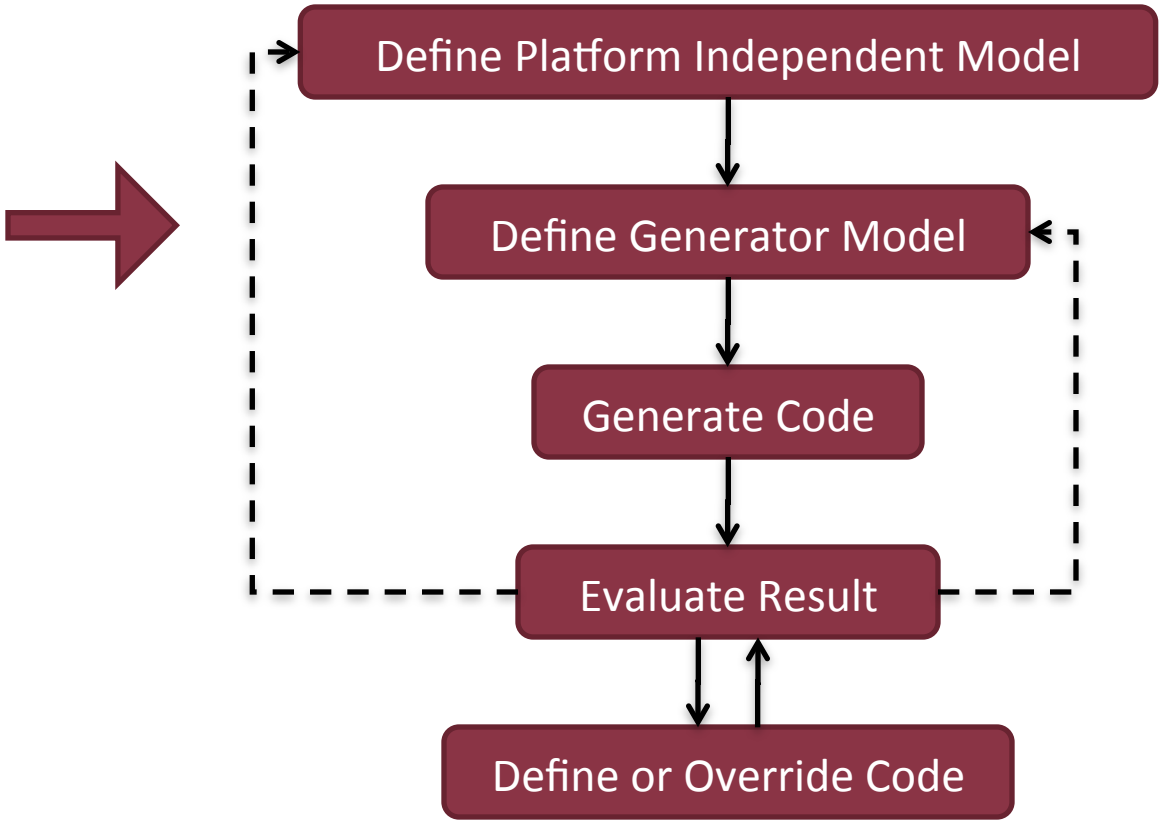
Example: Social network



Example: Social network



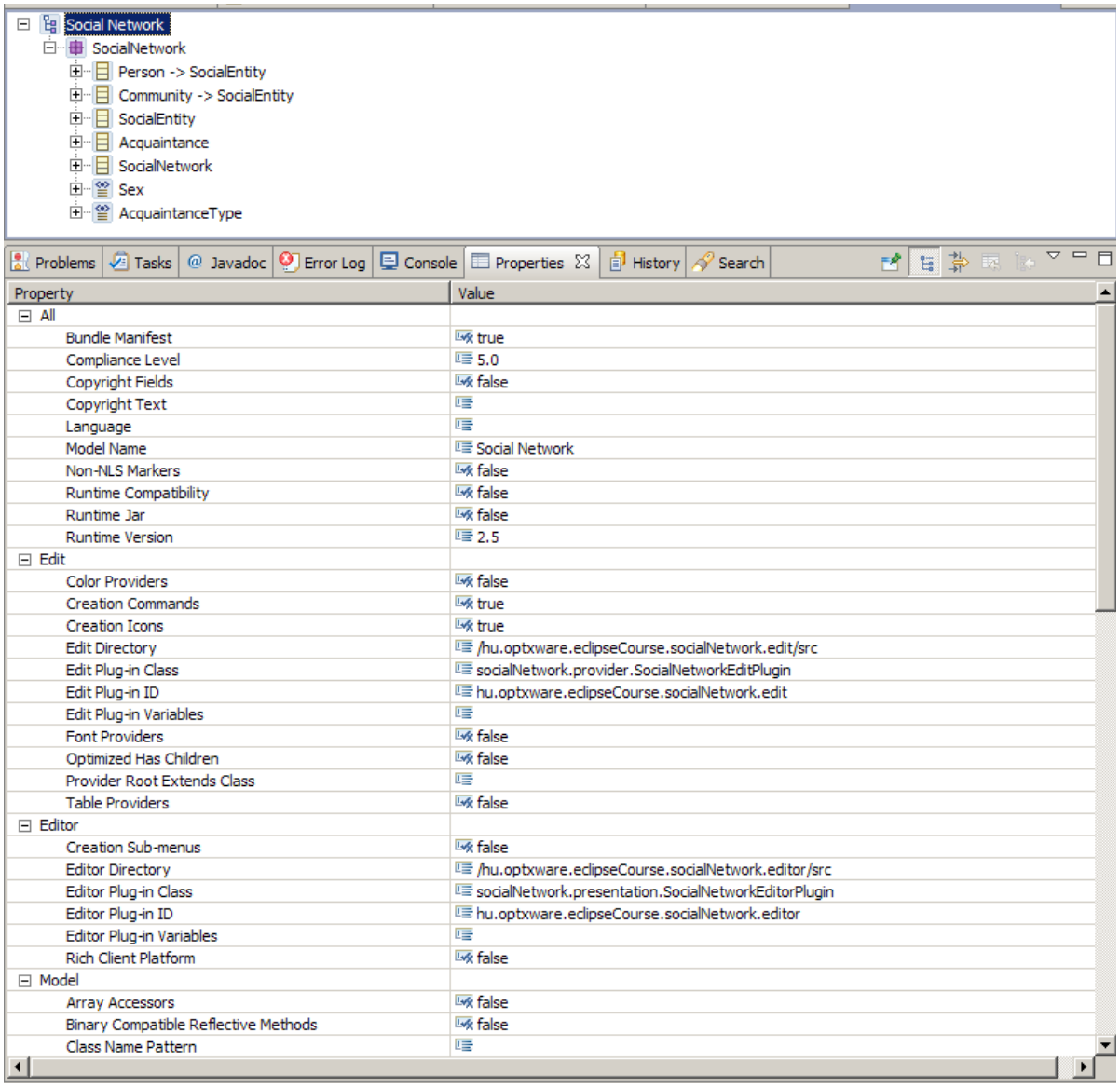
Using EMF


Define Platform Independent Model
Define Generator Model
Generate Code
Evaluate Result
Define or Override Code

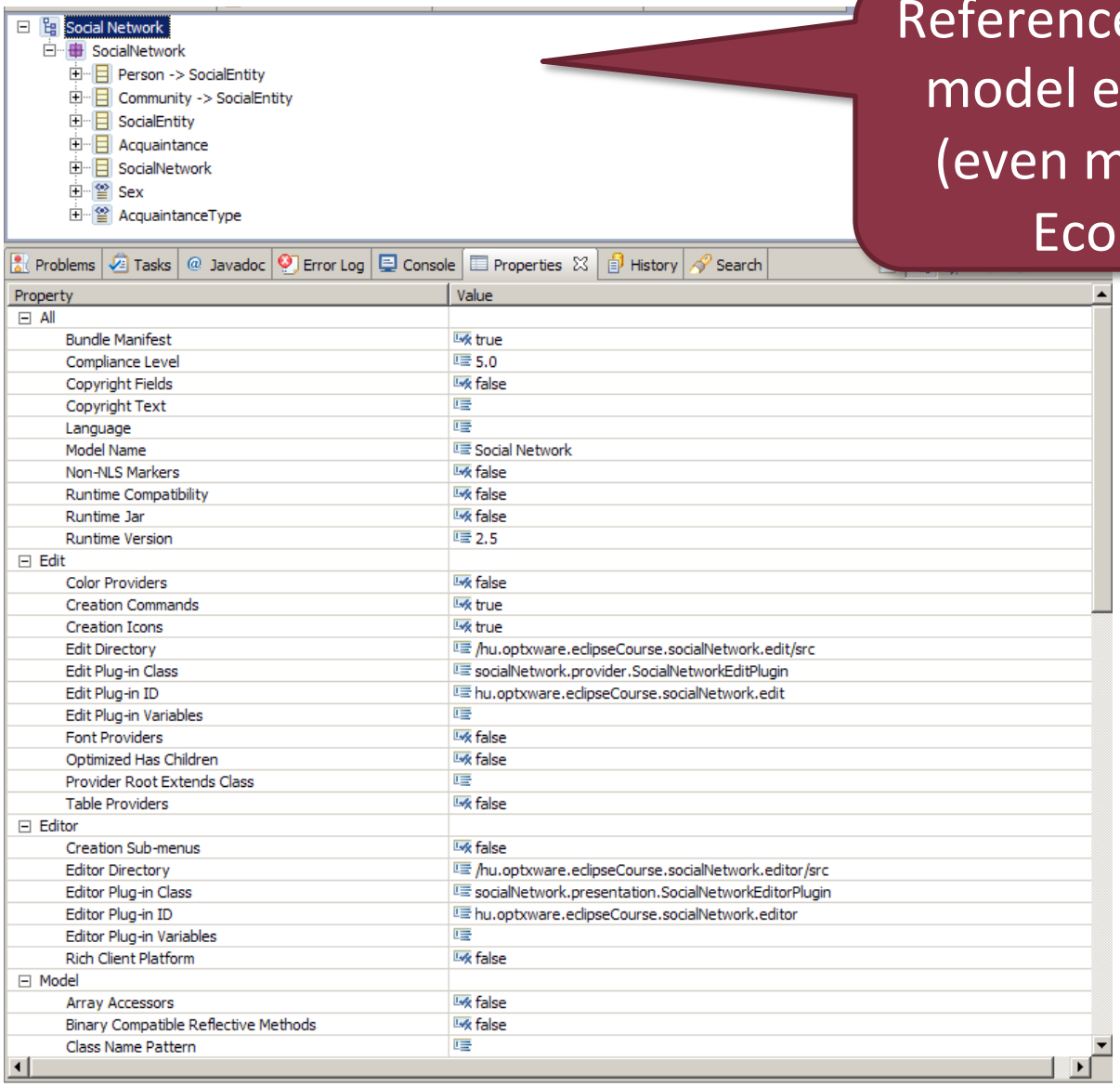
Generator model

- Providing code generation settings
- Also an EMF model
  - Refers to our Ecore metamodel
  - But uses a different metamodel
- Code generation settings
  - Java version (e.g. whether Java 5 enums are available)
  - Package and project names
  - …

Generator model

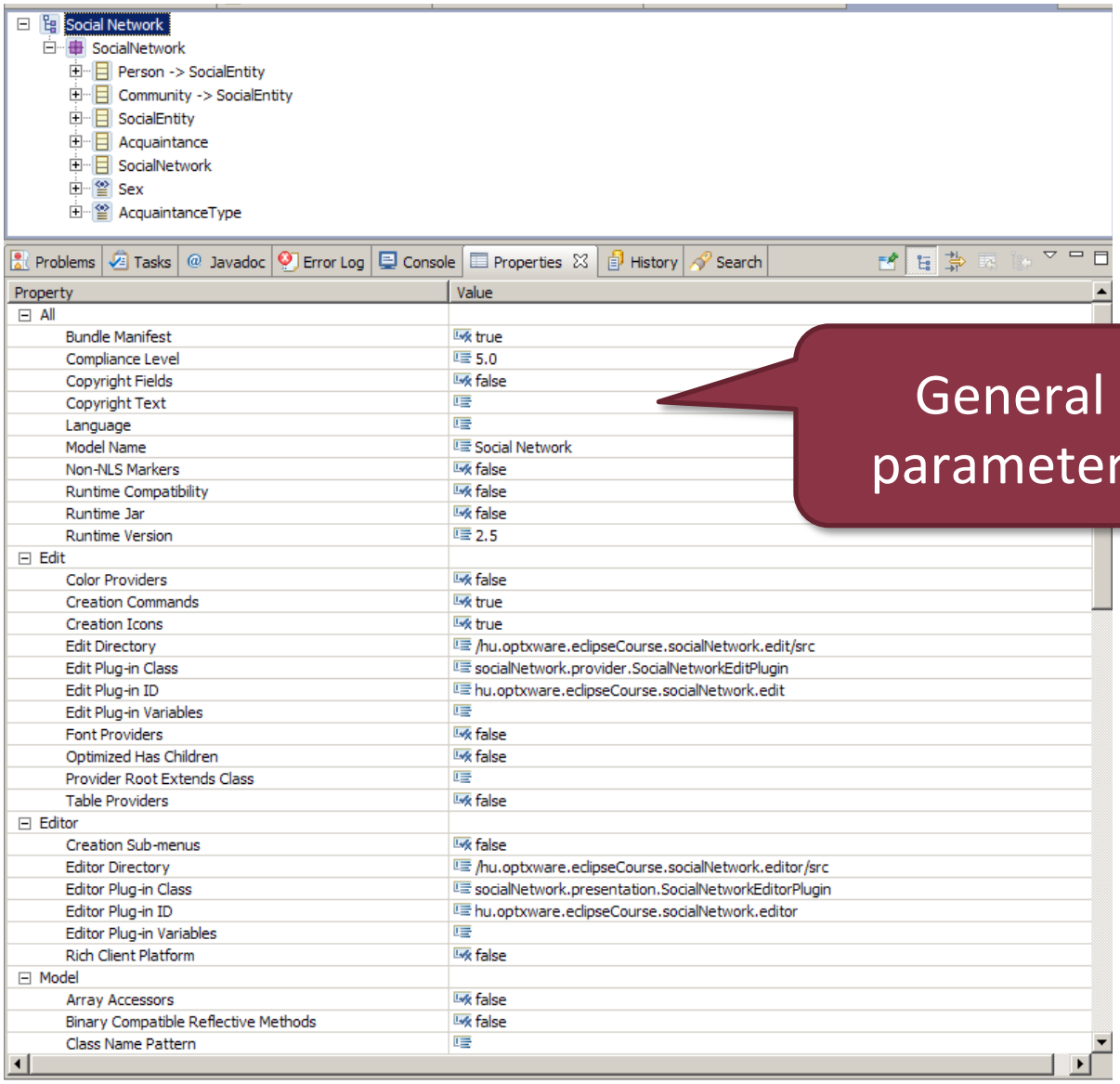


Generator model



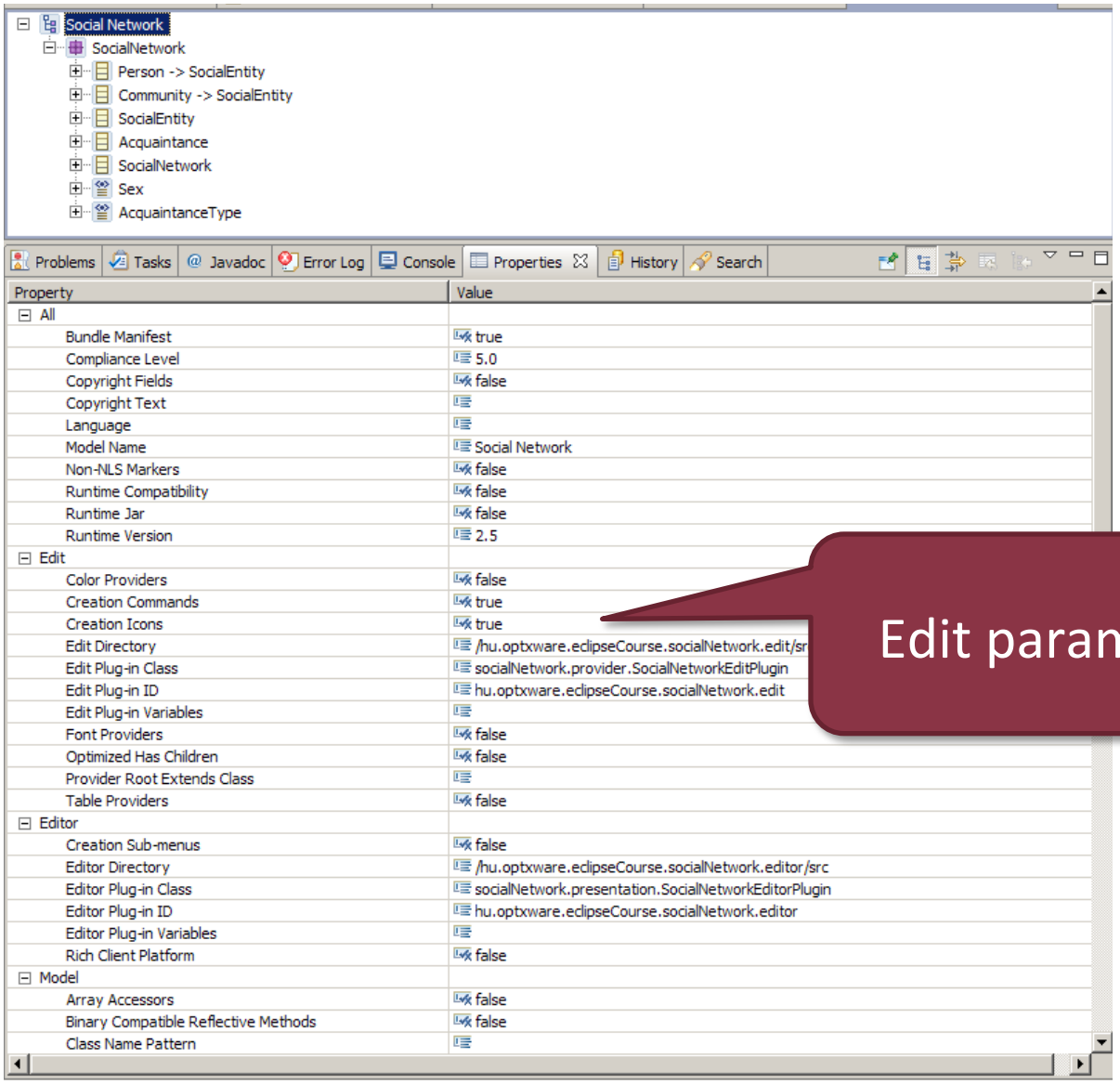
Referenced Ecore model element (even multiple Ecore)

Generator model

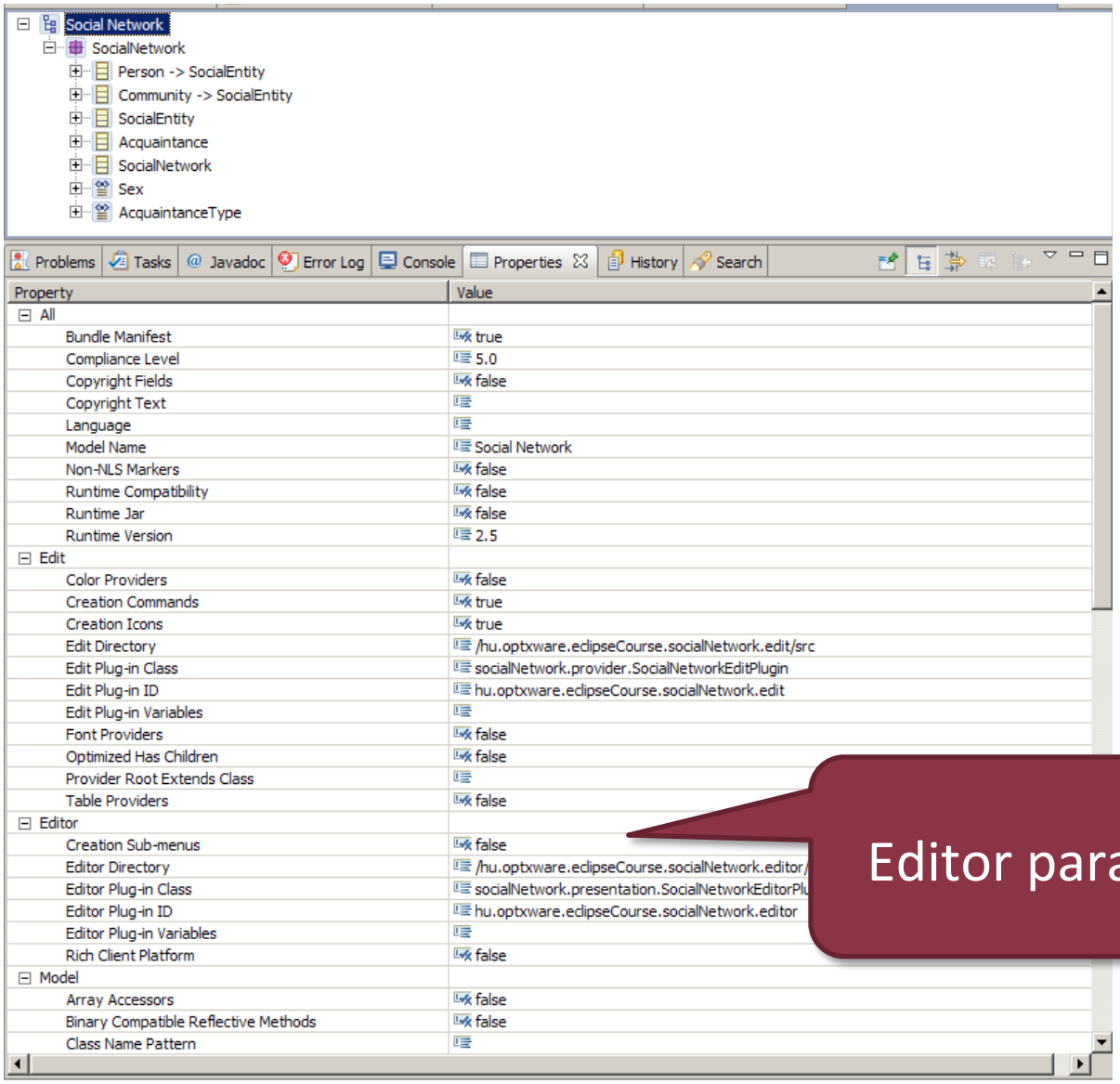


General parameters

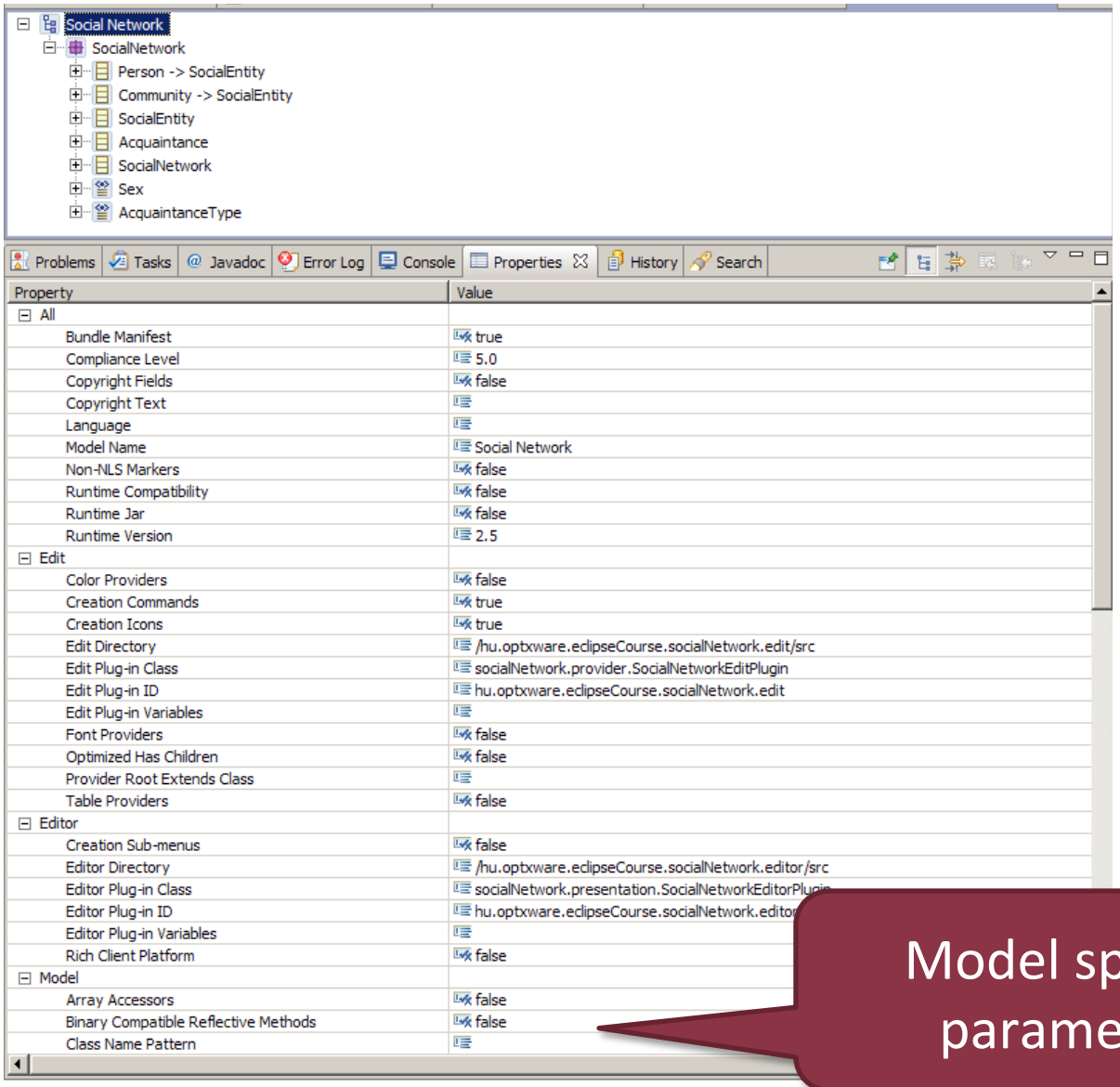
Generator model



Generator model

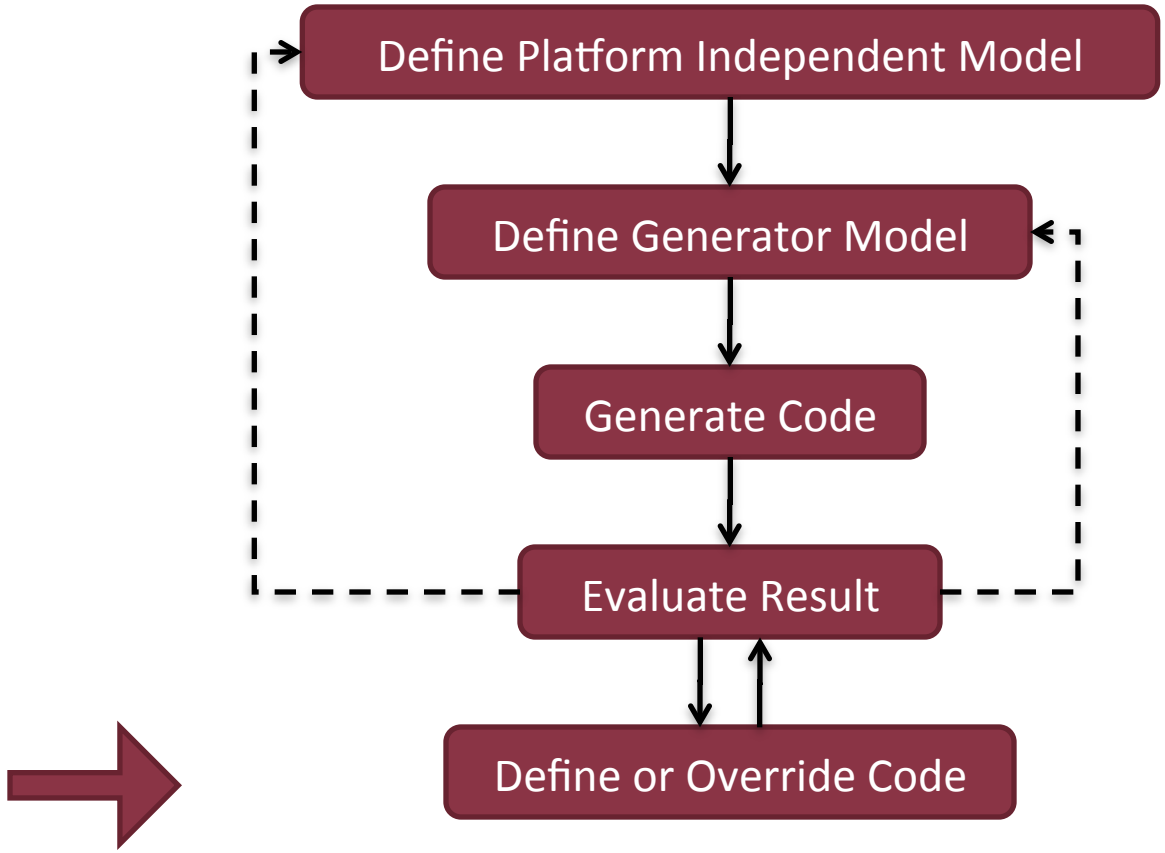


Generator model



Model specific parameters

Using EMF


Define Platform Independent Model
Define Generator Model
Generate Code
Evaluate Result
Define or Override Code

Generated EMF components

EMF.Editor

- Simple tree based editor

EMF.Edit

- User interface data sources
- Commands

EMF.Model

- Model management layer
- Persistence
- Reflective API

Generated EMF components

EMF.Editor

- Simple tree based editor

EMF.Edit

- User interface data sources
- Commands

EMF.Model

- Model management layer
- Persistence
- Reflective API

EMF.Model

- Complete implementation of the ECore metamodel
- Persistence handling
  - By default: XMI technology
  - Additionally: XSD-based XML, binary
  - Can be extended
- Model and code are similar
  - Easy to understand
  - Usually the generated code works well

EMF.Model

- Possible extensions
  - Custom file format
    - Parser
    - See: EMFText/Xtext
  - Inserting extra information into generated code
    - Avoid if possible
      - Maintainability

Reflective and Generated API

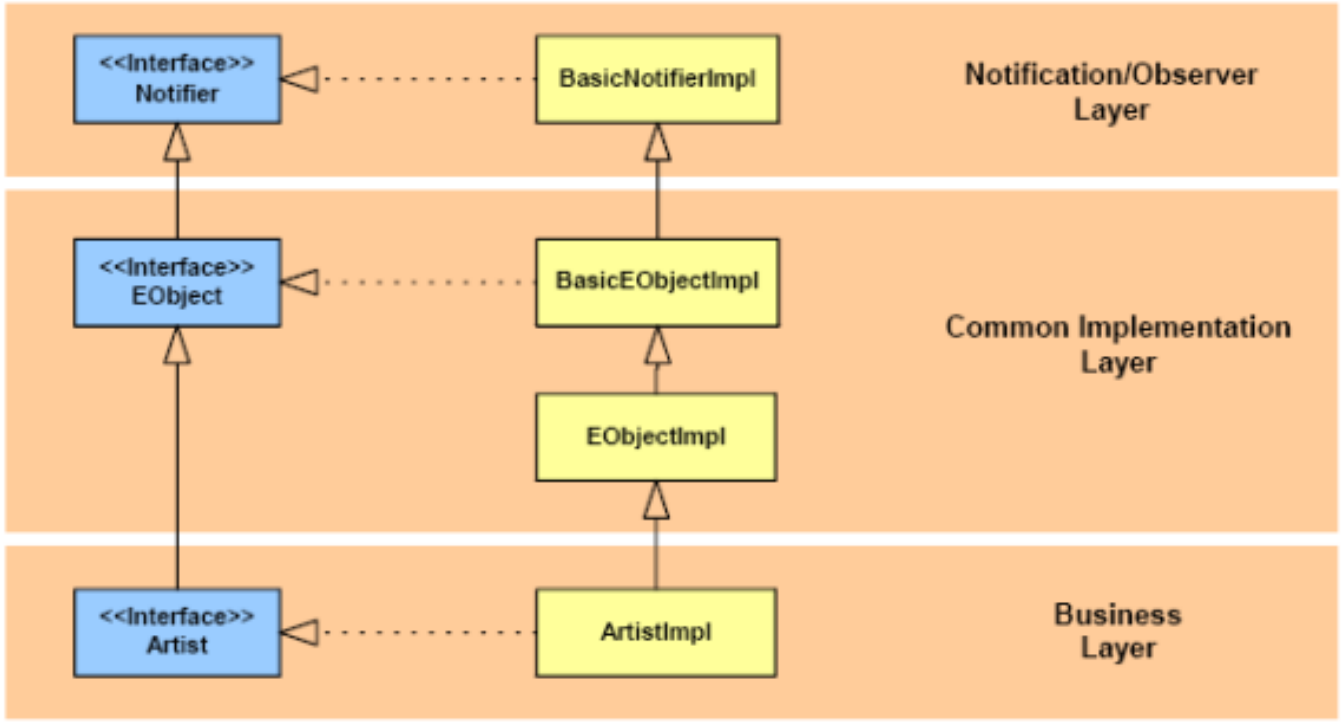
- Generated API
  - Typesafe
  - Simple getter/setter methods
  - Preferred
    - Safe to use
    - Performs better
- Reflective API
  - Parameterizing with Strings or EClass instances
    - See also: dynamic languages, Java reflection
  - For Generic code

Generic or generated implementations

- Generated solution
  - Code generation based on the model
  - Unique code possible for each case
    - E.g., model-specific tree editor, GMF editor, code generator
- Generic solution
  - Same implementation used for all cases
    - E.g., serialization, Reflective Tree Editor

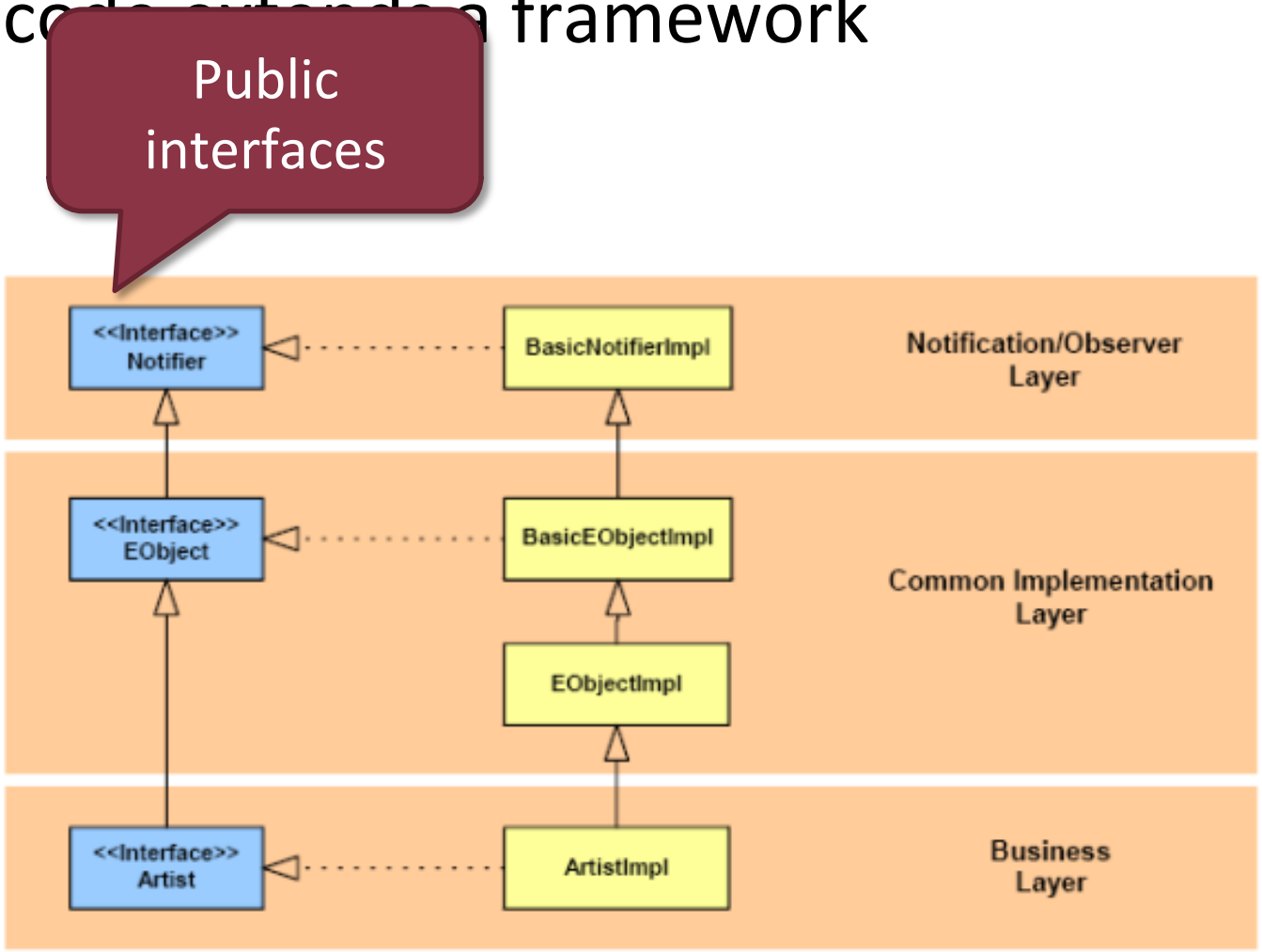
Generated structure

- Generated code extends a framework



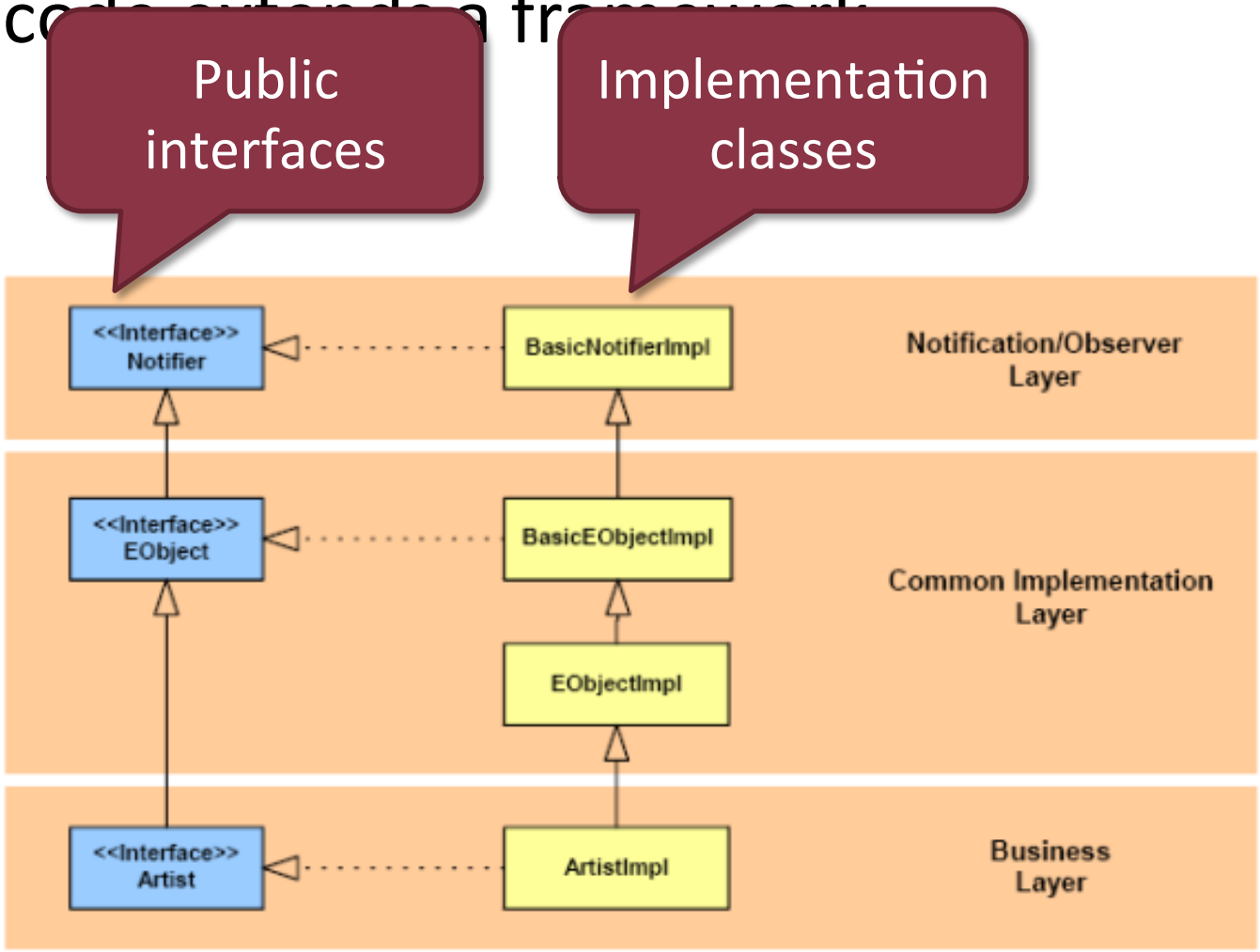
Generated structure

- Generated code extends a framework



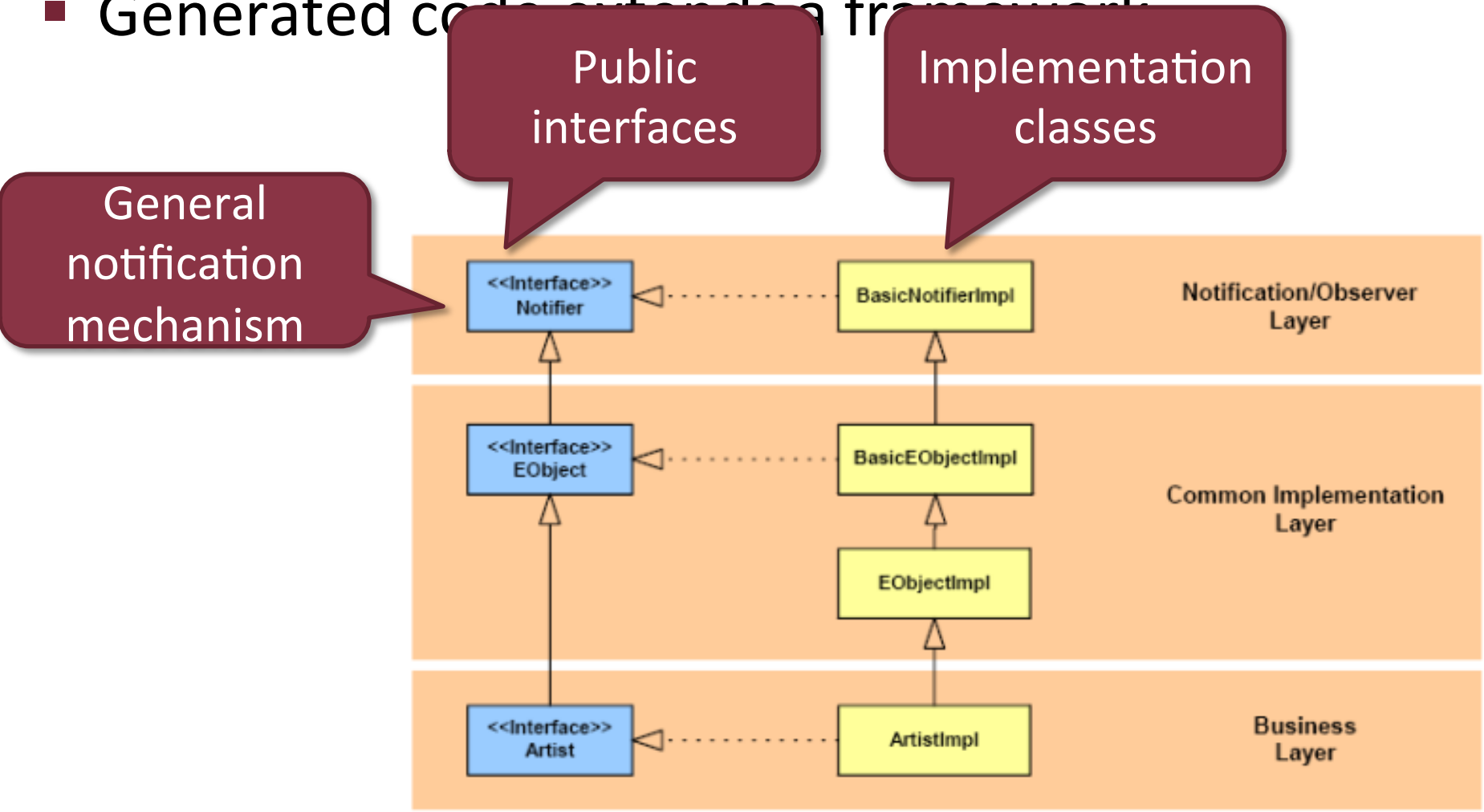
Generated structure

- Generated code extends a framework



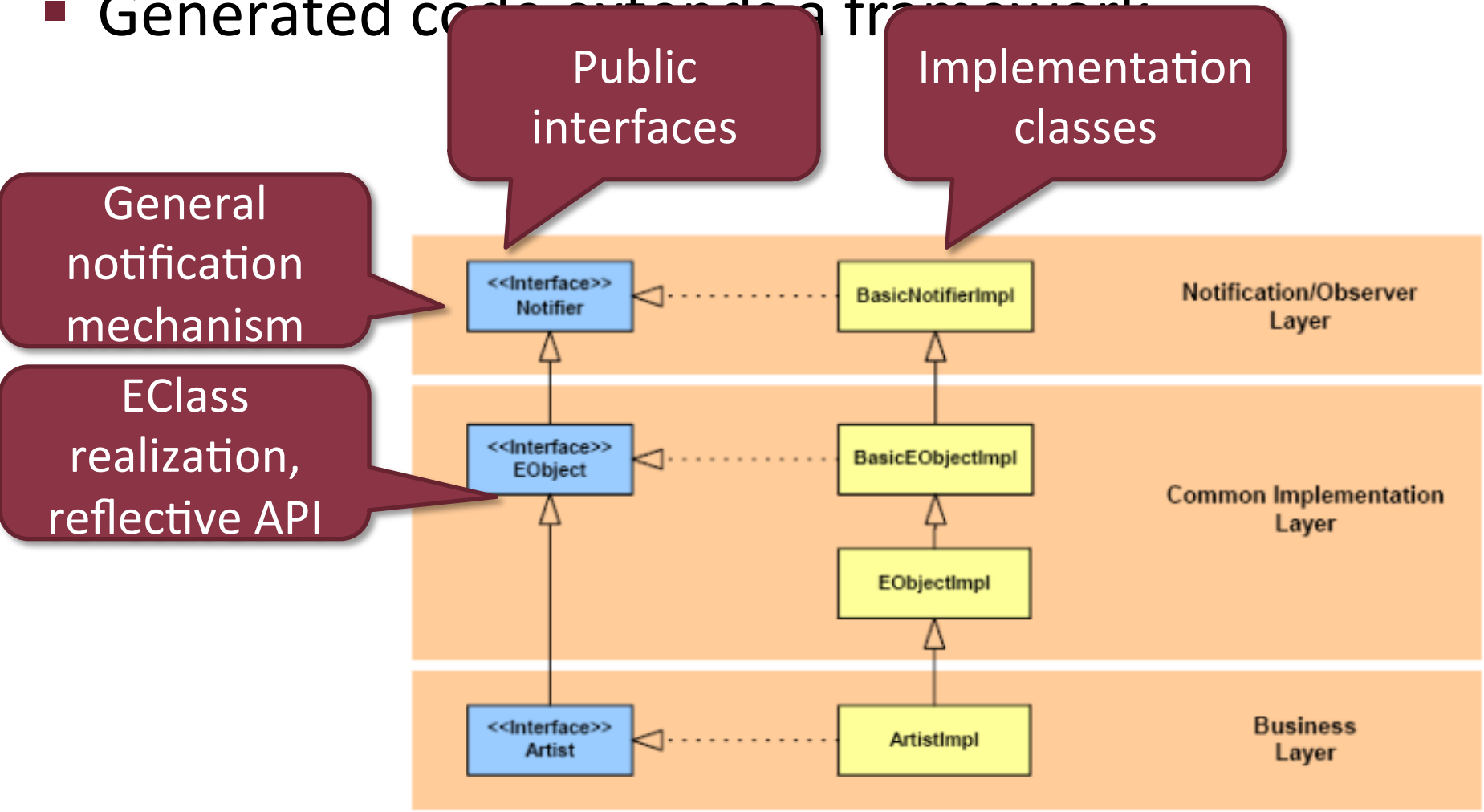
Generated structure

- Generated code extends a framework



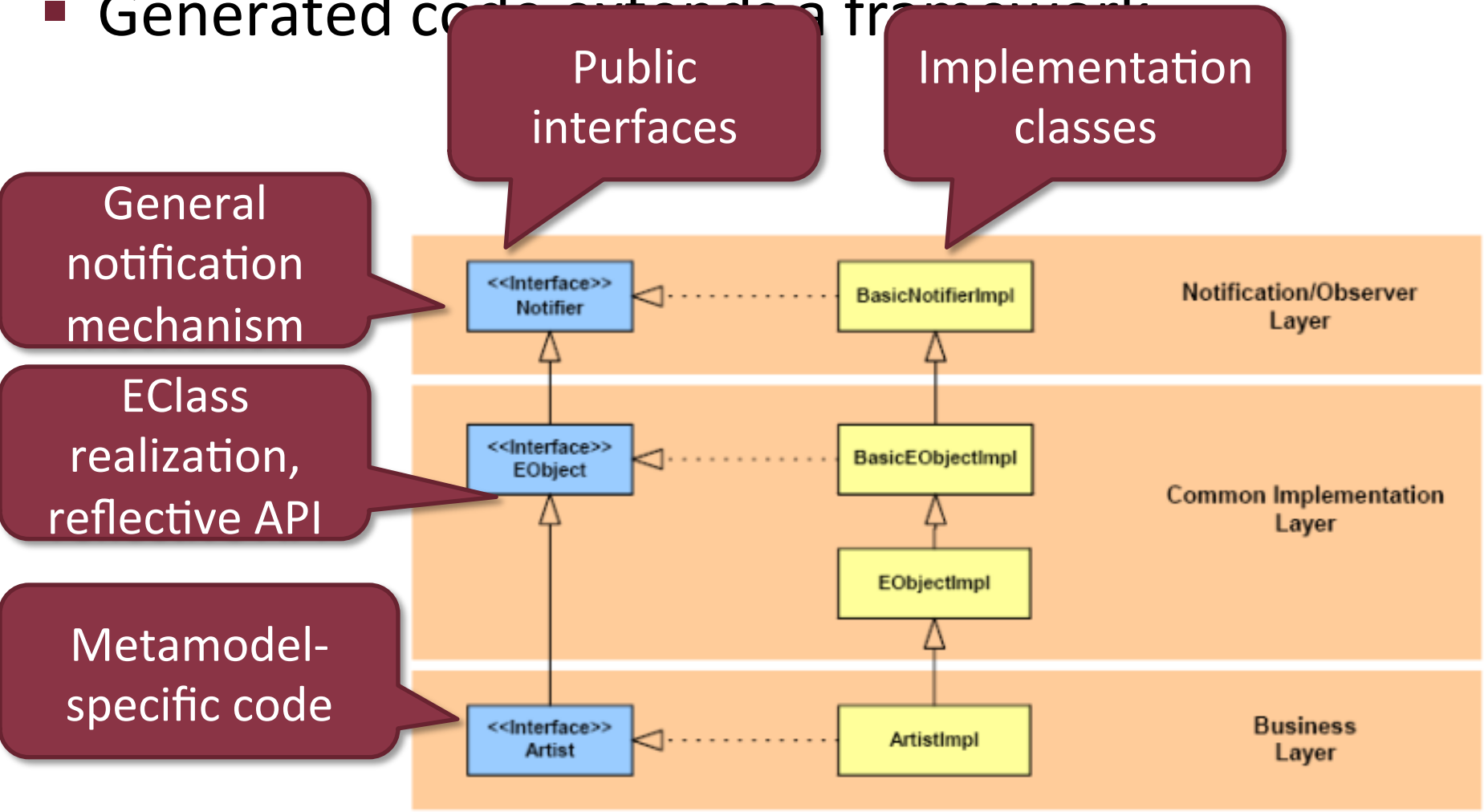
Generated structure

- Generated code extends a framework



Generated structure

- Generated code extends a framework



EClass implementation

- Manages attributes and references (feature)
  - Features with "*one*" multiplicity
    - getter method
    - setter method
  - Features with "*many*" multiplicity
    - getter method returns an **editable** collection

EReference implementation

- Type of an EReference is another EObject
- Some non-trivial cases
  - EClass can be stored in another file (Resource)
    - Reference resolution
  - Inverse references
    - Automatic synchronization

Using EOperation

- Method declaration in EClasses
- Method body
  - Ecore does not support specification
  - See Xcore project

EFactory

- EMF object must be created with factory
- Singleton instance
  - `<Package>Factory.eINSTANCE`
  - `<Package>Package.eINSTANCE`
    `.get<csomagnév>Factory`
- Concrete method for each type
  - `<typename> create<typename>`

EPackage implementation

- Singleton instance: <name>Package.eINSTANCE
- Contains EClass type literals
  - `EClass get<classname>`
  - `EStructuralFeature get<classname>_<featurename>`

Reflection

- eClass()
  - Each EObject can return its class
  - Similar to getClass() of Java
  - Generic property handling
    - eGet/eSet/eIsSet/eSet/eUnset()
- eContainer/eContents()
  - Navigating the containment hierarchy

Reflection

- EMF Provides it
- Uses it for generic services
  - Serialization
  - Nofitication
  - Switch classes

Notification

- Every model object sends change notifications
  - Observer design pattern
  - Event objects are sent
  - Notifications can be customized in generator model
- EMF provides the implementation
  - Not recommended to change it…

Notification

- Subscription
  - `eAdapters().add(Adapter)`
  - By default it is not recursive!
- Notification sending
  - `eNotify(Notification)`
  - Seldom required manually

Serializing EMF models

- EMF models are stored in resources
- An object is contained in a Resource instance
  - See `eResource()`
- Built-in implementation: `XMIResourceImpl`
  - Relies on XMI format

Example: Reading and modifying a model

```
ResourceSet set = new ResourceSetImpl();

URI uri = ...;
Resource res = set.getResource(uri, true);
try {

      for (EObject root : res.getContents()) {
            //TODO Model processing here
      }

      res.save(new HashMap<String, String>());
} catch (IOException e) {
      // TODO Exception handling here!
      e.printStackTrace();
}
```

Example: Reading and modifying a model

```
ResourceSet set = new ResourceSetImpl();

URI uri = ...;
Resource res = set.getResource(uri, tr
try {

      for (EObject root : res.getContents()) {
            //TODO Model processing here
      }

      res.save(new HashMap<String, String>());
} catch (IOException e) {
      // TODO Exception handling here!
      e.printStackTrace();
}
```

ResourceSet instantiation

Example: Reading and modifying a model

```
ResourceSet set = new ResourceSetImpl();

URI uri = ...;
Resource res = set.getReso
try {

      for (EObject root : res.getContents()) {
            //TODO Model processing here
      }

      res.save(new HashMap<String, String>());
} catch (IOException e) {
      // TODO Exception handling here!
      e.printStackTrace();
}
```

URI: model file selection

Example: Reading and modifying a model

```
ResourceSet set = new ResourceSetImpl();

URI uri = ...;
Resource res = set.getResource(uri, true);
try {

      for (EObject root : res.getConten
            //TODO Model processing here
      }

      res.save(new HashMap<String, String>());
} catch (IOException e) {
      // TODO Exception handling here!
      e.printStackTrace();
}
```

getResource: on-demand loading here

Example: Reading and modifying a model

```
ResourceSet set = new ResourceSetImpl();

URI uri = ...;
Resource res = set.getResource(uri, true);
try {

      for (EObject root : res.getContents()) {
            //TODO Model processing here
      }

      res.save(new HashMap<String, St
} catch (IOException e) {
      // TODO Exception handling here!
      e.printStackTrace();
}
```

Multiple model roots possible
Type safety is not checked

Example: Reading and modifying a model

```
ResourceSet set = new ResourceSetImpl();

URI uri = ...;
Resource res = set.getResource(uri, true);
try {

      for (EObject root : res.getContents()) {
            //TODO Model processing here
      }

      res.save(new HashMap<String, String>());
} catch (IOException e) {
      // TODO Exception handling here!
      e.printStackTrace();
}
```

Saving

Example: Reading and modifying a model

```
ResourceSet set = new ResourceSetImpl();

URI uri = ...;
Resource res = set.getResource(uri, true);
try {

      for (EObject root : res.getContents()) {
            //TODO Model processing here
      }

      res.save(new HashMap<String, String>());
} catch (IOException e) {
      // TODO Exception handling here!
      e.printStackTrace();
}
```

Don’t forget exception handling

URI schema

- Resource location is described
  - By an URI
  - Creatable by the URI class
- Built-in support (extensible)
  - File (java.io.File)
    - `URI uri = URI.createFileURI("/path/to/file”)`
  - Eclipse workspace file (IFile)
    - `URI.createPlatformResourceURI("projectName/foldername/filename", true)`
  - File packaged in an Eclipse plug-in
    - `URI.createPlatformPluginURI("pluginName/folderName/filename", true)`

Serialization and the containment hierarchy

- Containment hierarchy is critical
  - Defined in the metamodel
    - Enumerating all containment reference types
- Containment hierarchy must be a forest (on the instance level)
  - Each EObject must have at most one parent
  - Via containment references all contents must be available
  - No circles allowed
  - In case of erroneous structure **serialization errors** occur
    - Referenced model element is not in the hierarchy
    - Circle in the containment hierarchy

Important questions

Important questions

1. Can a **class** be the **source** of multiple containment references?
2. Can an **object** be the **source** of multiple containment references?
3. Can a **class** be the **target** of multiple containment references?
4. Can an **object** be the **target** of multiple containment references?

Important questions

1. Can a **class** be the **source** of multiple containment references?
2. Can an **object** be the **source** of multiple containment references?
3. Can a **class** be the **target** of multiple containment references?
4. Can an **object** be the **target** of multiple containment references?

Important questions

1. Can a **class** be the **source** of multiple containment references?
2. Can an **object** be the **source** of multiple containment references?
3. Can a **class** be the **target** of multiple containment references?
4. Can an **object** be the **target** of multiple containment references?

Important questions

1. Can a **class** be the **source** of multiple containment references?
2. Can an **object** be the **source** of multiple containment references?
3. Can a **class** be the **target** of multiple containment references?
4. Can an **object** be the **target** of multiple containment references?

Important questions

1. Can a **class** be the **source** of multiple containment references?
2. Can an **object** be the **source** of multiple containment references?
3. Can a **class** be the **target** of multiple containment references?
4. Can an **object** be the **target** of multiple containment references?

Important questions

1. Can a **class** be the **source** of multiple containment references?
2. Can an **object** be the **source** of multiple containment references?
3. Can a **class** be the **target** of multiple containment references?
4. Can an **object** be the **target** of multiple containment references?



Important questions

1. Can a **class** be the **source** of multiple containment references?
2. Can an **object** be the **source** of multiple containment references?
3. Can a **class** be the **target** of multiple containment references?
4. Can an **object** be the **target** of multiple containment references?

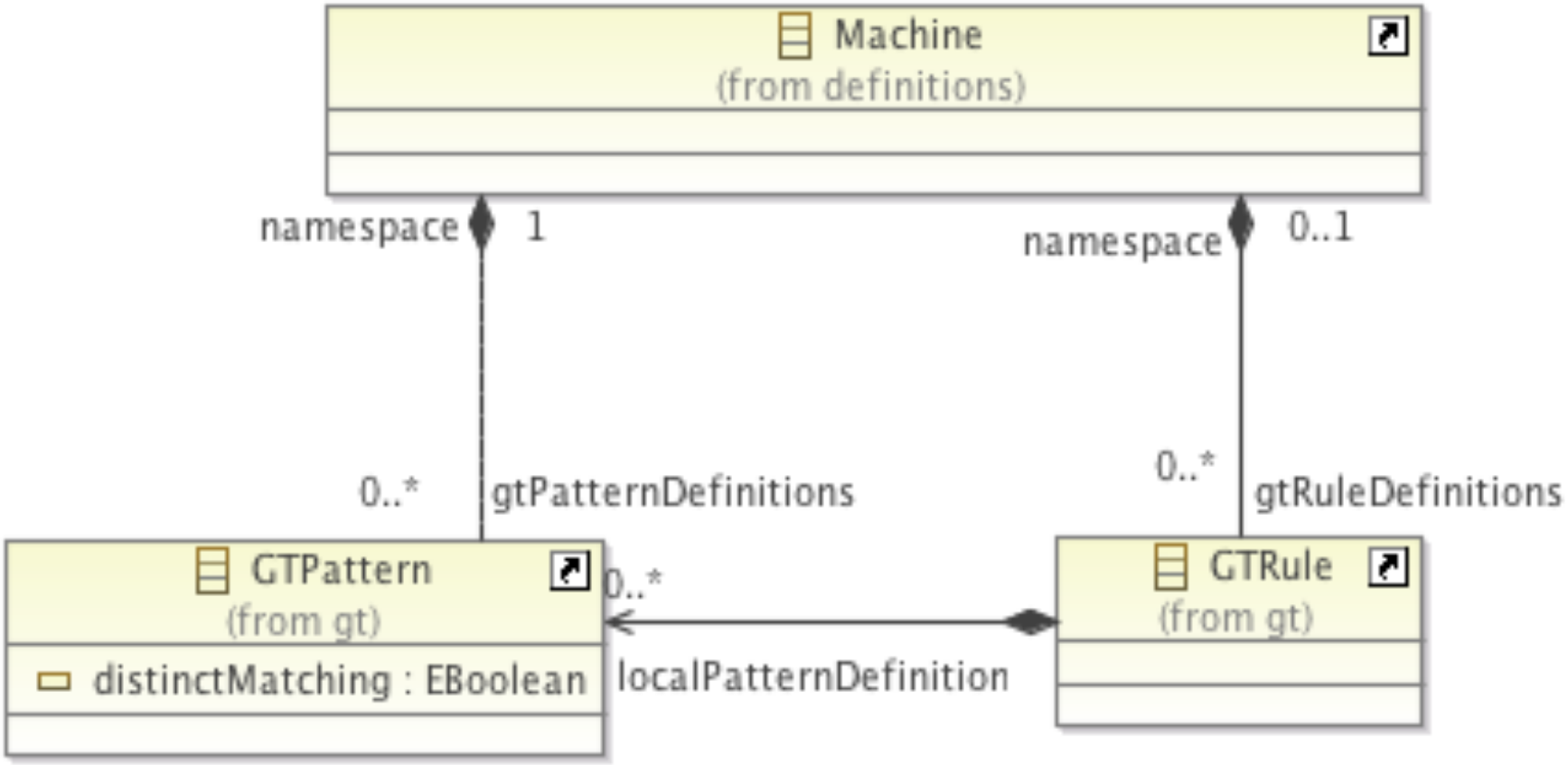


Important questions

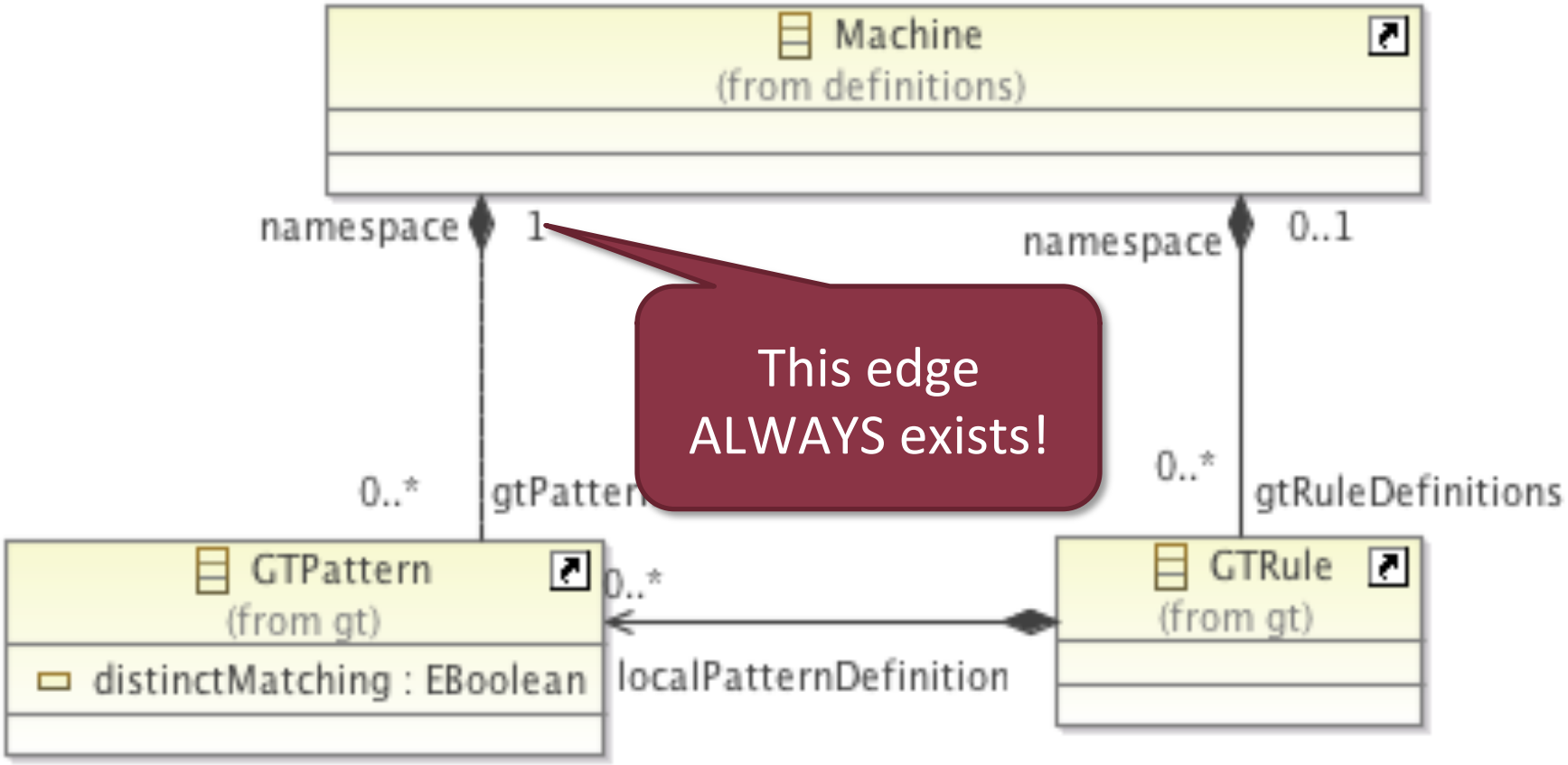
1. Can a **class** be the **source** of multiple containment references?
2. Can an **object** be the **source** of multiple containment references?
3. Can a **class** be the **target** of multiple containment references?
4. Can an **object** be the **target** of multiple containment references?




What is the error in the following metamodel?



What is the error in the following metamodel?



Generated EMF components

EMF.Editor

- Simple tree based editor

EMF.Edit

- User interface data sources
- Commands

EMF.Model

- Model management layer
- Persistence
- Reflective API

Generated EMF components

EMF.Editor

- Simple tree based editor

EMF.Edit

- User interface data sources
- Commands

EMF.Model

- Model management layer
- Persistence
- Reflective API

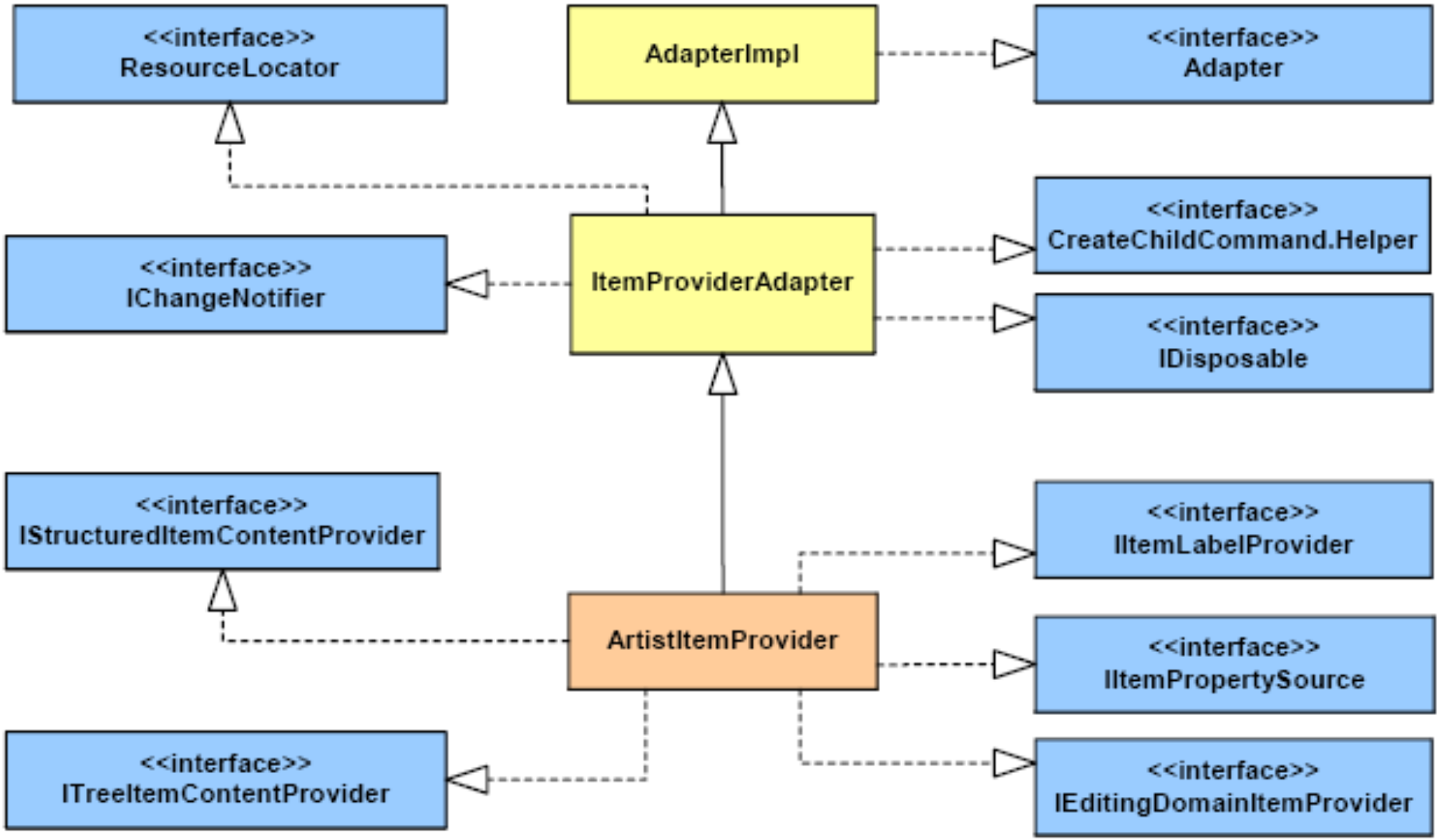
EMF.Edit

- Goal
  - Separation of GUI and model
  - GUI-independent command implementation
- Modifications are not uncommon
  - Element provider updates (different hierarchy)
  - New command definitions

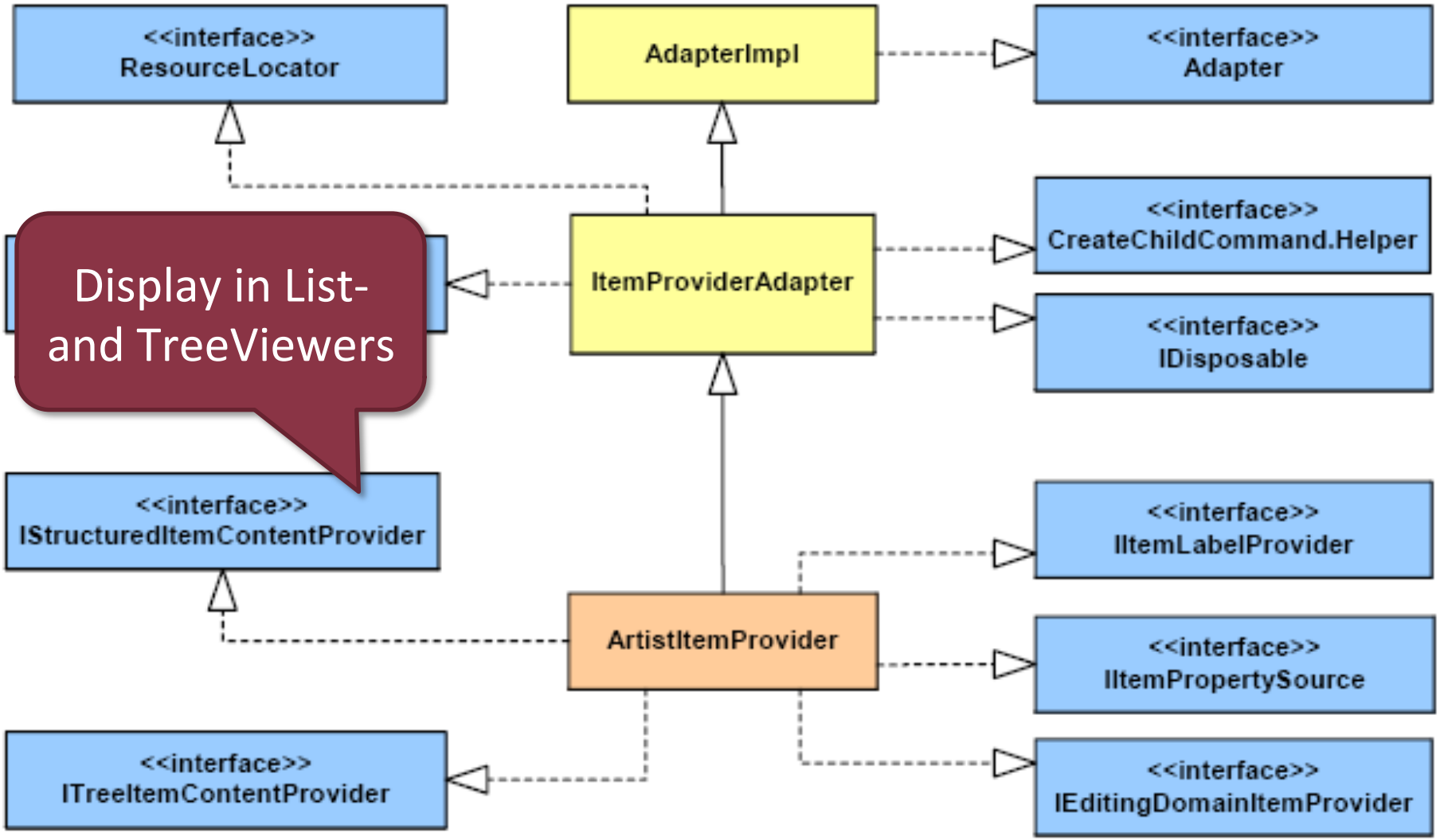
Adapter pattern

- For each model object
  - An adapter is created (ItemProvider)
    - Returns child elements and possible commands
    - E.g., ArtistItemProvider
- Ancestor:
  - org.eclipse.emf.edit.provider.ItemProviderAdapter
    - Default implementation for base behaviour
    - Some parts are redefined

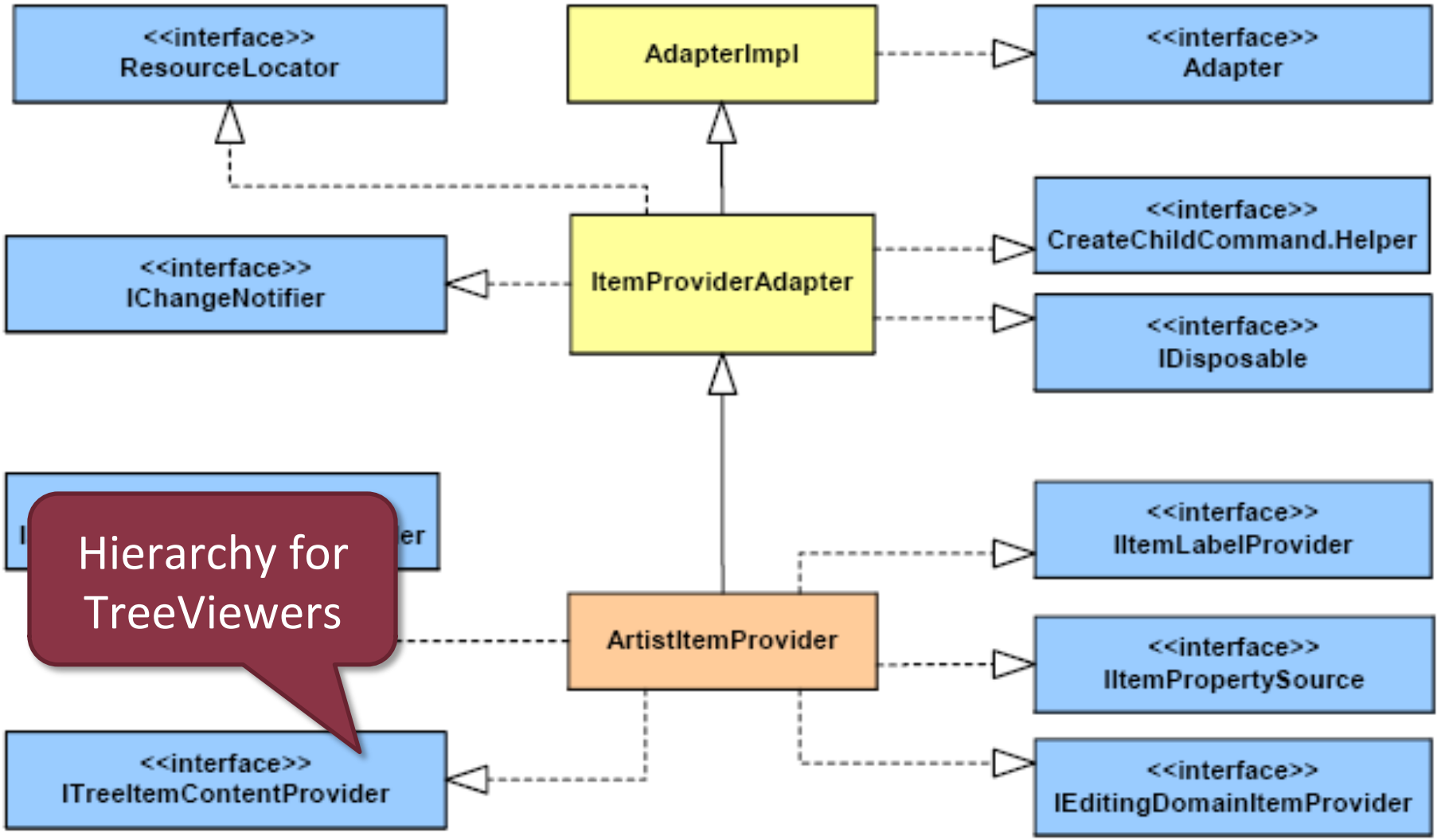
Structure


<<interface>>
ResourceLocator
AdapterImpl
<<interface>>
Adapter
<<interface>>
IChangeNotifier
ItemProviderAdapter
<<interface>>
CreateChildCommand.Helper
<<interface>>
IDisposable
<<interface>>
IStructuredItemContentProvider
<<interface>>
IItemLabelProvider
ArtistItemProvider
<<interface>>
IItemPropertySource
<<interface>>
ITreeItemContentProvider
<<interface>>
IEditingDomainItemProvider

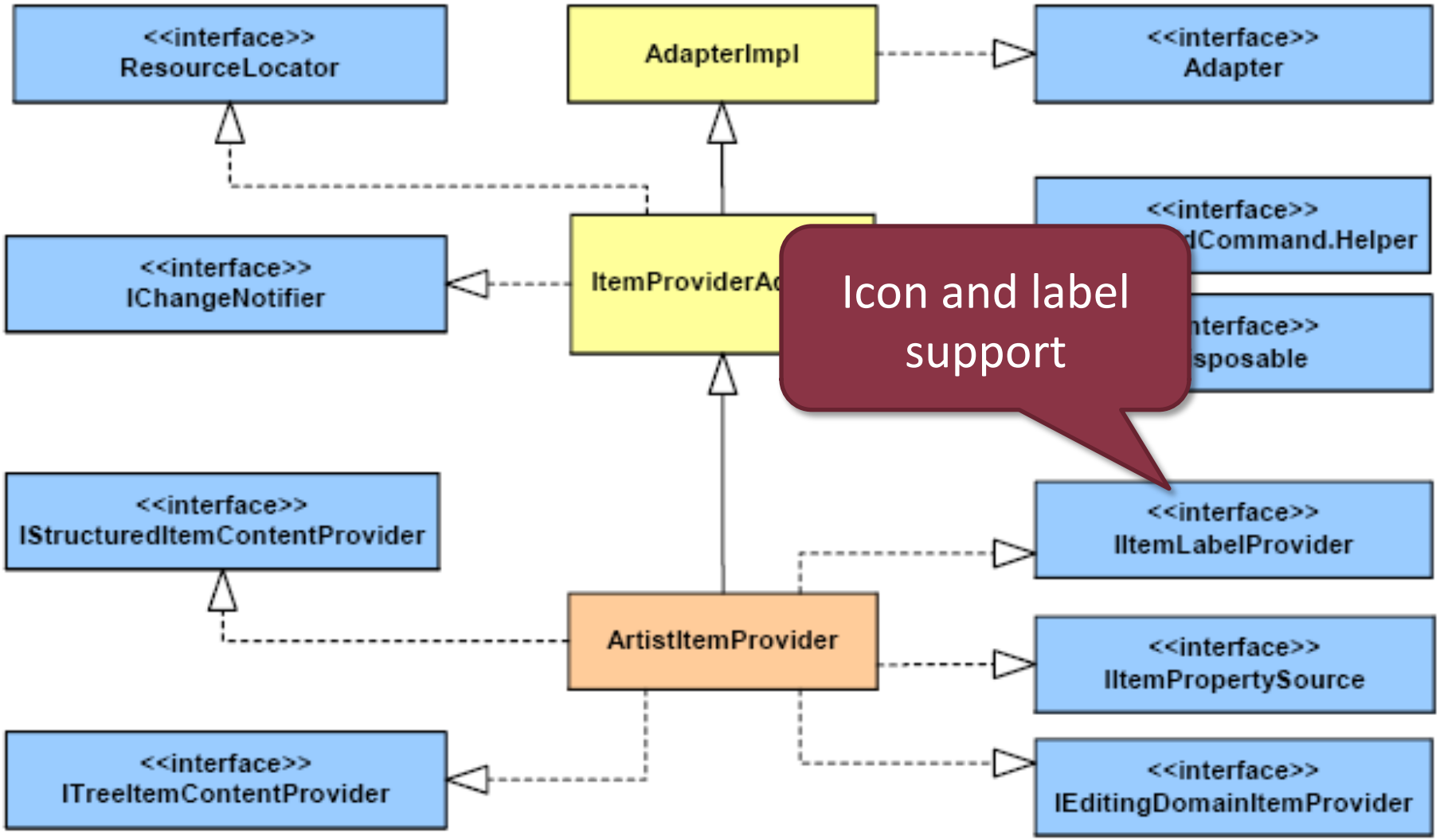
Structure


<<interface>>
ResourceLocator
AdapterImpl
<<interface>>
Adapter
<<interface>>
CreateChildCommand.Helper
ItemProviderAdapter
<<interface>>
IDisposable
Display in List- and TreeViewers
<<interface>>
IStructuredItemContentProvider
<<interface>>
IItemLabelProvider
ArtistItemProvider
<<interface>>
IItemPropertySource
<<interface>>
ITreeItemContentProvider
<<interface>>
IEditingDomainItemProvider

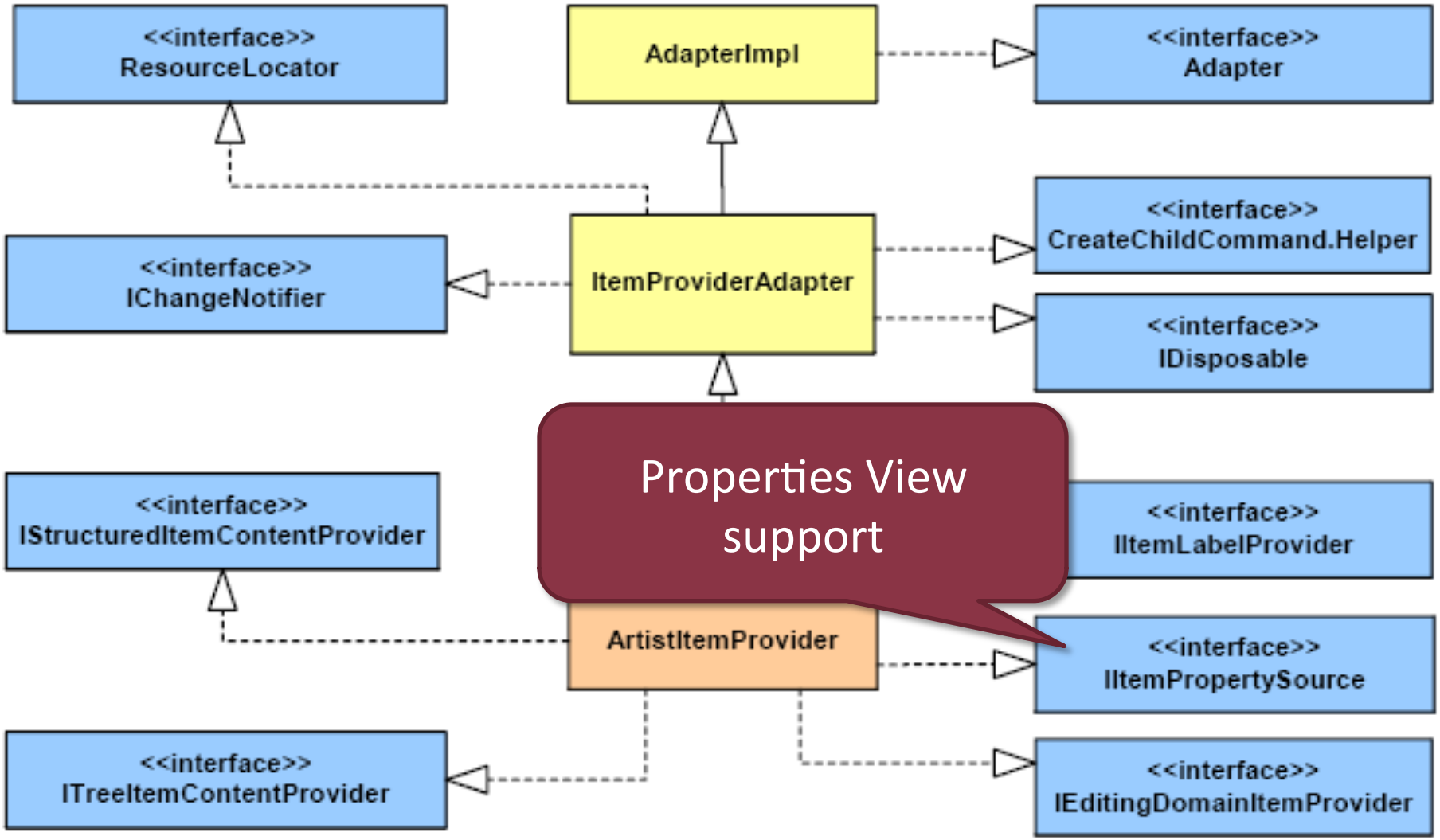
Structure


<<interface>>
ResourceLocator
AdapterImpl
<<interface>>
Adapter
<<interface>>
IChangeNotifier
ItemProviderAdapter
<<interface>>
CreateChildCommand.Helper
<<interface>>
IDisposable
<<interface>>
IItemLabelProvider
ArtistItemProvider
<<interface>>
IItemPropertySource
<<interface>>
ITreeItemContentProvider
<<interface>>
IEditingDomainItemProvider
Hierarchy for
TreeViewers

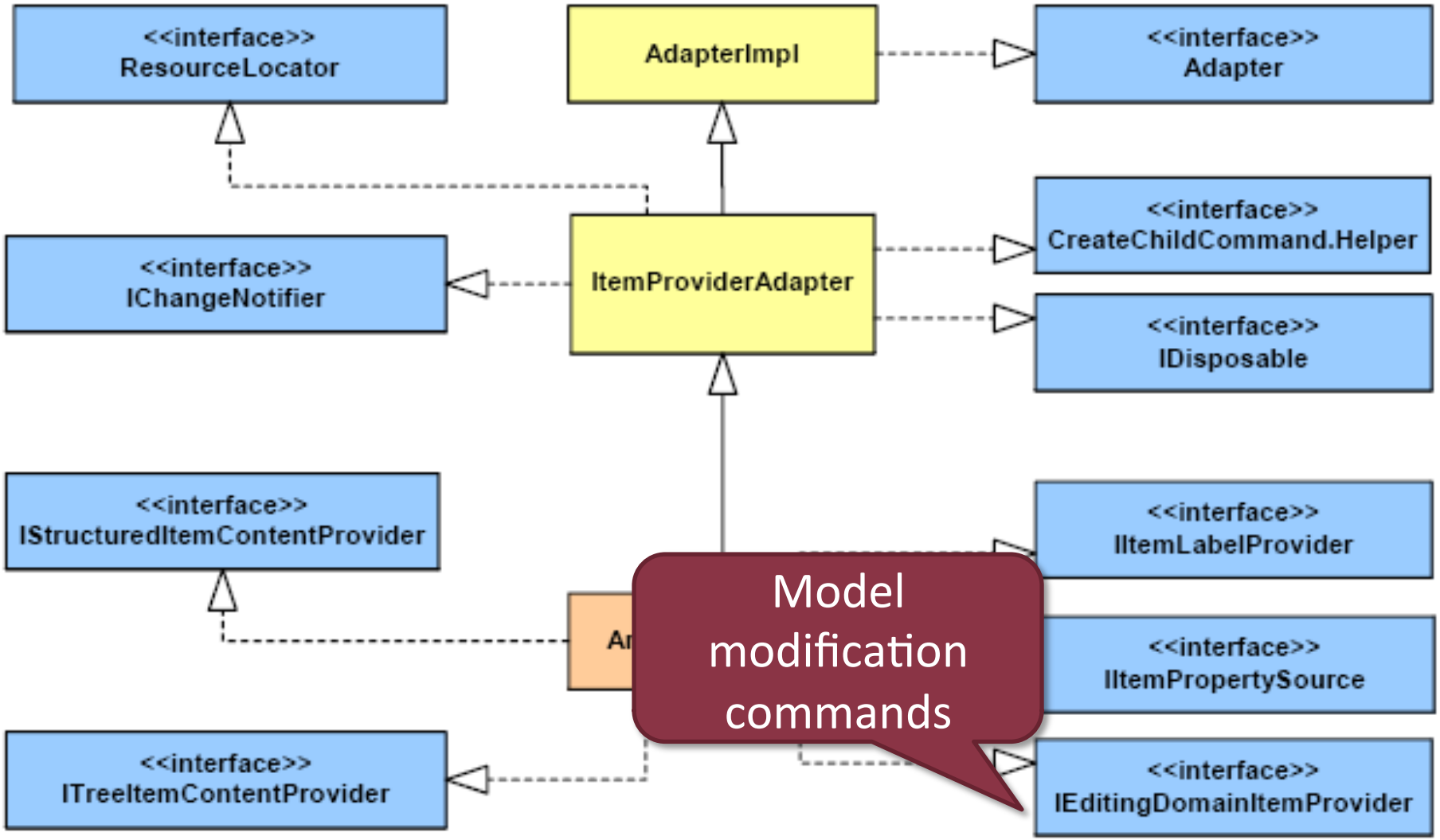
Structure


<<interface>>
ResourceLocator
AdapterImpl
<<interface>>
Adapter
<<interface>>
IChangeNotifier
ItemProviderAd
<<interface>>
dCommand.Helper
nterface>>
sposable
Icon and label support
<<interface>>
IStructuredItemContentProvider
<<interface>>
IItemLabelProvider
ArtistItemProvider
<<interface>>
IItemPropertySource
<<interface>>
ITreeItemContentProvider
<<interface>>
IEditingDomainItemProvider

Structure


<<interface>>
ResourceLocator
AdapterImpl
<<interface>>
Adapter
<<interface>>
IChangeNotifier
ItemProviderAdapter
<<interface>>
CreateChildCommand.Helper
<<interface>>
IDisposable
Properties View support
<<interface>>
IStructuredItemContentProvider
<<interface>>
IItemLabelProvider
ArtistItemProvider
<<interface>>
IItemPropertySource
<<interface>>
ITreeItemContentProvider
<<interface>>
IEditingDomainItemProvider

Structure


<<interface>>
ResourceLocator
AdapterImpl
<<interface>>
Adapter
<<interface>>
IChangeNotifier
ItemProviderAdapter
<<interface>>
CreateChildCommand.Helper
<<interface>>
IDisposable
<<interface>>
IStructuredItemContentProvider
<<interface>>
IItemLabelProvider
<<interface>>
IItemPropertySource
<<interface>>
ITreeItemContentProvider
<<interface>>
IEditingDomainItemProvider
Model modification commands

Replacing default labels

- Common customization of generated edit code
  - Simple cases can be managed in generator model
  - For complex cases update `ItemProvider.getText`
    - Updated `@generated` in the header to `@generated NOT`
    - Then provide custom implementation
      - Be mindful of notification requirements!

Icon change

- Generator: one icon for each model element
  - Base icons generated in edit project
    - obj16: class icons
    - ctool16: child creation command
- Changes
  - Either replace the generated icon
  - Or override `ItemProvider.getImage`

EMF.Edit commands

- Every modification happends through a command
  - Menu action
  - Attribute change
  - Drag-n-drop
  - Required for Undo/redo support
- Combination of generated and generic commands
  - EMF Common Command Framework (CCF)
  - EMF.Edit generated commands

Commands in EMF.Edit

- ItemProvider implements `createCommand()`
- Dispatches queries to overridable methods
  - `createAddCommand()`
  - `createRemoveCommand()`
  - …

Example EMF.Edit command: Logging added

```
public class SetArtistNameCommand extends SetCommand {

	public SetArtistNameCommand(EditingDomain domain,
Eobject owner,
	EStructuralFeature feature, Object value) {
	super(domain, owner, feature, value);
	}

	public void doExecute() {
	Artist artist = (Artist)this.owner;
	Logger.log(MessageFormat.format(
	"Name of artist changed from {0} to {1}",
	artist.getName(), value.toString()));
	super.doExecute();
	}

}
```

Generated EMF components

EMF.Editor

- Simple tree based editor

EMF.Edit

- User interface data sources
- Commands

EMF.Model

- Model management layer
- Persistence
- Reflective API

Generated EMF components

EMF.Editor

- Simple tree based editor

EMF.Edit

- User interface data sources
- Commands

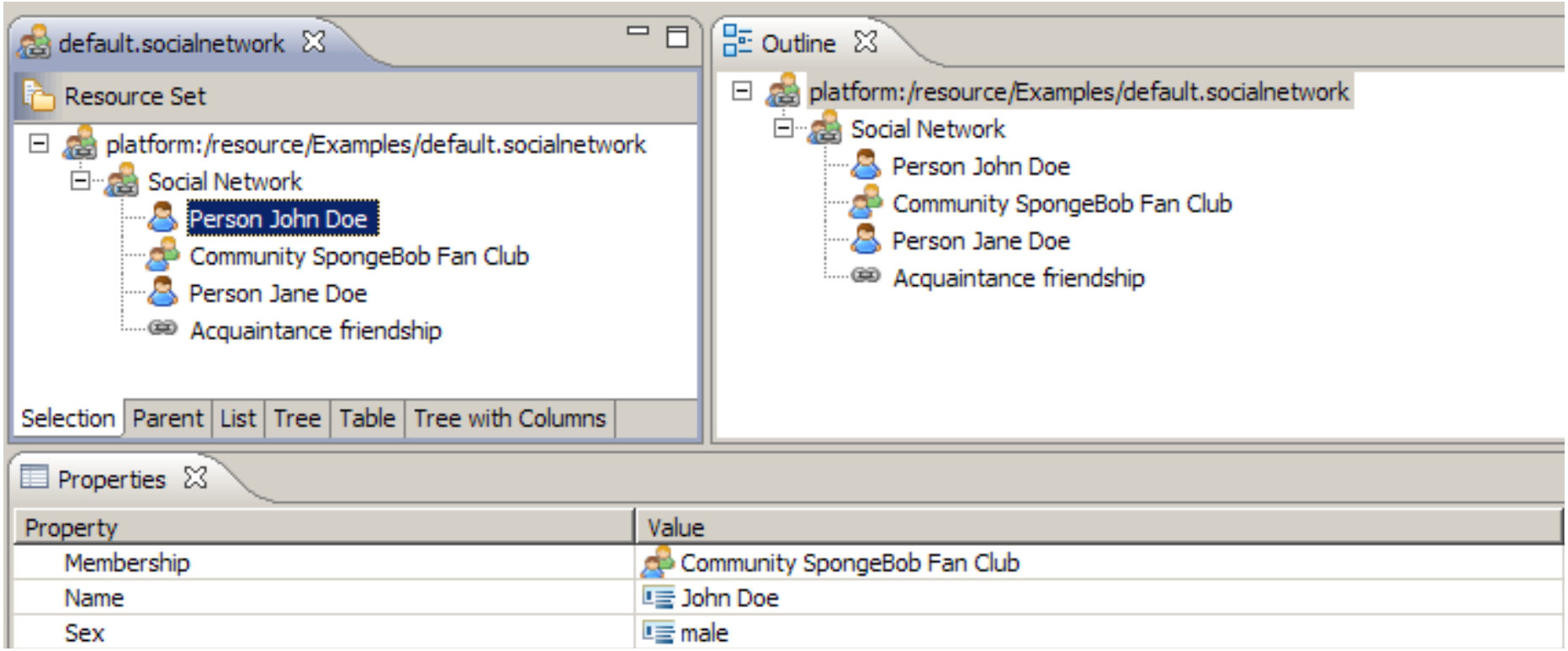
EMF.Model

- Model management layer
- Persistence
- Reflective API

Generated editor

- Editor (based on Eclipse JFace API)
  - Tree-based
  - Provides editing commands
  - Manages workbench settings
- Menus
- Wizard (New model...)
- Plugin activator

Generated editor



EMF.Editor - Problems

- Generated tree editor is only for testing
  - Tree structure is hard to understand
  - Only basic operations are supported
- Alternatives
  - GMF, Graphiti (, Sirius): graphical editors for EMF models
  - EMFText, Xtext: textual editors for EMF models

Extension: Xcore

Xcore: New metamodeling support

- Textual metamodel description language
  - Known shortcomings of Ecore models handled
  - Derived attributes and operations
    - Method body included
- Mostly compatible with existing tools
  - Common runtime
  - Can be generated
- Details:
  - http://wiki.eclipse.org/Xcore

Xcore

*Library.xcore

```
package org.example.library

class Library
{
    String name
    contains Book[] books opposite library
    contains Writer[] authors opposite library
}

class Book
{
    String title
    int pages
    container Library library opposite books
    refers Writer[] authors
}

class Writer
{
    String name
    container Library library opposite authors
    refers Book[] books opposite
}
```

authors

Ctrl+Space to show shortest proposals

Kép forrása: http://wiki.eclipse.org/Xcore

Xcore

```
class Library
{
    String name
    contains Book[] books opposite library
    contains Writer[] authors opposite library
    op Book getBook(String title)
    {
        for (Book book : books)
        {
            if (title == book.title) return book
        }
        return null
    }
}

class Book

class Writer
```

EMF - Summary

EMF

- General metamodeling framework
  - Multiple input types
  - Serialization and editing support
- Code generation
  - Customizable
  - Sane default setting
- Many users
  - Base for many Eclipse-based projects
  - Well applicable

Related techniques

- Validation
  - Well-formedness constraint validation
- Query (not EMF-IncQuery!)
  - Query execution
- Compare
  - Structural compare (e.g. for version control)

- Teneo
  - Persists EMF models into a database
- CDO
  - Distributed, client-server EMF model handling