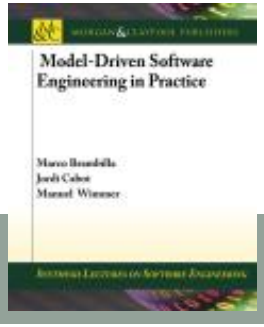
TEXTUAL CONCRETE SYNTAX



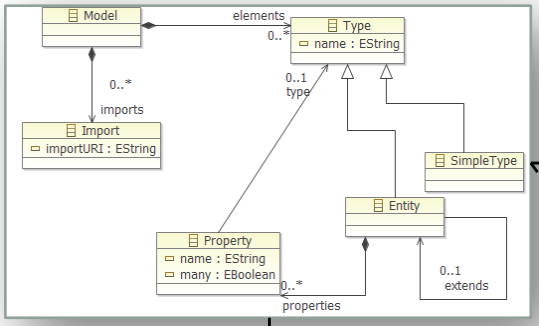
Textual Modeling Languages

- EBNF vs Ecore
- Xtext
- Bridging Xtext grammars and Ecore metamodels

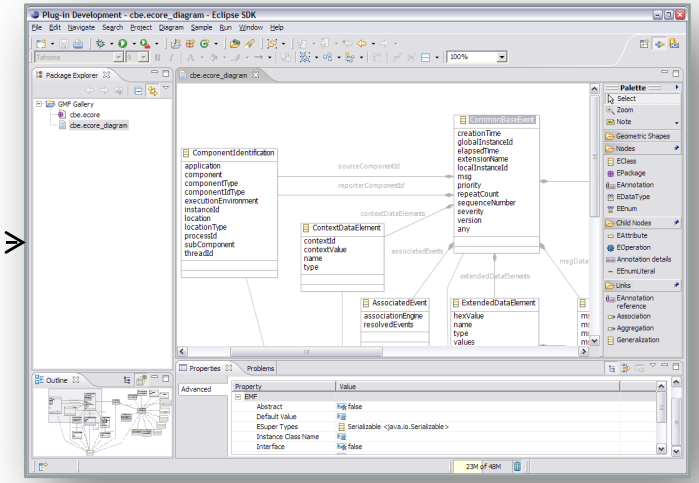
Textual Concrete Syntax

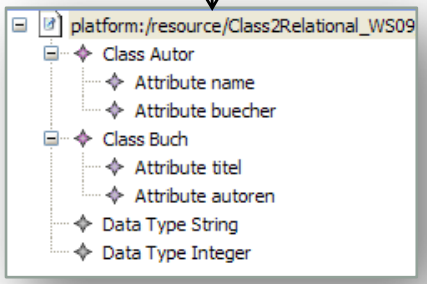
Concrete Syntaxes in Eclipse

Ecore-based Metamodels



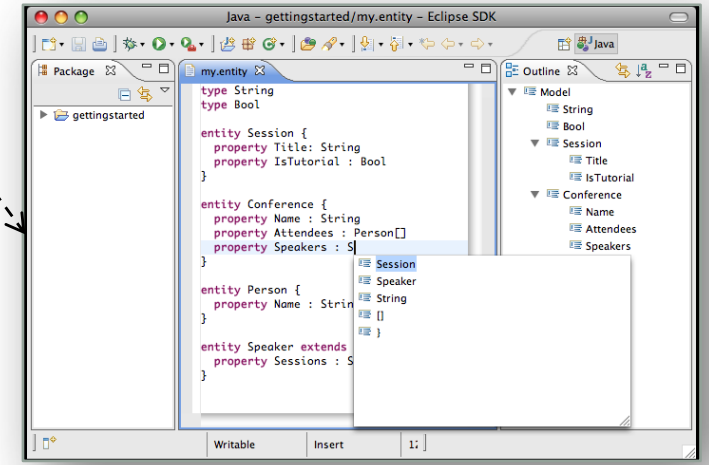
Graphical Concrete Syntax



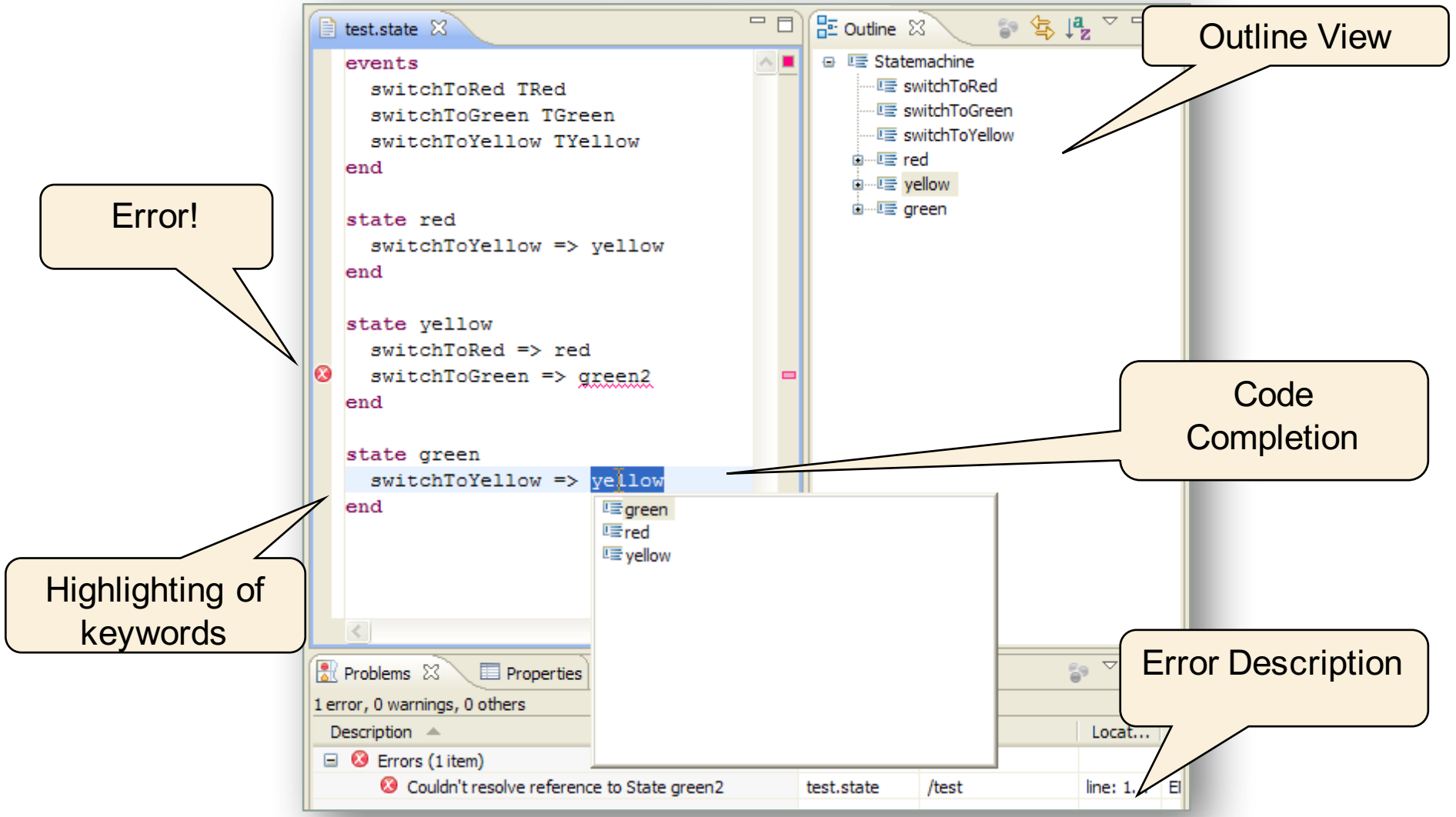


Generic tree-based EMF Editor

Textual Concrete Syntax



Anatomy of Modern Text Editors



Textual Languages

Entity DSL

- **Example**

```
type String
type Boolean

entity Conference {
  property name : String
  property attendees : Person[]
  property speakers : Speaker[]
}

entity Person {
  property name : String
}

entity Speaker extends Person {
        …
}
```


Textual Languages

Entity DSL

- **Sequence analysis**

```
type String
type Boolean

entity Conference {
  property name : String
  property attendees : Person[]
  property speakers : Speaker[]
}

entity Person {
  property name : String
}

entity Speaker extends Person {

}
```

Legend:

- Keywords
- Scope borders
- Separation characters
- Reference
- Arbitrary character sequences

Textual Languages

Entity DSL

- **EBNF Grammar**

```
Model := { Type };

Type := SimpleType | Entity;

SimpleType := 'type', ID;

Entity := 'entity', ID, ['extends', ID], '{', {Property}, '}';

Property := 'property', ID, ':', ID, ['[]'];

ID := ('a'..'z'|'A'..'Z'|'_'),{('a'..'z'|'A'..'Z'|'_'|'0'..'9')};
```

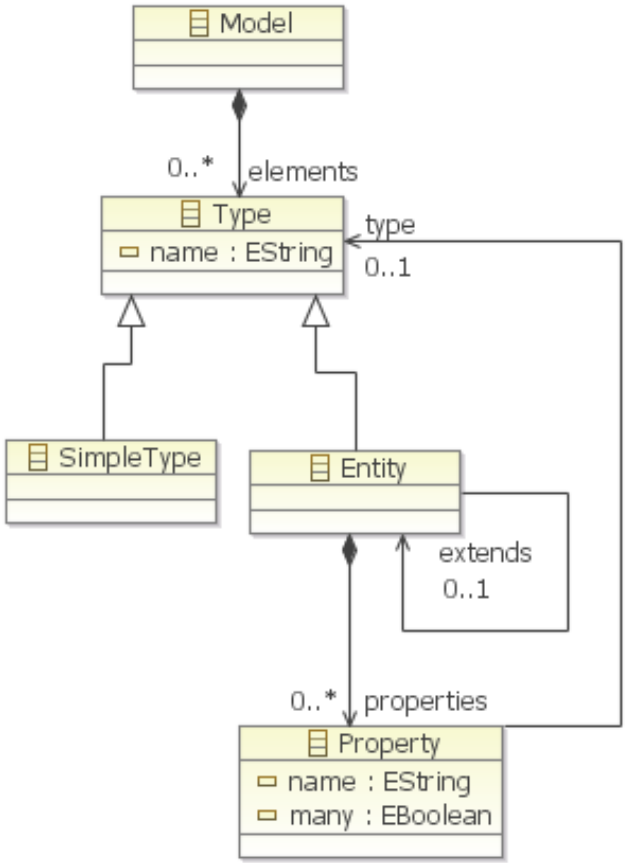
```
type String
type Boolean

entity Conference {
  property name : String
  property attendees : Person[]
  property speakers : Speaker[]
}

entity Person {
  property name : String
}

entity Speaker extends Person {

}
```


Textual Languages

Entity DSL

- **EBNF vs. Ecore**

```
Model := { Type };

Type := SimpleType | Entity;

SimpleType := 'type' ID;

Entity := 'entity' ID ['extends' ID]
   '{' {Property} '}';

Property := 'property' ID ':' ID ['[]'];

ID := ('a'..'z'|'A'..'Z'|'_')
   {('a'..'z'|'A'..'Z'|'_'|'0'..'9')};
```



University of Leicester

EBNF vs Ecore

Similarities

- Grammar
  - Non-terminals
  - Terminals
- Sentence (AST)
- EBNF
  - Definition of grammars
  - Auto-definable
- Compiler

- Metamodel
  - Object types
  - Data types
- Model representation
- MOF
  - Definition of metamodels
  - Auto-definable
- Code generation/model transformation

Differences

- Concrete syntax
- Only recursive structures
- Partial specification of abstract syntax: names
- Parser generation

- Abstract syntax: no literals
- References
- XMI persistence and model-driven development technology
- Code generation

Textual Languages

Solutions to map grammars and metamodels

Generic Syntax

- Like XML for serializing models
- Advantage: Metamodel is sufficient, i.e., no concrete syntax definition is needed
- Disadvantage: no syntactic sugar!
- Protagonists: *HUTN* and *XMI* (OMG Standards)

Language-specific Syntax

- *Metamodel First!*
  - Step 1: Specify metamodel
  - Step 2: Specify textual syntax
  - For instance: *TCS* (Eclipse Plug-in)
- *Grammar First!*
  - Step 1: Syntax is specified by a grammar (concrete syntax & abstract syntax)
  - Step 2: Metamodel is derived from output of step 1, i.e., the grammar
  - For instance: *Xtext* (Eclipse Plug-in)
    - Alternative process: take a metamodel and transform it to an intial Xtext grammar!

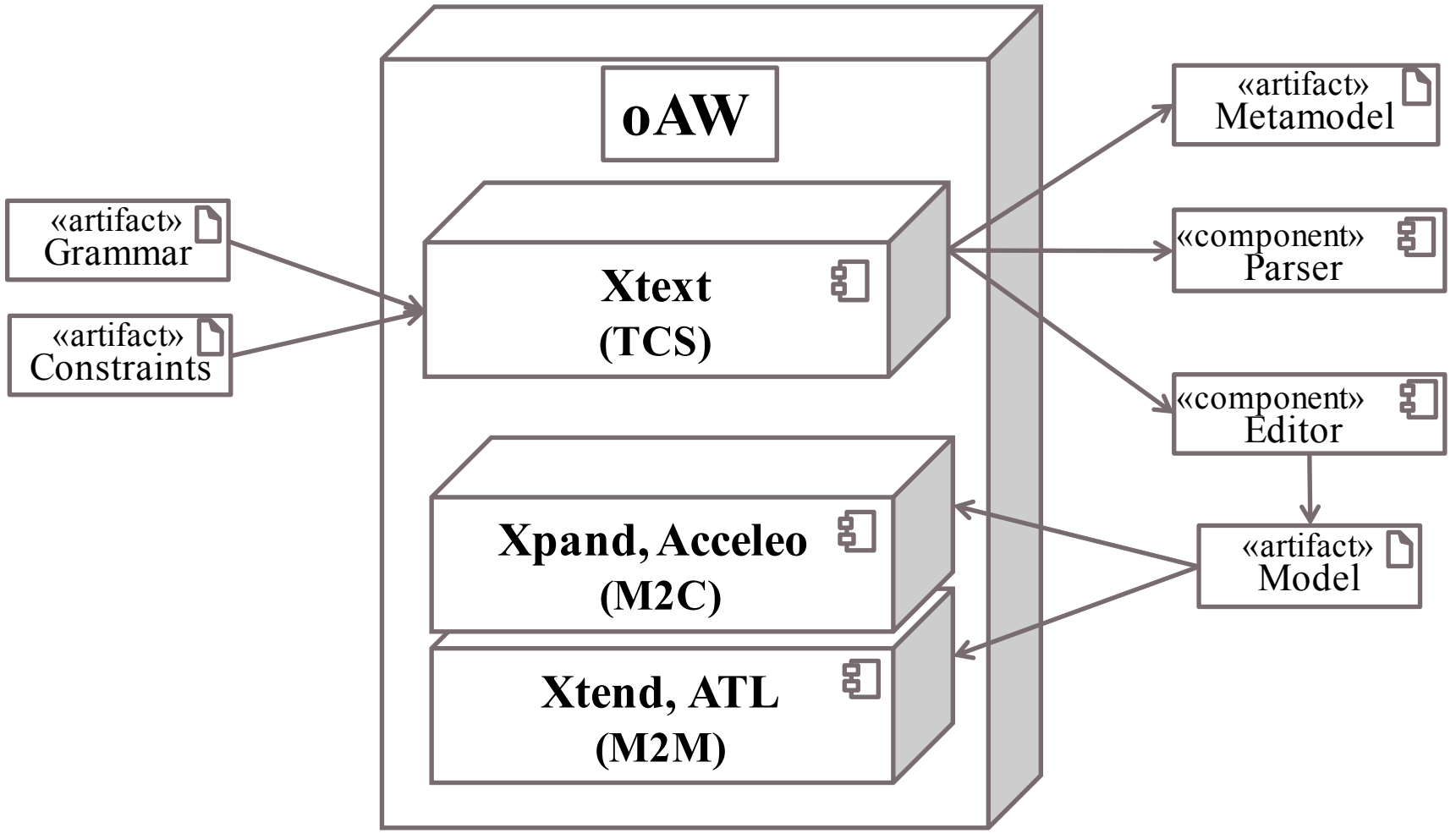
Xtext

Introduction

- **Xtext** is used for developing **textual domain specific languages**
- **Grammar** definition similar to **EBNF**, but with **additional features** inspired by **metamodeling**
- Creates **metamodel**, **parser**, and **editor** from grammar definition
- Editor supports **syntax check**, **highlighting**, and **code completion**
- **Context-sensitive constraints** on the grammar described in OCL-like language

Xtext

Introduction



Xtext

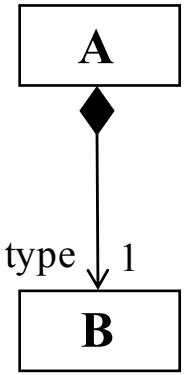
Grammar

- **Xtext** grammar **similar** to **EBNF**

- But **extended** by
  - Object-oriented concepts
  - Information necessary to derive metamodels and modeling editors

- **Example**

```
A: (type=B);
```



Xtext

Grammar

- **Terminal rules**
  - Similar to EBNF rules
  - Return value is String by default

- **EBNF expressions**
  - Cardinalities
    - `?` = One or none; `*` = Any; `+` = One or more
  - Character Ranges `'0'..'9'`
  - Wildcard `'f'.'o'`
  - Until Token `'/*' -> '*/'`
  - Negated Token `'#' (!'#')* '#'`

- **Predefined rules**
  - ID, String, Int, URI

Xtext

Grammar

- **Examples**

```
terminal ID :
    ('^')?('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')*;


terminal INT returns ecore::EInt :
    ('0'..'9')+;


terminal ML_COMMENT :
    '/*' -> '*/';
```


Xtext

Grammar

- **Type rules**
  - For each type rule a **class** is generated in the metamodel
  - Class name corresponds to rule name
- **Type rules contain**
  - Terminals -> *Keywords*
  - Assignments -> *Attributes or containment references*
  - Cross References -> *NonContainment references*
  - …
- **Assignment Operators**
  - = for features with multiplicity 0..1
  - += for features with multiplicity 0..*
  - ?= for Boolean features

Examples

- Assignment

```
State :
  'state' name=ID
    (transitions+=Transition)*
  'end';
```

- Cross References

```
Transition :
  event=[Event] '=>' state=[State];
```


Xtext

Grammar

- **Enum rules**
  - Map Strings to enumeration literals

- **Examples**

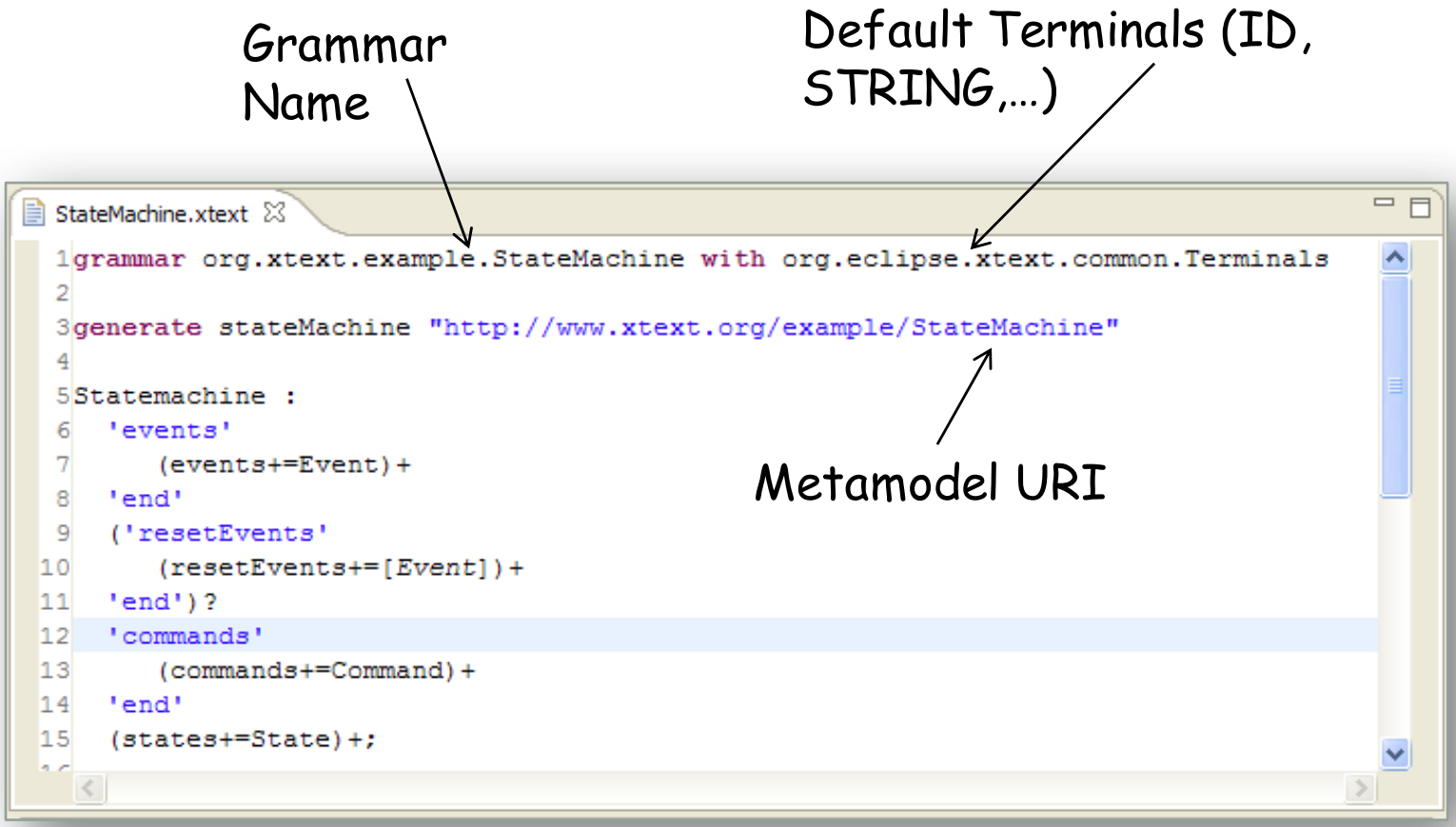
```
enum ChangeKind :
    ADD | MOVE | REMOVE
  ;
```

```
enum ChangeKind :
    ADD = 'add' | ADD = '+' |
    MOVE = 'move' | MOVE = '->' |
    REMOVE = 'remove' | REMOVE = '-'
  ;
```


Xtext

Tooling

- **Xtext Grammar Definition**



Xtext

Tooling

- **Xtext Grammar Definition for State Machines**

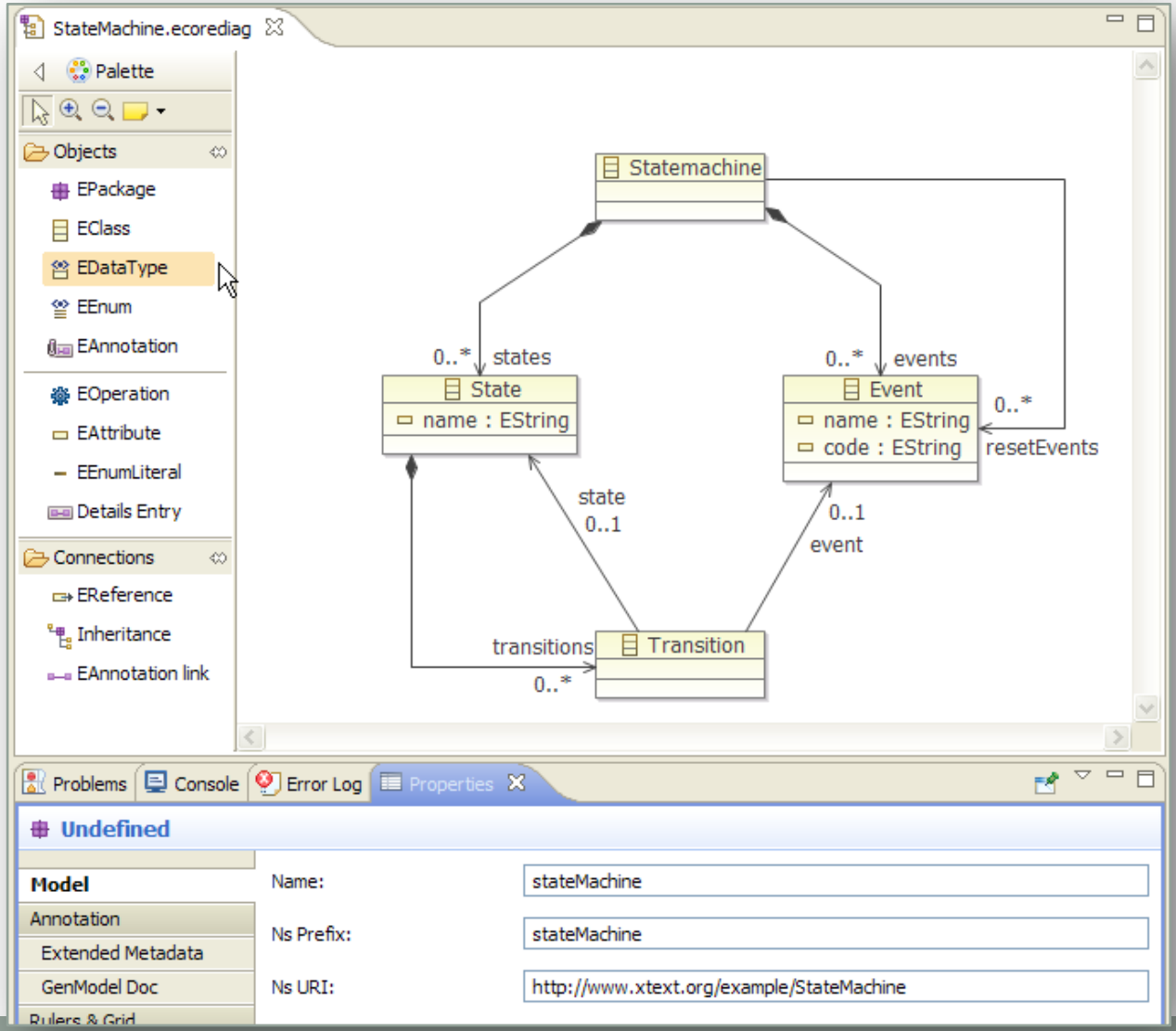
StateMachine.xtext

```
1  grammar org.xtext.example.StateMachine with org.eclipse.xtext.common.Terminals
2
3  generate stateMachine "http://www.xtext.org/example/StateMachine"
4
5  Statemachine :
6    'events'
7       (events+=Event)+
8    'end'
9    ('resetEvents'
10      (resetEvents+=[Event])+
11   'end')?
12   (states+=State)+;
13
14 Event :
15   name=ID code=ID;
16
17 State :
18   'state' name=ID
19      (transitions+=Transition)*
20   'end';
21
22 Transition :
23   event=[Event] '=>' state=[State];
24
25
```


Xtext

Tooling

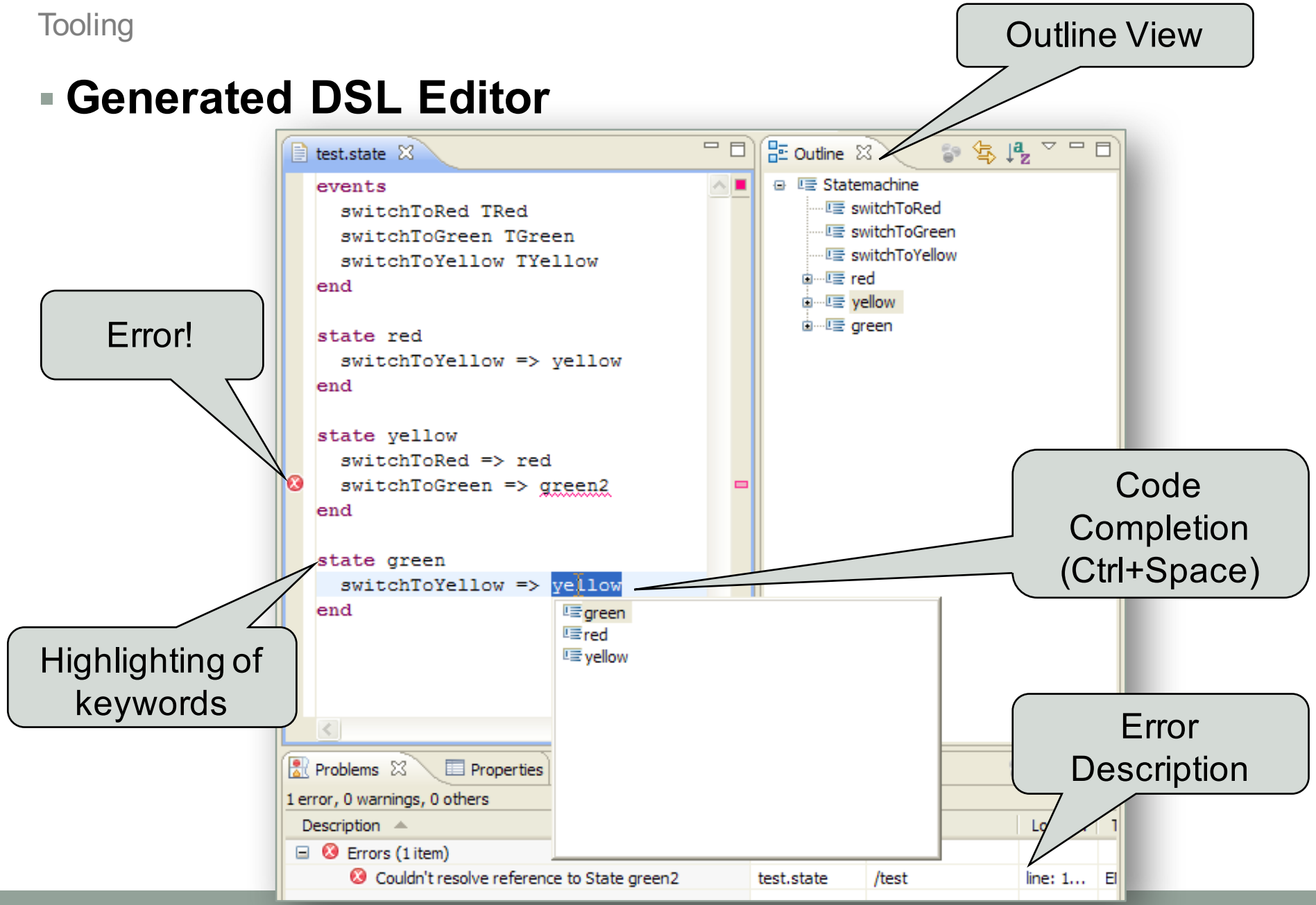
- **Automatically generated Ecore-based Metamodel**



Xtext

Tooling

- **Generated DSL Editor**



Example #1: Entity DSL

Entity DSL Revisited

Example Model

```
type String
type Bool

entity Conference {
  property name : String
  property attendees : Person[]
  property speakers : Speaker[]
}

entity Person {
  property name : String
}

entity Speaker extends Person {
        …
}
```

EBNF Grammar

```
Model := { Type };

Type := SimpleType | Entity;

SimpleType := 'type' ID;

Entity := 'entity' ID ['extends' ID]
          '{' {Property} '}';

Property := 'property' ID ':' ID ['[]'];

ID := ('a'..'z'|'A'..'Z'|'_')
      {('a'..'z'|'A'..'Z'|'_'|'0'..'9')};
```


Example #1

From EBNF to Xtext

EBNF Grammar

```
Model := { Type };

Type := SimpleType | Entity;

SimpleType := 'type' ID;

Entity := 'entity' ID ['extends' ID]
          '{' {Property} '}';

Property := 'property' ID ':' ID ['[]'];

ID := ('a'..'z'|'A'..'Z'|'_')
      {('a'..'z'|'A'..'Z'|'_'|'0'..'9')};
```

Xtext Grammar

```
grammar MyDsl with
org.eclipse.xtext.common.Terminals

generate myDsl "http://MyDsl"

Model : (elements+=Type)*;

Type: SimpleType | Entity;

SimpleType: 'type' name=ID;

Entity : 'entity' name=ID
  ('extends' extends=[Entity])? '{'
  properties+=Property*
'}';

Property: 'property' name=ID ':'
  type=[Type] (many?='[]')?;
```


Example #1

How to specify context sensitive constraints for textual DSLs?

- **Examples**
  - Entity names must start with an Upper Case character
  - Entity names must be unique
  - Property names must be unique within one entity
- **Answer**
  - Use the same techniques as for metamodels!

Xtext Grammar

```
grammar MyDsl with
org.eclipse.xtext.common.Terminals

generate myDsl "http://MyDsl"

Model : elements+=Type*;

Type: SimpleType | Entity;

SimpleType: 'type' name=ID;

Entity : 'entity' name=ID
  ('extends' extends=[Entity])? '{'
  properties+=Property*
'}';

Property: 'property' name=ID ':'
  type=[Type] (many?='[]')?;
```


Example #1

How to specify context sensitive constraints for textual DSLs?

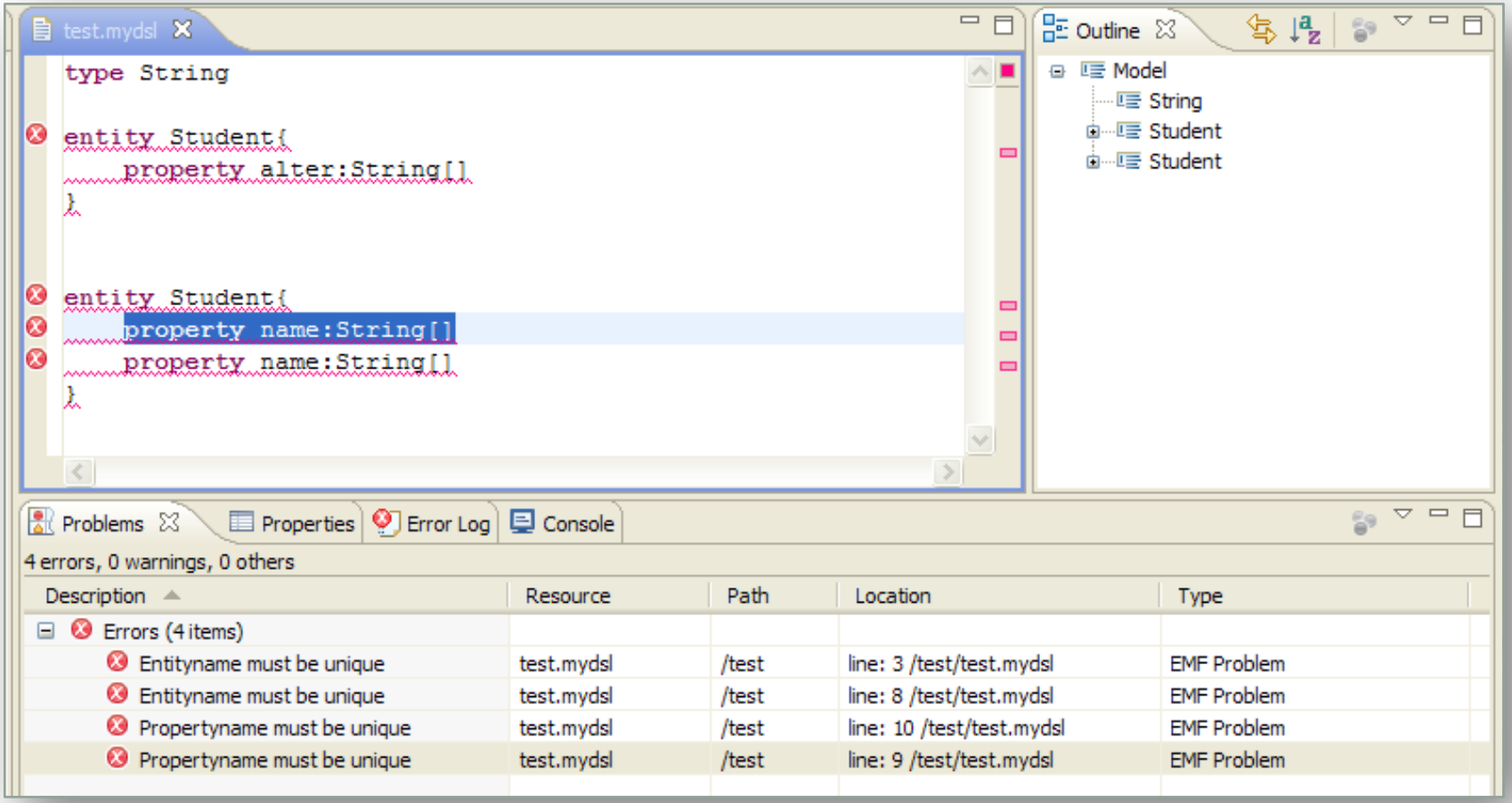
- Examples
  1. Entity names must start with an Upper Case character
  2. Entity names must be unique within one model
  3. Property names must be unique within one entity

- Solution shown in Check language (similar to OCL)
  1. **context** myDsl::Entity
     WARNING "Name should start with a capital":
     name.toFirstUpper() == name;
  2. **context** myDsl::Entity
     ERROR "Name must be unique":
     ((Model)this.eContainer).elements.name.
     select(e|e == this.name).size == 1;
  3. **context** myDsl::Property
     ERROR "Name must be unique":
     ((Entity)this.eContainer).properties.name.
     select(p|p == this.name).size == 1;

Example #1

When to evaluate context sensitive constraints?

- Every edit operation for cheap constraints
- Every save operation for cheap to expensive constraints
- Every generation operation for very expensive constraints

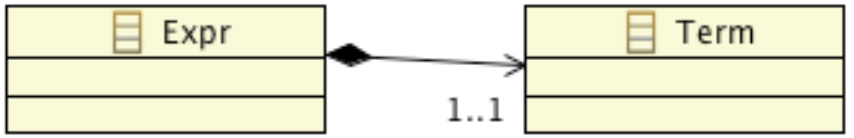


Bridging Xtext Grammars to Metamodels

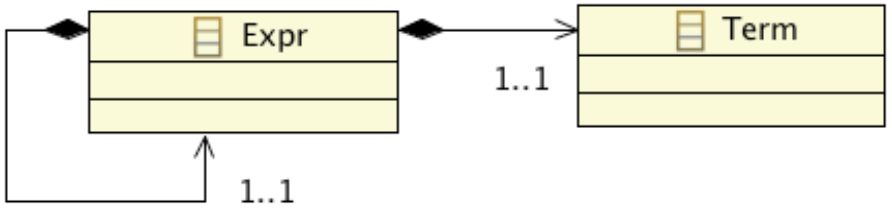
- ✓Production of a metamodel (M2)
  - Mapping a EBNF-based grammar into a MOF-based metamodel
- ✓Optimization of the metamodel
- ✓Production of a program transformer (M1)
  - Mapping a well-formed program into a metamodel-conformant model

EBNF To MOF (1)

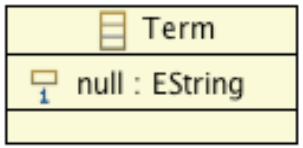
- ✓ Production
  - Both sides are mapped into classes and connected with a composition.
    - *Expr* = *Term* ;



- ✓ Non-terminal
  - Two classes and a association between them.
    - *Expr* = *Expr* , '+' , *Term* ;



- ✓ Terminal defined by a regular expression
  - Attribute
    - *Term* = ID ;

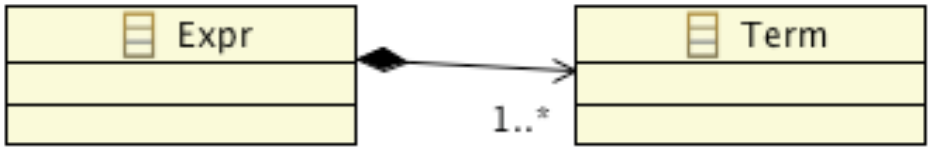


- ✓ Terminals are removed

EBNF To MOF (2)

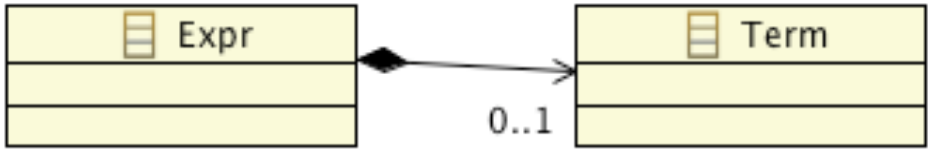
✓Repetition

- Resulting association is one to many
  - *Expr* = *Term* , {+ , *Term*} ;



✓Option

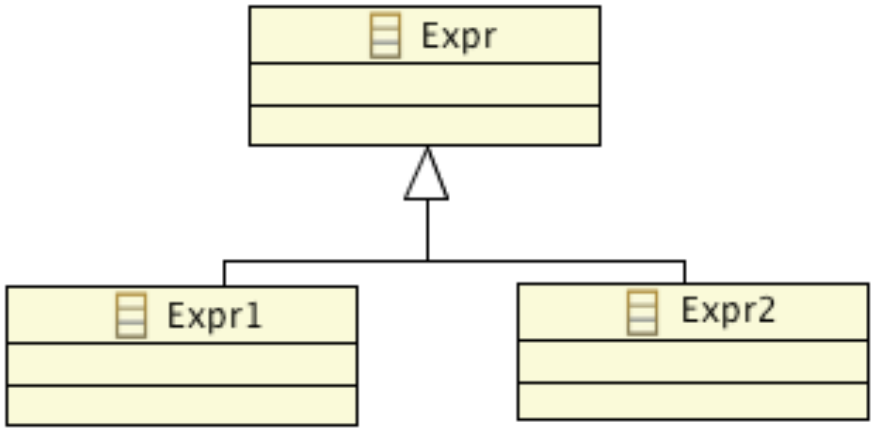
- Resulting association has upper bound one
  - *Expr* = [*Term*] ;



EBNF To MOF (3)

✓Alternative

- Each alternative non-terminal is mapped to a subclass of the class that corresponds to the non-terminal Expr.
  - *Expr = Expr1 | Expr2 ;*



Refactoring of the Metamodel

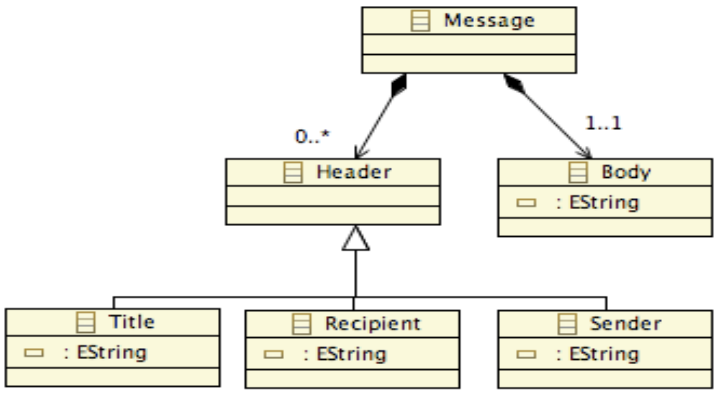
- ✓References
  - Turn containment references into regular references when needed
- ✓Names
  - Add names to structural features
- ✓Pull up structural features
  - Attributes and references should not be duplicated in subclasses of the same class
  - They have to be pulled up in the class hierarchy

Obtaining the Metamodel from the Grammar

send message
to “ab@le.ac.uk”
from “ba@le.ac.uk”
subject “hello”
{
  Hello world!
}

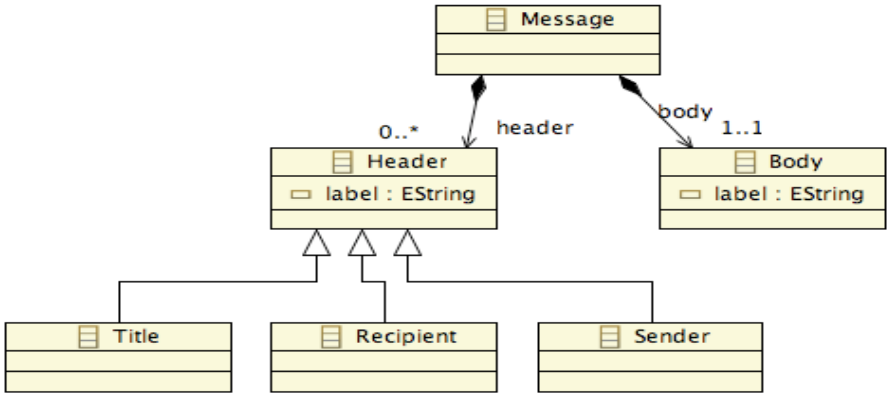
```
Message   = ‘send message’, { Header } ,‘{‘, Body ,‘}’ ;
Header    = Title | Recipient | Sender ;
Recipient = ‘to’, ID ;
Sender    = ‘from’, ID ;
Title     = ‘subject’, ID ;
Body      = ID ;
```

textual representation of a model



generated metamodel

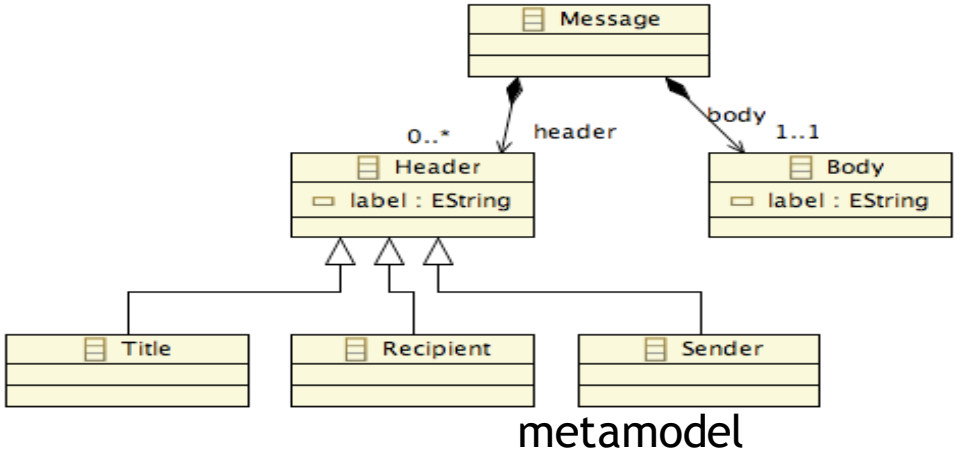
EBNF grammar



optimized metamodel

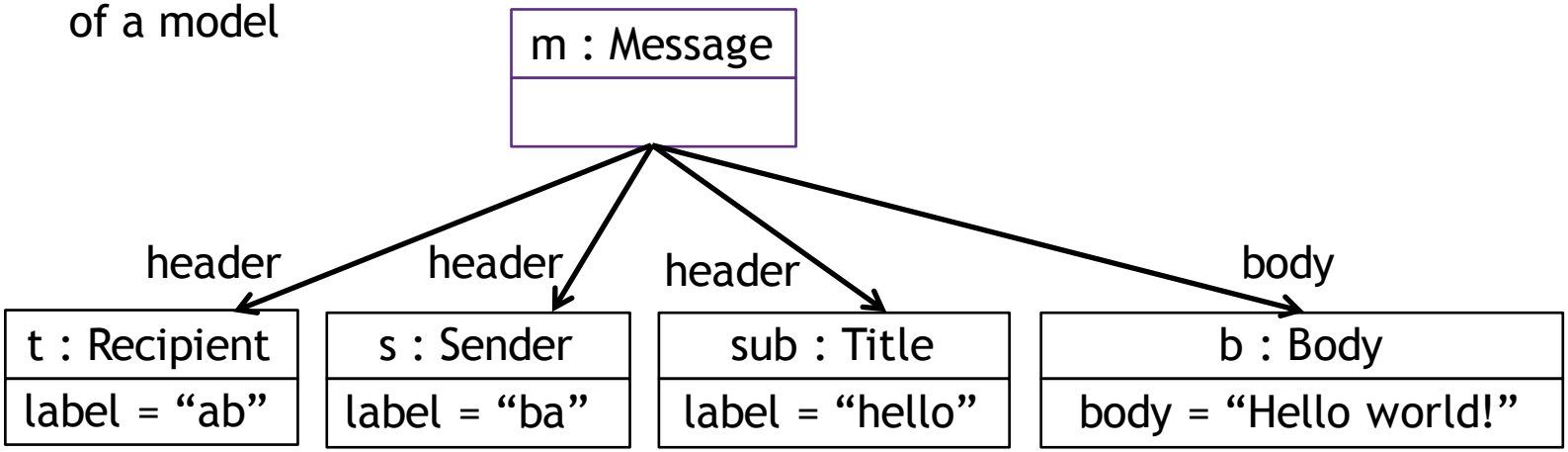
Obtaining the Model from a Program

```
send message
to "ab"
from "ba"
subject "hello"
{
  Hello world!
}
```



metamodel

textual representation of a model



model

Xtext

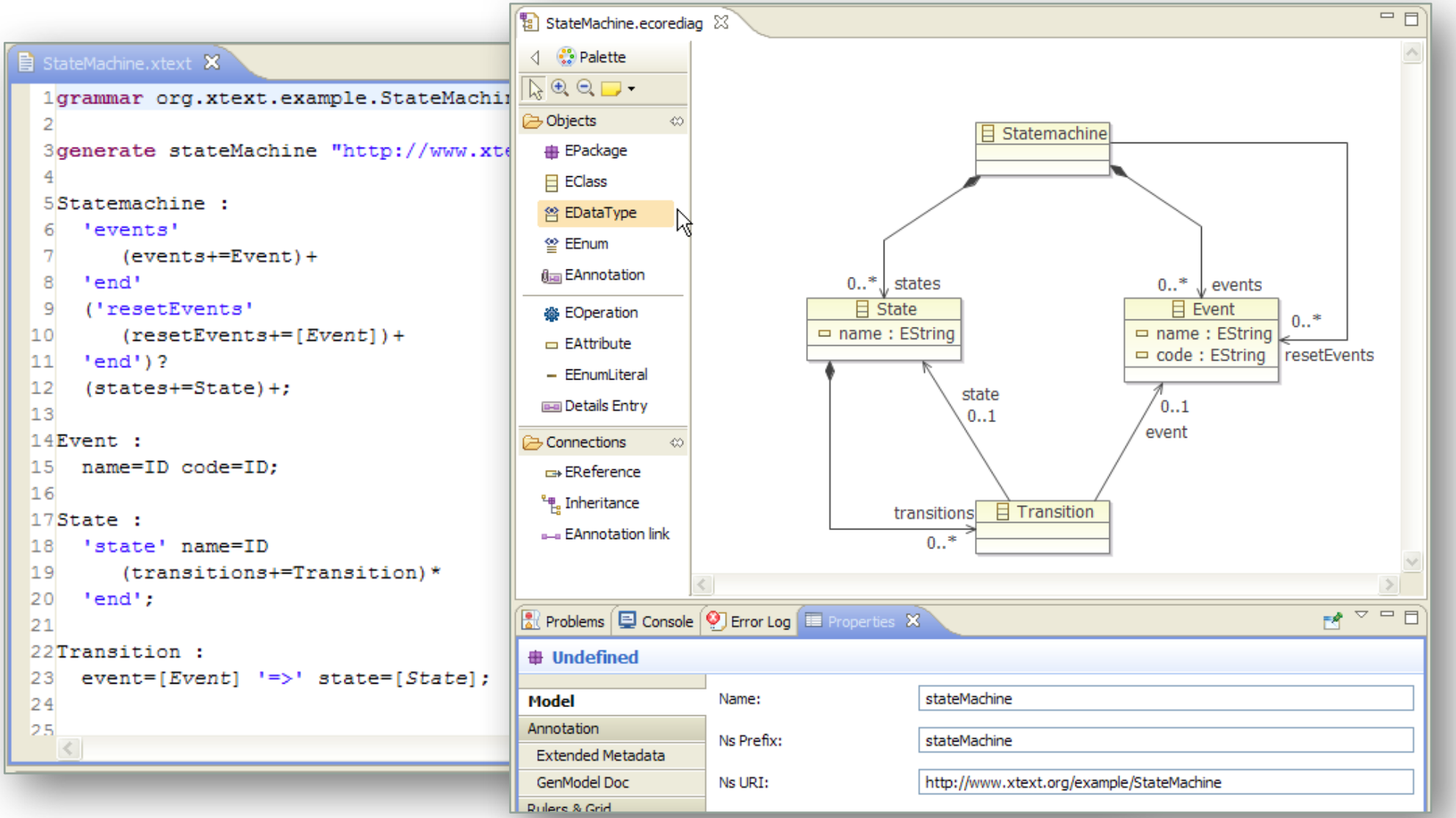
Tooling

- **Xtext Grammar Definition for State Machines**

StateMachine.xtext

```
1  grammar org.xtext.example.StateMachine with org.eclipse.xtext.common.Terminals
2
3  generate stateMachine "http://www.xtext.org/example/StateMachine"
4
5  Statemachine :
6    'events'
7       (events+=Event)+
8    'end'
9    ('resetEvents'
10      (resetEvents+=[Event])+
11   'end')?
12   (states+=State)+;
13
14 Event :
15   name=ID code=ID;
16
17 State :
18   'state' name=ID
19      (transitions+=Transition)*
20   'end';
21
22 Transition :
23   event=[Event] '=>' state=[State];
24
25
```


Example #1: State machines

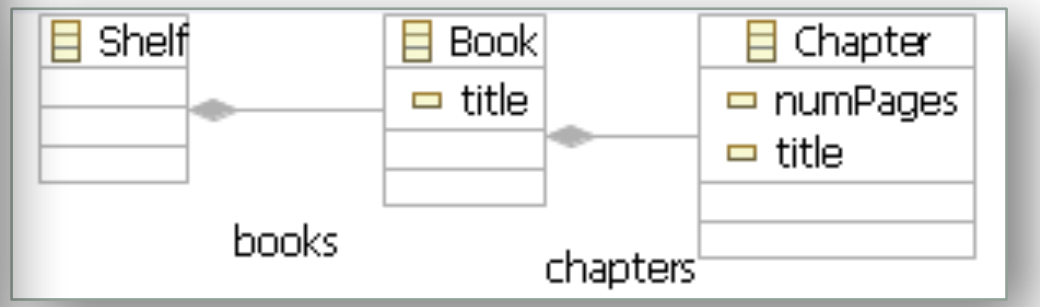


Example #2: Bookshelf (Homework)

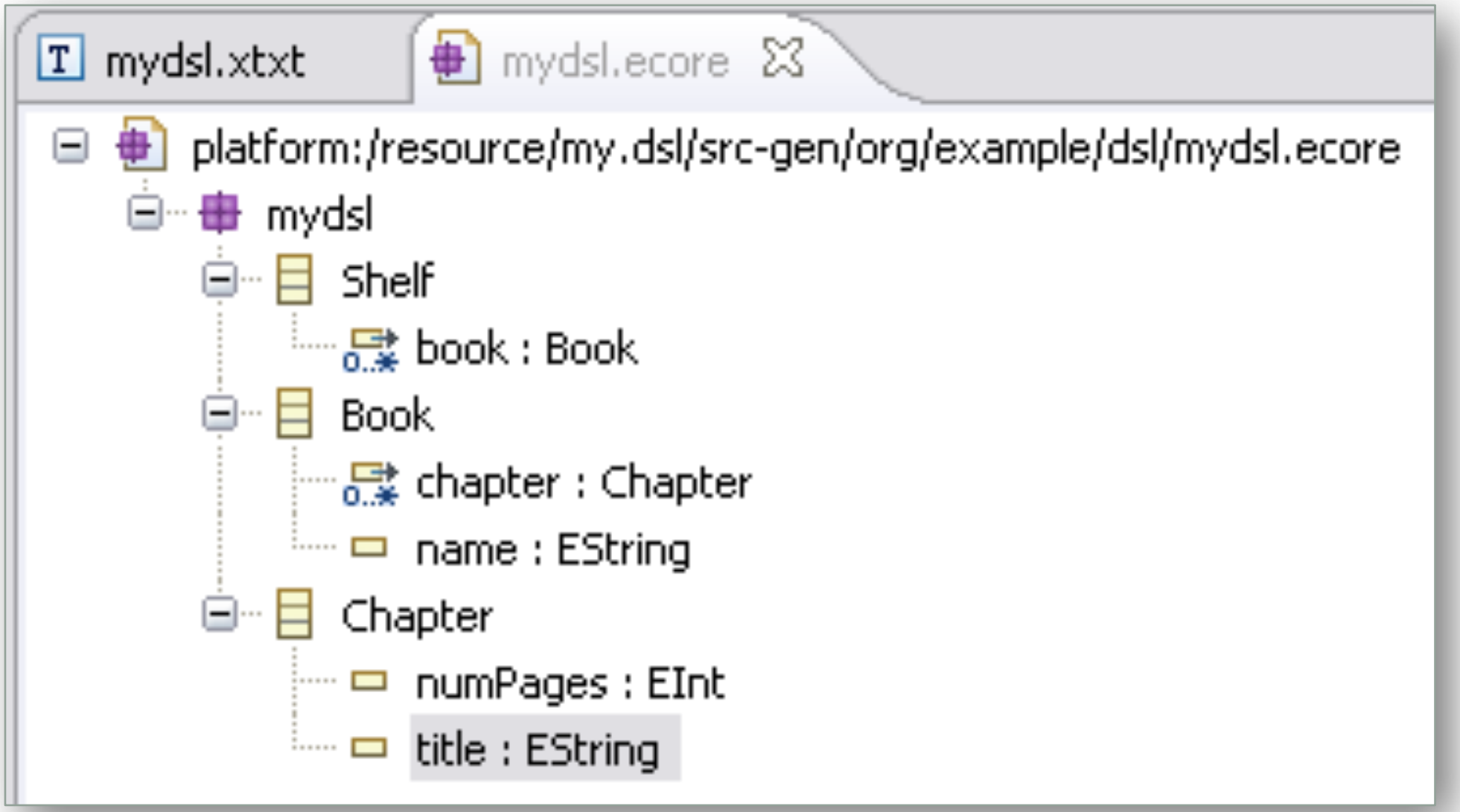
- Edit “Bookshelf” models in a text-based fashion
- **Given**: Example model as well as the metamodel
- **Asked**: Grammar, constraints, and editor for Bookshelf DSL

model.xmi

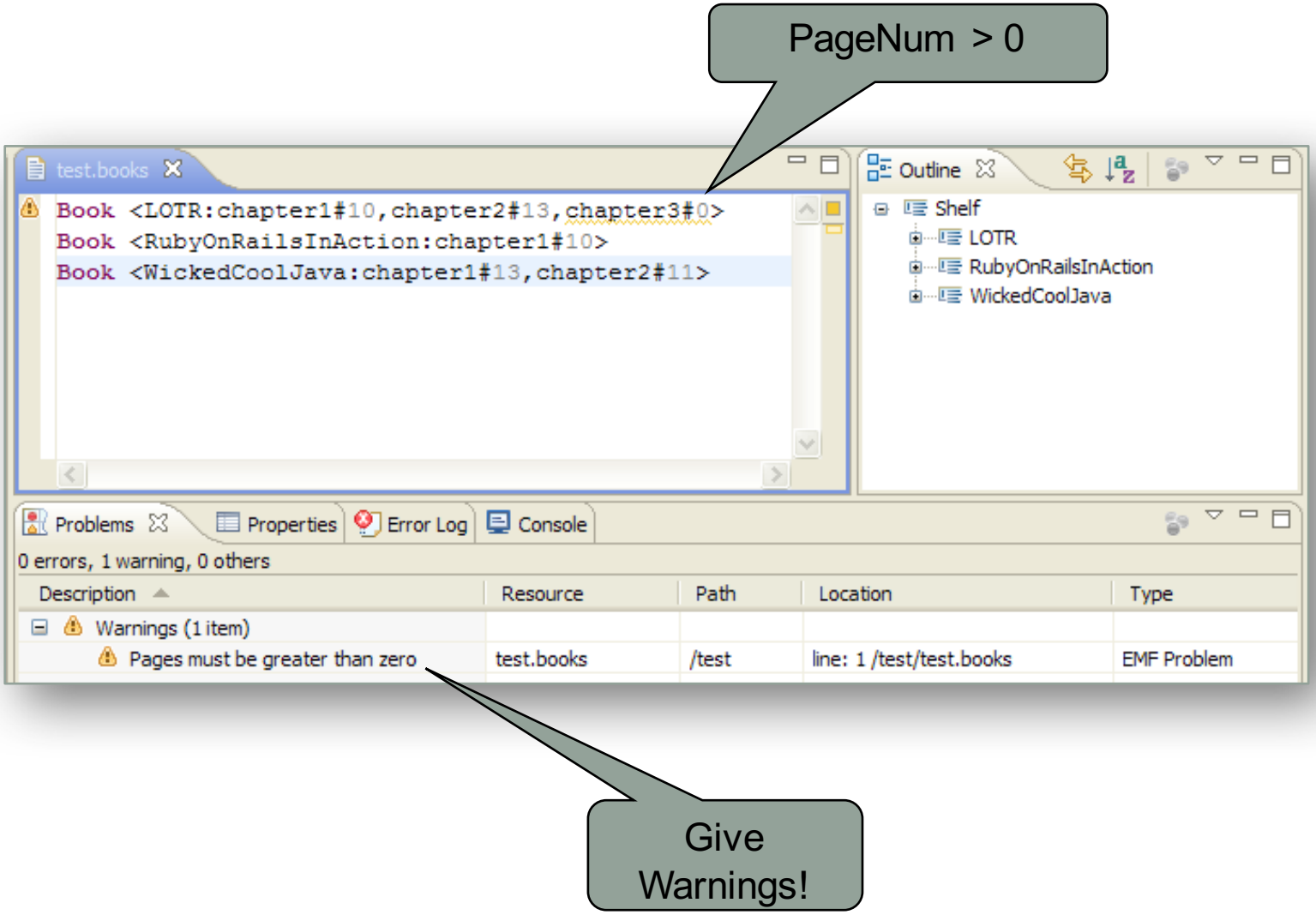
```
1<?xml version="1.0" encoding="ASCII"?>
2<bookshelf:Shelf xmi:version="2.0" xmlns:xmi="http:/
3  <book name="LOTR">
4    <chapter title="chapter1" numPages="10"/>
5    <chapter title="chapter2" numPages="13"/>
6    <chapter title="chapter3" numPages="12"/>
7  </book>
8  <book name="RubyOnRailsInAction">
9    <chapter title="chapter1" numPages="10"/>
10 </book>
11 <book name="WickedCoolJava">
12   <chapter title="chapter1" numPages="13"/>
13   <chapter title="chapter2" numPages="11"/>
14 </book>
15</bookshelf:Shelf>
16
```



Example #2: Metamodel Details



Example #2: Editor



Wrapping up

- Xtext can generate web editors that can be integrated in cloud systems
  - more sophisticated interfaces
- For example, we can model GraphQL syntax in an Xtext and parse GraphQL queries

MORGAN & CLAYPOOL PUBLISHERS

MODEL-DRIVEN SOFTWARE ENGINEERING IN PRACTICE. CHAPTERS 6, 7

Marco Brambilla,
Jordi Cabot,
Manuel Wimmer.
Morgan & Claypool, USA, 2012.

www.mdse-book.com
www.morganclaypool.com