Modellbasierte Softwareentwicklung (MODSOFT)

Part II

Domain Specific Languages

xText

Prof. Joachim Fischer /
Dr. Markus Scheidgen / Dipl.-Inf. Andreas Blunk

{fischer,scheidge,blunk}@informatik.hu-berlin.de

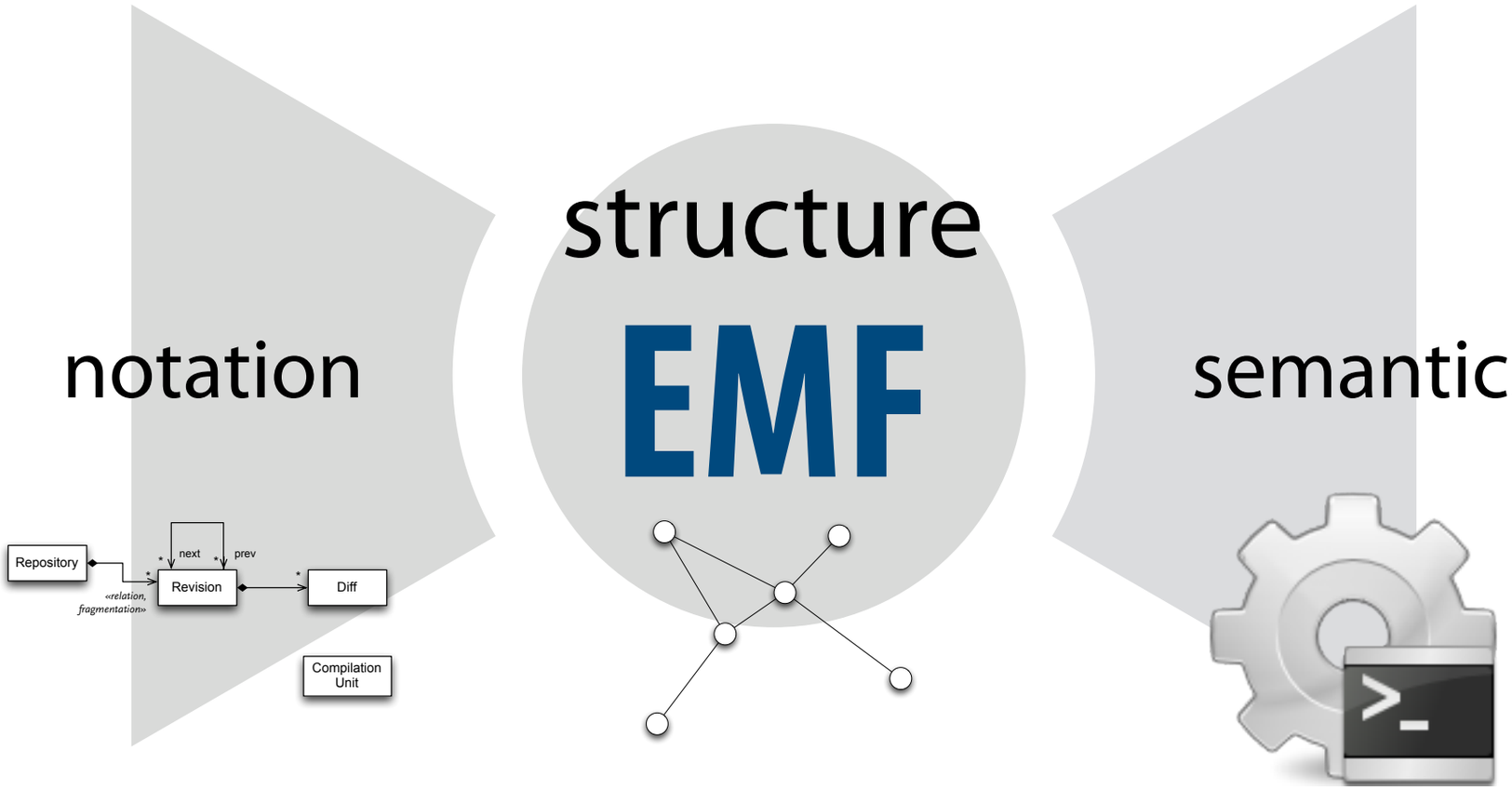
LFE Systemanalyse, III.310

Agenda

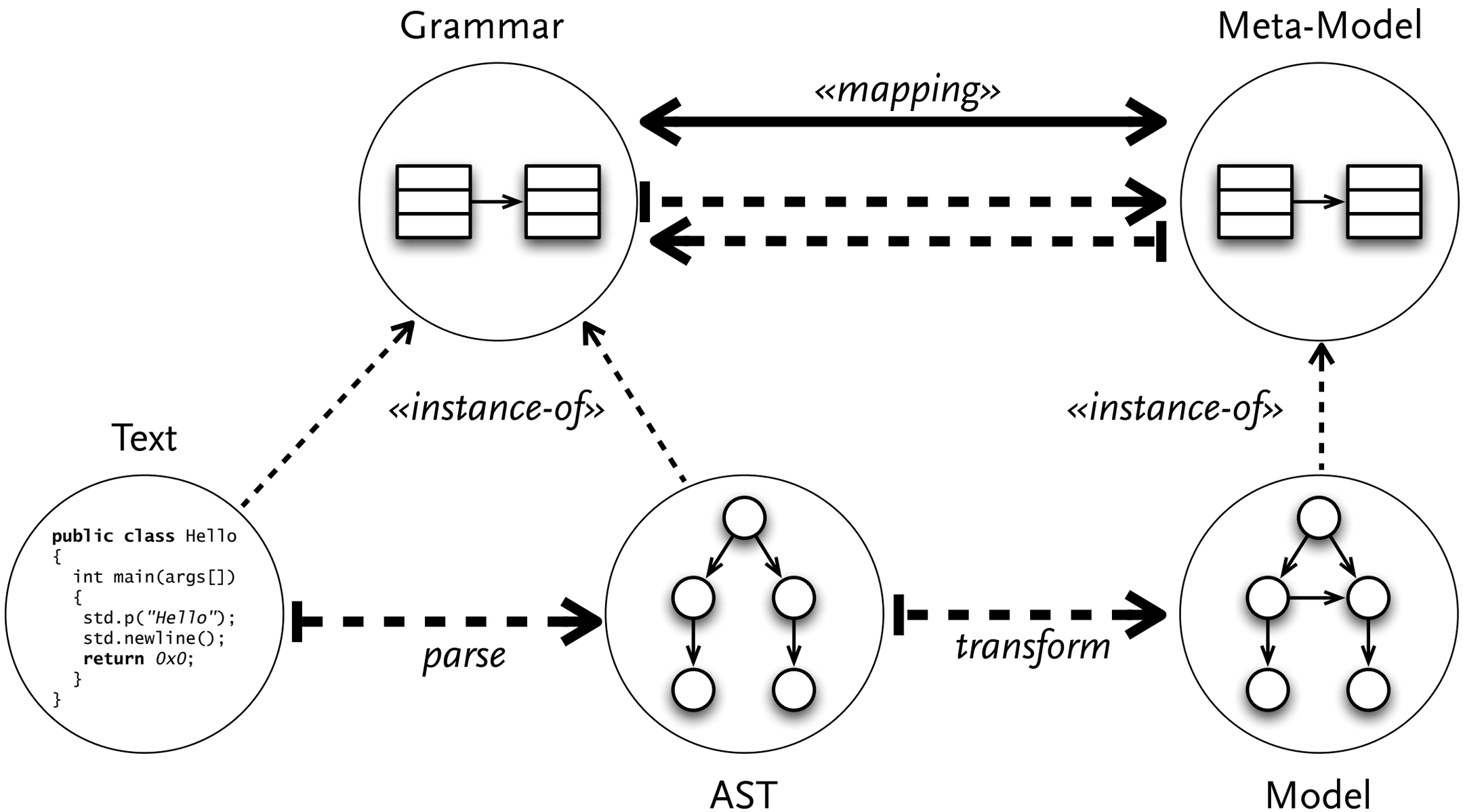
prolog (1 VL)	**Introduction:** languages and their aspects, modeling vs. programming, meta-modeling and the 4 layer model
0. (2 VL)	**Eclipse/Plug-ins:** eclipse, plug-in model and plug-in description, features, *p2*-repositories, *RCP*s
1. (2 VL)	**Structure:** *Ecore*, *genmodel*, working with generated code, constraints with *Java* and *OCL*, *XML/XMI*
➡ 2. (3 VL)	**Notation:** Customizing the tree-editor, textural with *XText*, graphical with *GEF* and *GMF*
3. (4 VL)	**Semantics:** interpreters with Java, code-generation with *Java* and *XTend*, model-transformations with *Java* and *ATL*
epilog (2 VL)	**Tools:** persisting large models, model versioning and comparison, model evolution and co-adaption, modular languages with *XBase*, *Meta Programming System* (MPS)

Previously on MODSOFT

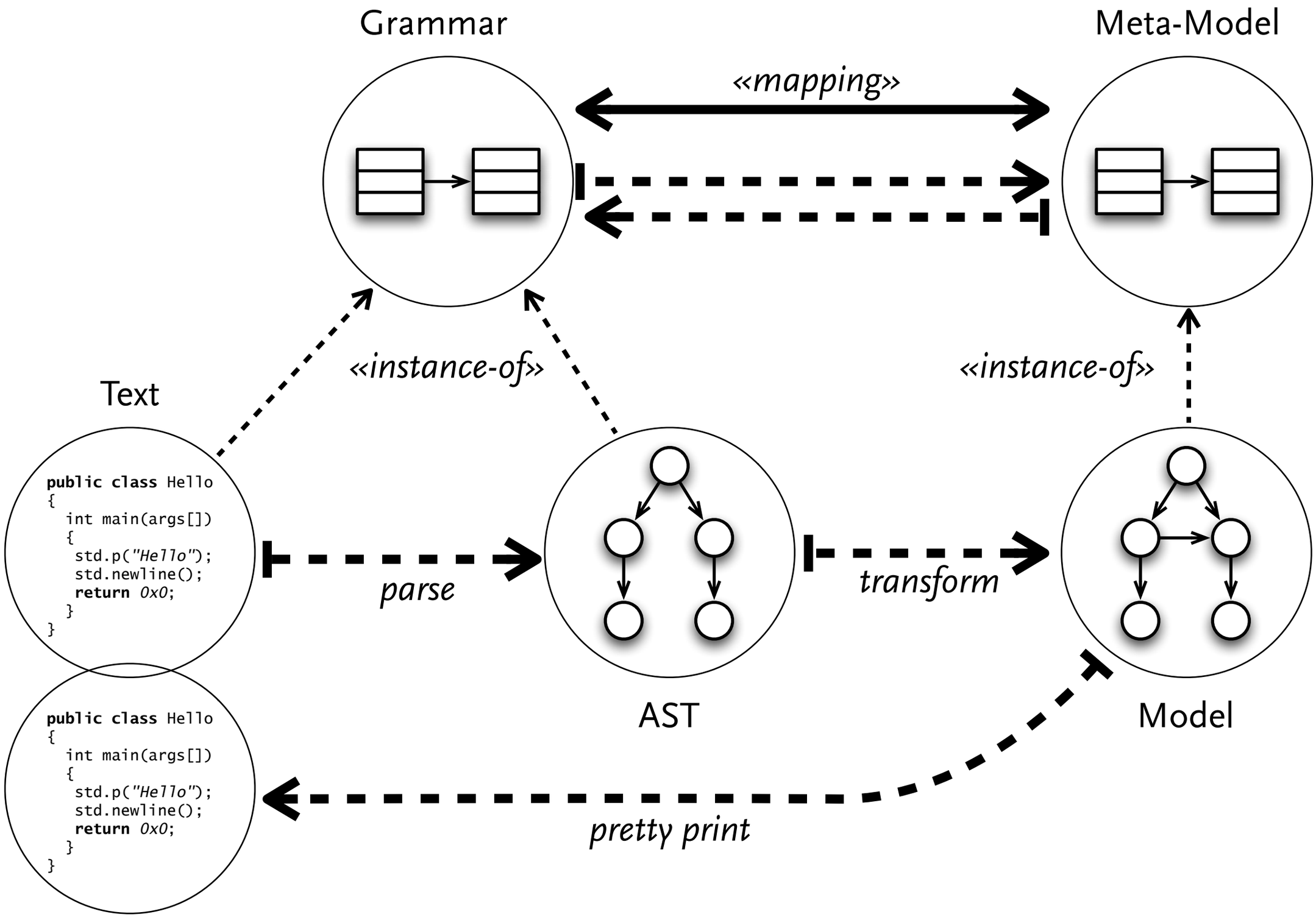
Eclipse Modeling Framework



Parsing (2)


Grammar
Meta-Model
«mapping»
«instance-of»
«instance-of»
Text
public class Hello
{
int main(args[])
{
std.p("Hello");
std.newline();
return 0x0;
}
}
parse
transform
AST
Model

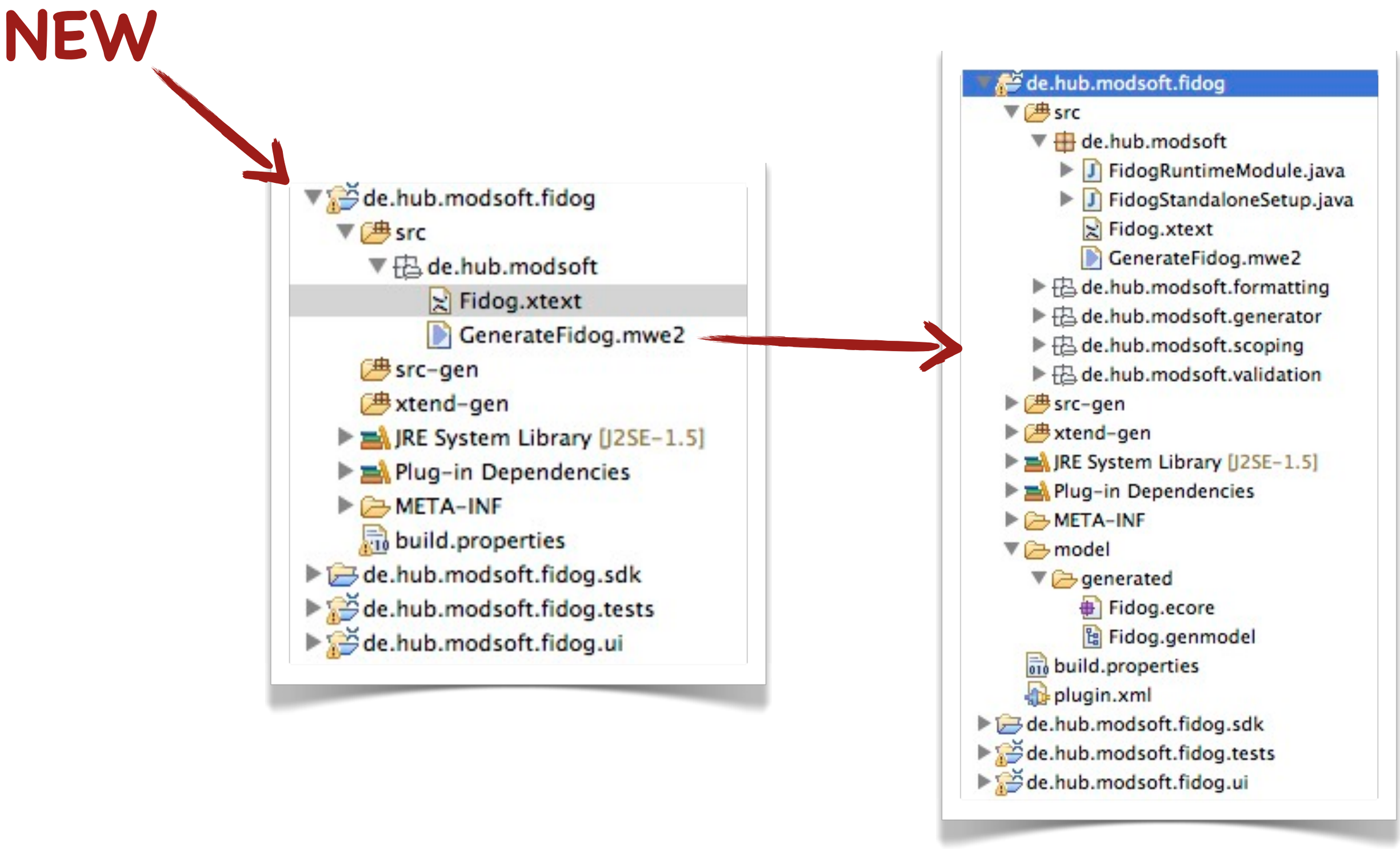
Parsing (2)


Grammar
Meta-Model
«mapping»
«instance-of»
«instance-of»
Text
public class Hello
{
int main(args[])
{
std.p("Hello");
std.newline();
return 0x0;
}
}
parse
AST
transform
Model
public class Hello
{
int main(args[])
{
std.p("Hello");
std.newline();
return 0x0;
}
}
pretty print

xText

xText PDE Projects

NEW



xText – Grammar

xText Grammar

```
grammar de.hub.modsoft.Fidog with org.eclipse.xtext.common.Terminals

generate fidog "http://www.hub.de/modsoft/Fidog"

Model:
    owners+=Owner*
;

Owner:
    name=ID 'has' pets+=Pet ( 'and' pets+=Pet )
;

Pet:
    (Dog | Cat)
    'called' name=STRING '(' weight=INT 'kg' ')'
    (',' 'who' 'is' 'friends' 'with' friends+=[Pet|STRING]
          ('and' friends+=[Pet|STRING])* )?
;

Dog:
    'a' 'dog'
;

Cat:
    'a' 'cat'
;
```

```
grammar de.hub.modsoft.Fidog with org.eclipse.xtext.common.Terminals

generate fidog "http://www.hub.de/modsoft/Fidog"

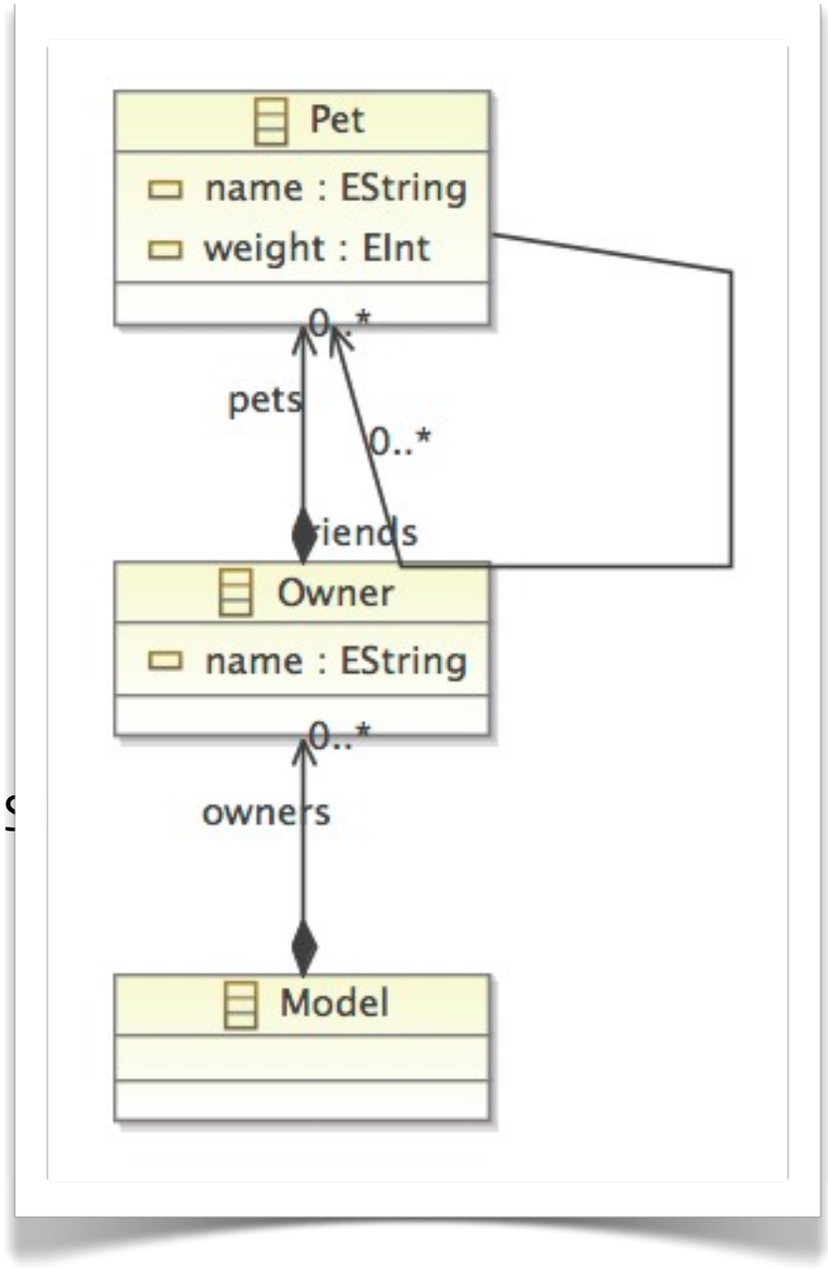
Model:
    owners+=Owner*
;

Owner:
    name=ID 'has' pets+=Pet ( 'and' pets+=Pet )
;

Pet:
    (Dog | Cat)
    'called' name=STRING '(' weight=INT 'kg' ')'
    (',' 'who' 'is' 'friends' 'with' friends+=[Pet|S
          ('and' friends+=[Pet|STRING])* )?
;

Dog:
    'a' 'dog'
;

Cat:
    'a' 'cat'
;
```



```
grammar de.hub.modsoft.Fidog with org.eclipse.xtext.common.Terminals

generate fidog "http://www.hub.de/modsoft/Fidog"

Model:
    owners+=Owner*
;

Owner:
    name=ID 'has' pets+=Pet ( 'and' pets+=Pet )
;

Pet:
    (Dog | Cat)
    'called' name=STRING '(' weight=INT 'kg' ')'
    (',' 'who' 'is' 'friends' 'with' friends+=[Pet|STRING] ('and' friends+=[Pet|
STRING])* )?
;

Dog:
    {Dog}
    'a' 'dog'
;

Cat:
    {Cat}
    'a' 'cat'
;
```

```
grammar de.hub.modsoft.Fidog with org.eclipse.xtext.common.Terminals

generate fidog "http://www.hub.de/modsoft/Fidog"

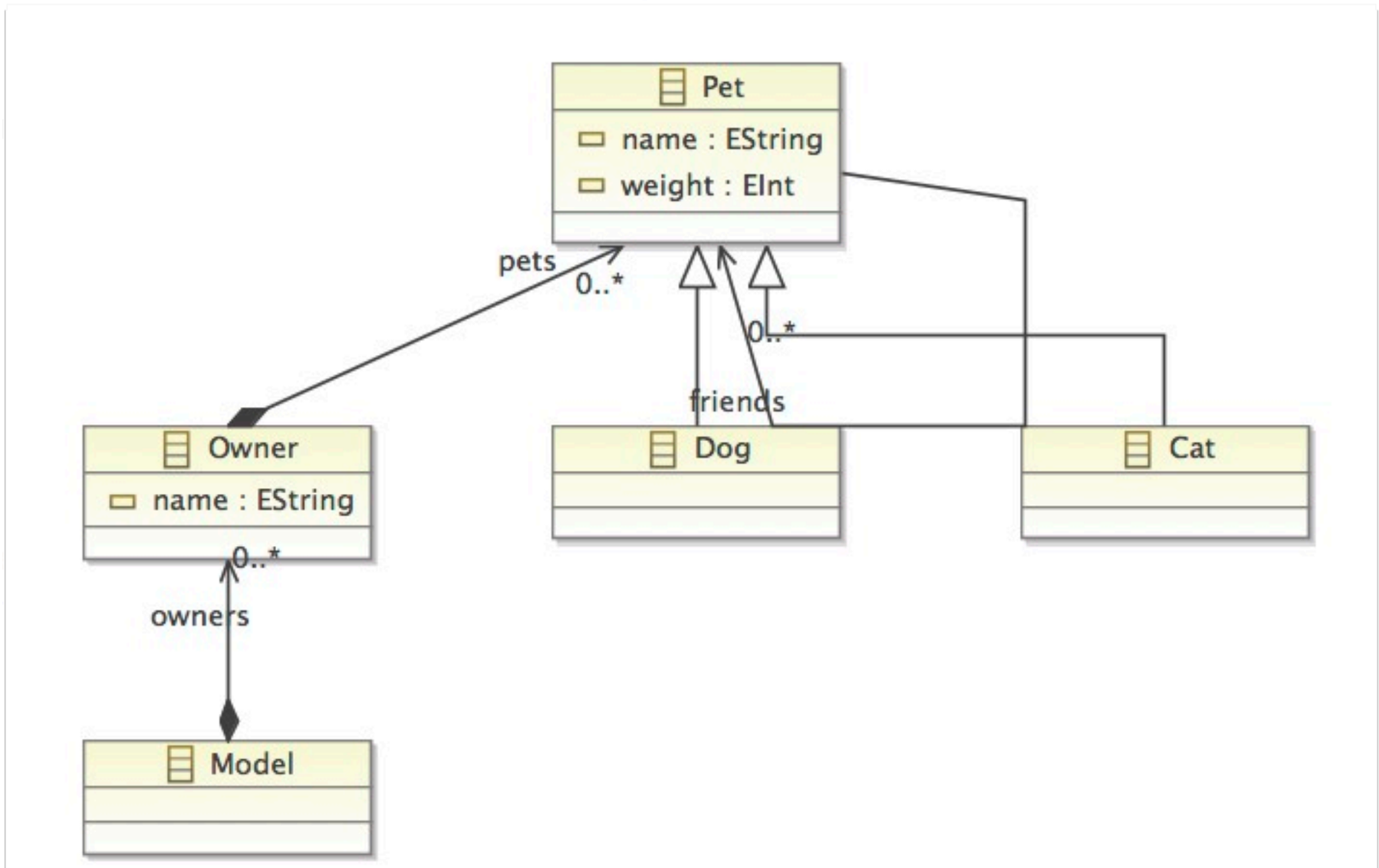
Model:
    owners+=Own
;

Owner:
    name=ID 'ha
;

Pet:
    (Dog | Cat)
    'called' na
    (',' 'who'
STRING])* )?
;

Dog:
    {Dog}
    'a' 'dog'
;

Cat:
    {Cat}
    'a' 'cat'
;
```



```
grammar de.hub.modsoft.Fidog with org.eclipse.xtext.common.Terminals

generate fidog "http://www.hub.de/modsoft/Fidog"

Model:
    owners+=Owner*
;

Owner:
    name=ID 'has' pets+=Pet ( 'and' pets+=Pet )
;

Pet:
    (Dog | Cat)
    '(' weight=INT 'kg' ')'
    (',' 'who' 'is' 'friends' 'with' friends+=[Pet|STRING] ('and' friends+=[Pet|
STRING])* )?
;

Dog:
    'called' name=STRING
    'a' 'dog'
;

Cat:
    'called' name=STRING
    'a' 'cat'
;
```

```
grammar de.hub.modsoft.Fidog with org.eclipse.xtext.common.Terminals

generate fidog "http://www.hub.de/modsoft/Fidog"

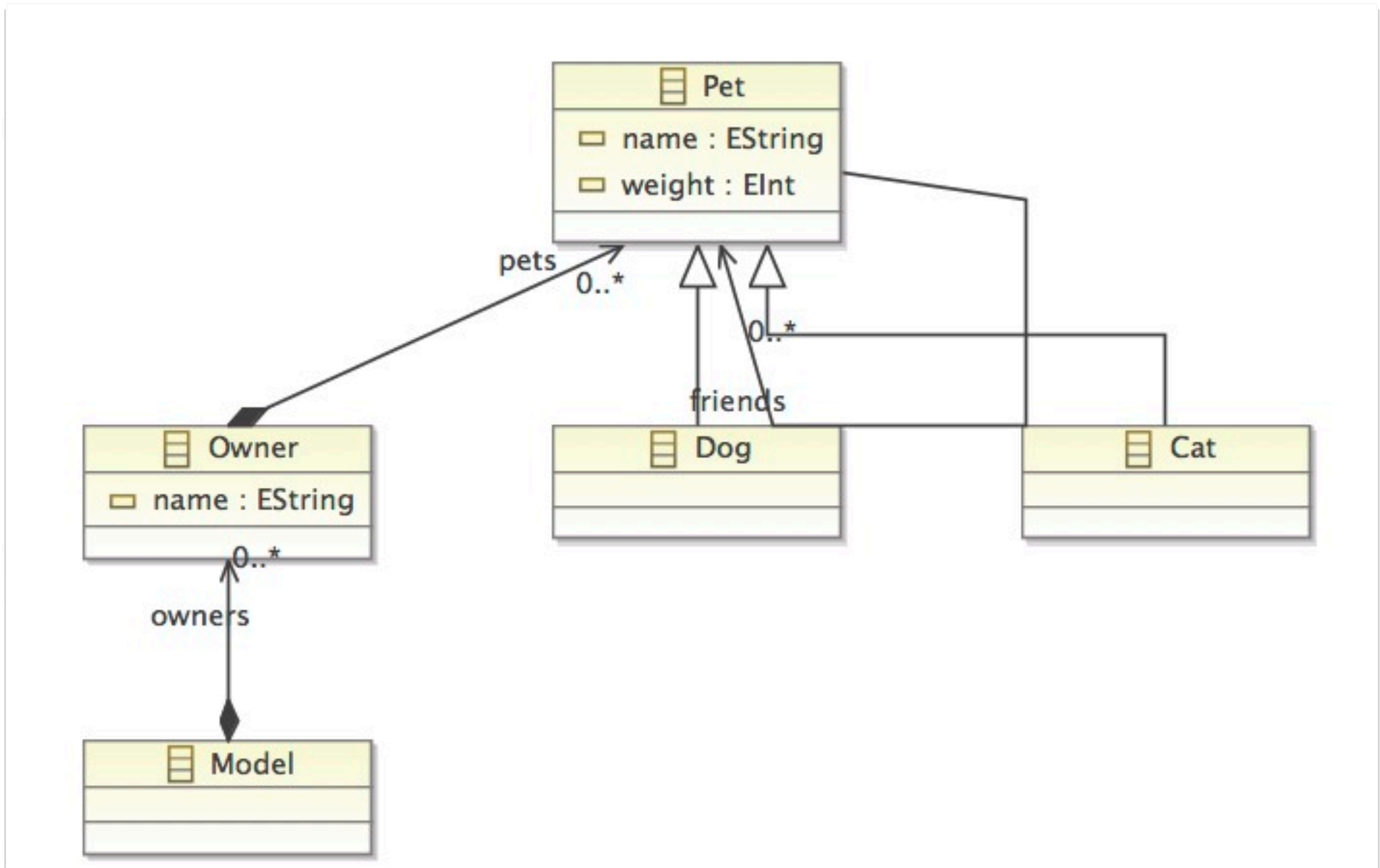
Model:
    owners+=Own
;

Owner:
    name=ID 'ha
;

Pet:
    (Dog | Cat)
    '(' weight=
    (',' 'who'
STRING])* )?
;

Dog:
    'called' na
    'a' 'dog'
;

Cat:
    'called' name=STRING
    'a' 'cat'
;
```



```
grammar de.hub.modsoft.Fidog with org.eclipse.xtext.common.Terminals

generate fidog "http://www.hub.de/modsoft/Fidog"

Model:
    owners+=Owner*
;

Owner:
    name=ID 'has' pets+=Pet ( 'and' pets+=Pet )
;

Pet:
    (Dog | Cat)
    'called' name=STRING
    (',' 'who' 'is' 'friends' 'with' friends+=[Pet|STRING] ('and' friends+=[Pet|
STRING])* )?
;

Dog:
    'a' 'dog'
    '(' weight=INT 'kg' ')'
;

Cat:
    {Cat}
    'a' 'cat'
;
```

```
grammar de.hub.modsoft.Fidog with org.eclipse.xtext.common.Terminals

generate fidog

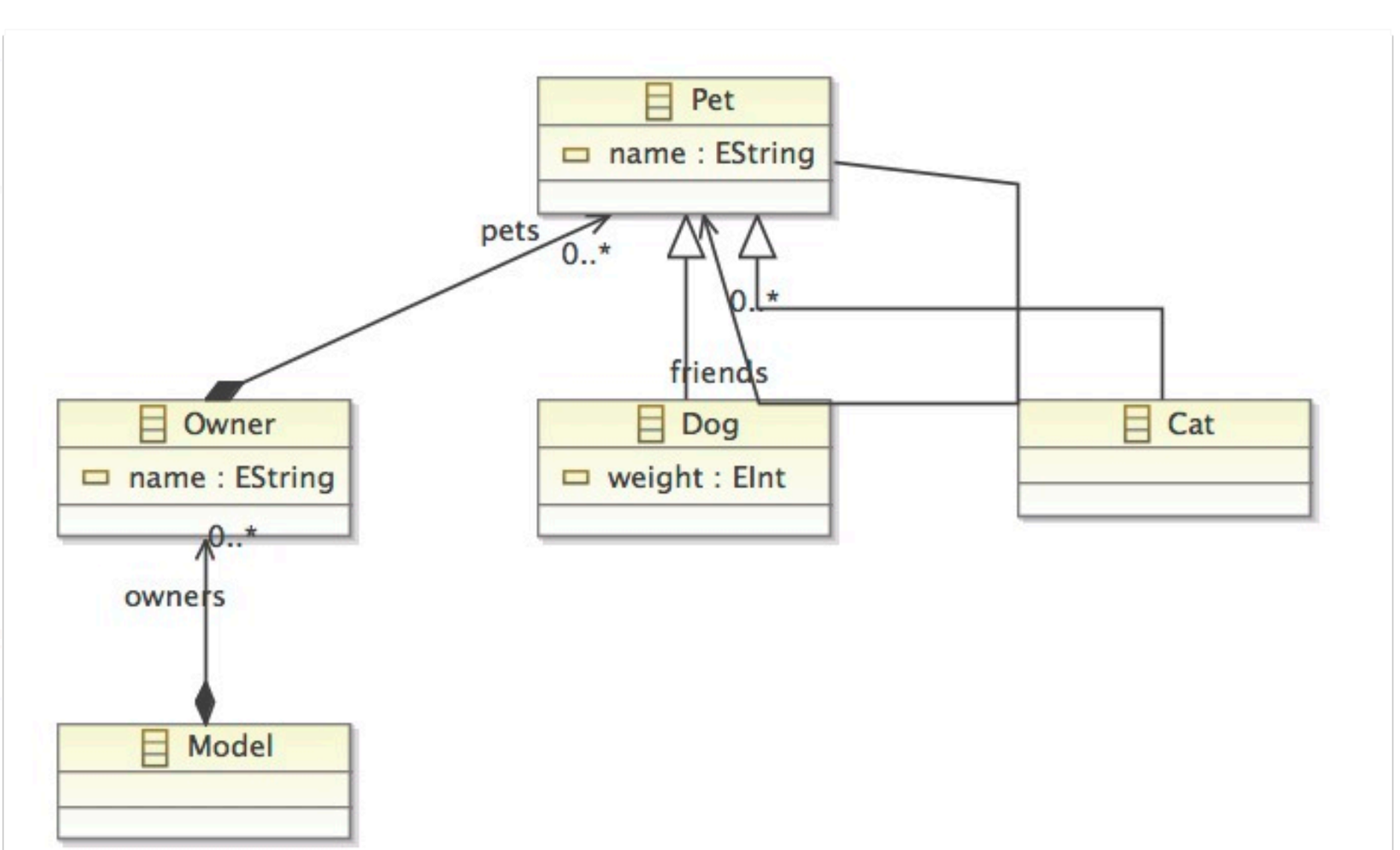
Model:
    owners+=Owne
;

Owner:
    name=ID 'has
;

Pet:
    (Dog | Cat)
    'called' nam
    (',' 'who' '
STRING])* )?
;

Dog:
    'a' 'dog'
    '(' weight=INT 'kg' ')'
;

Cat:
    {Cat}
    'a' 'cat'
;
```



xText's Basic Grammar Syntax

- EBNF
- rules: <name> ':' <rule> ';'
- parser rules:
  - groups: <rule>+ | '(' <rule>+ ')'
  - alternatives: <rule> ('|' <rule>)*
  - keywords: ''' <keyword> '''
  - rule calls: <name>
  - EBNF expressions: *,+,?
- examples:
  - `Pet : Dog | Cat`
  - `ParameterList : ( Type Name (',' Type Name)*)?`

xText – Meta-Model Mapping Syntax

- rules: <name> ‘returns’ <ecore type> ‘:’ <rule> ‘;’
- parser rules:
  - rule calls: <feature name> (‘=’ | ’+=’ | ‘?=’) (<rule call> | ‘[‘ <ecore type> ‘|’ <rule call> ‘]’)
    - =, assign
    - +=, add value
    - **?=**, assign true if the following rule call is consumed, assign false otherwise, works only for boolean features.
  - instantiate: ‘{‘<ecore type>’}’
- example:
  - ```
    Dog returns fido::Dog :
      {fido::Dog}
      ‘a’ ‘Dog’ ‘called’ name=STRING
      ‘friends’ ‘with’ friends+=[fido::Pet|STRING];
    ```
    ```

Generated vs. Imported Meta-Model

```
grammar de.hub.modsoft.Fidog with org.eclipse.xtext.common.Terminals

generate fidog "http://www.hub.de/modsoft/Fidog"

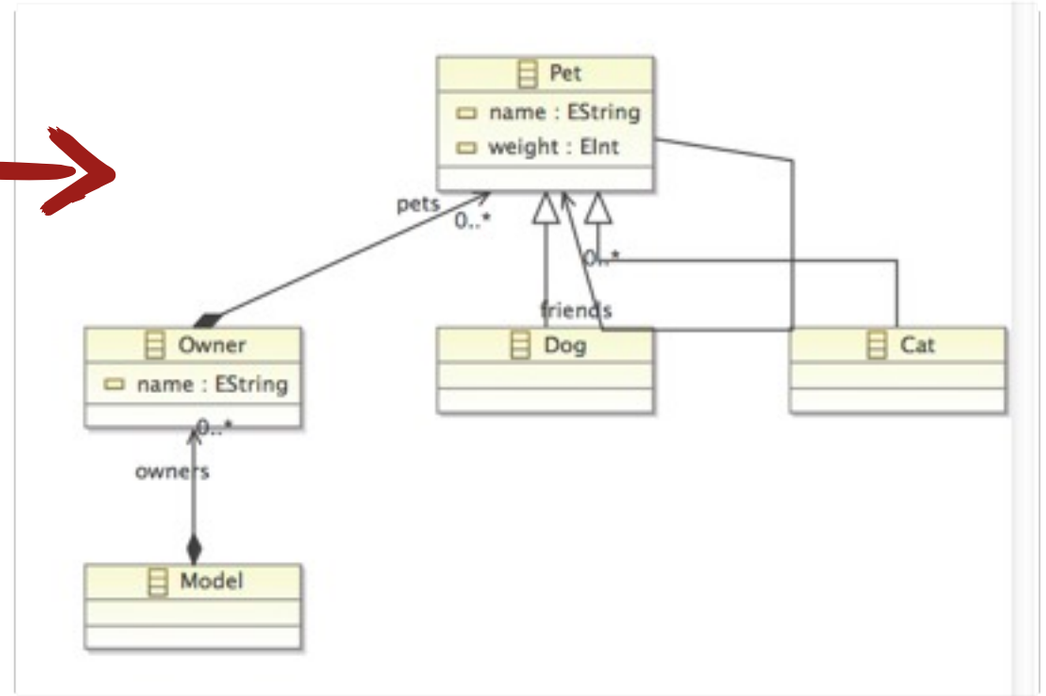
Model:
	owners+=Owner*
;

Owner:
	name=ID 'has' pets+=Pet ( 'and' pets+=Pet )
;

Pet:
	(Dog | Cat)
	'called' name=STRING '(' weight=INT 'kg' ')'
	(',' 'who' 'is' 'friends' 'with' friends+=[Pet|STRING]
		('and' friends+=[Pet|STRING])* )?
;

Dog:
	{Dog}
	'a' 'dog'
;

Cat:
	{Cat}
	'a' 'cat'
;
```



Generated vs. Imported Meta-Model

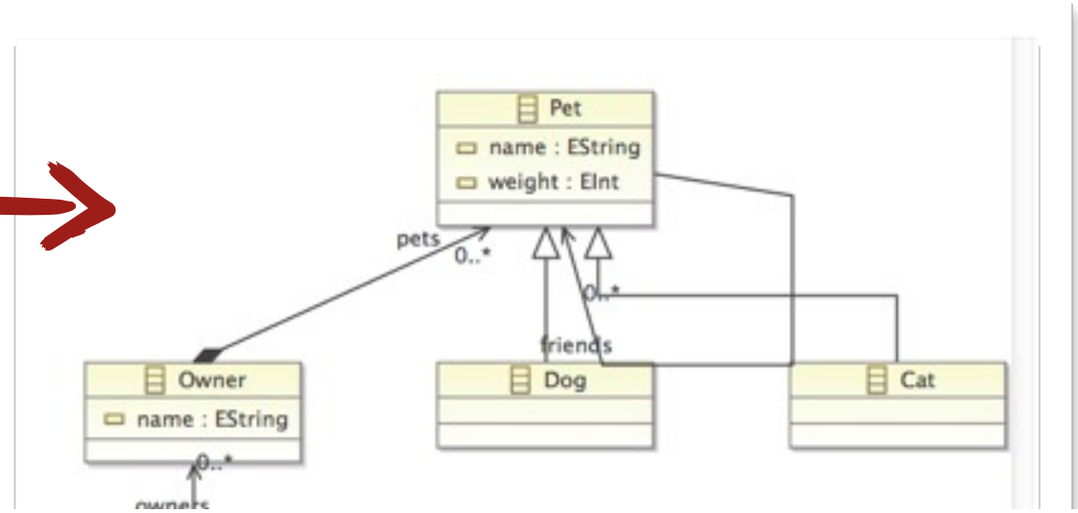
```
grammar de.hub.modsoft.Fidog with org.eclipse.xtext.common.Terminals

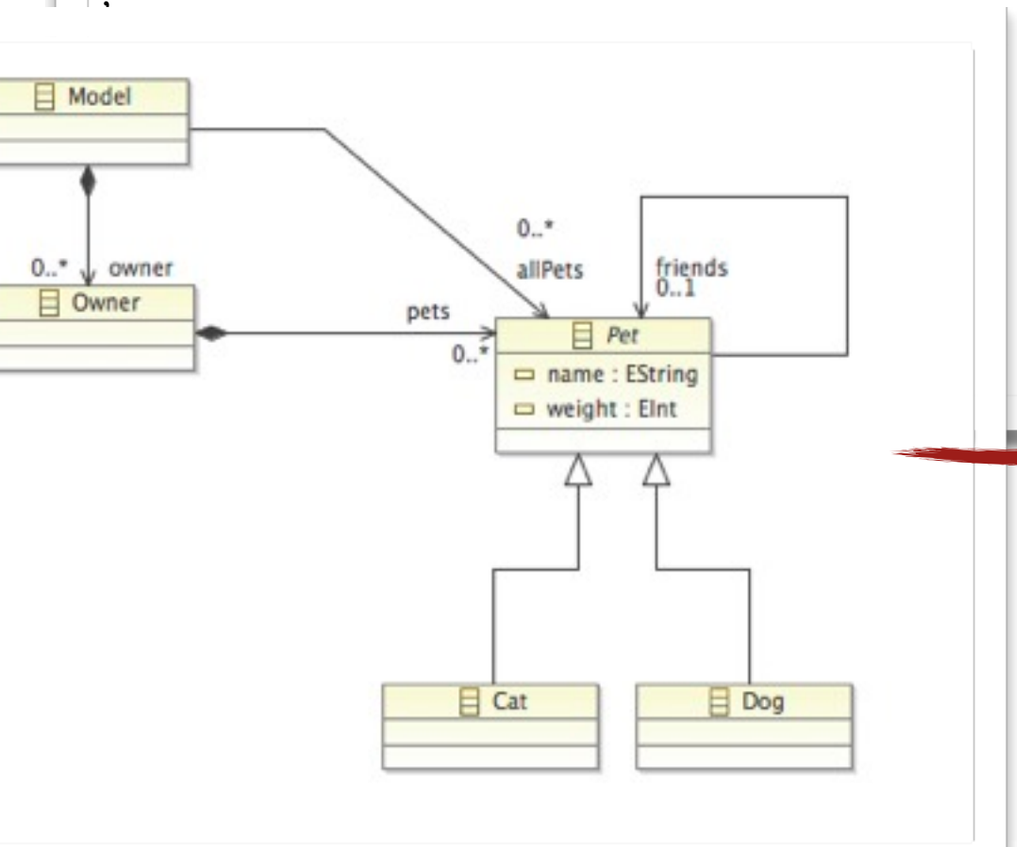
generate fidog "http://www.hub.de/modsoft/Fidog"

Model:
      owners+=Owner*
;

Owner:
      name=ID 'has' pets+=Pet ( 'and' pets+=Pet )
;

Pet:
      (Dog | Cat)
      'called' name=STRING '(' weight=INT 'kg' ')'
      (',' 'who' 'is' 'friends' 'with' friends+=[Pet|STRING]
                ('and' friends+=[Pet|STRING])* )?
;
```






```
// automatically generated by Xtext
grammar de.hub.modsoft.fido.Notation with org.eclipse.xtext.common.Terminals

import "http://fido/1.0"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

Model returns Model:
      {Model}
      'Model'
      '{'
            ('owner' '{' owner+=Owner ( "," owner+=Owner)* '}' )?
      '}';


Pet returns Pet:
      Dog | Cat;

Owner returns Owner:
      {Owner}
      'Owner'
      '{'
            ('pets' '{' pets+=Pet ( "," pets+=Pet)* '}' )?
      '}';

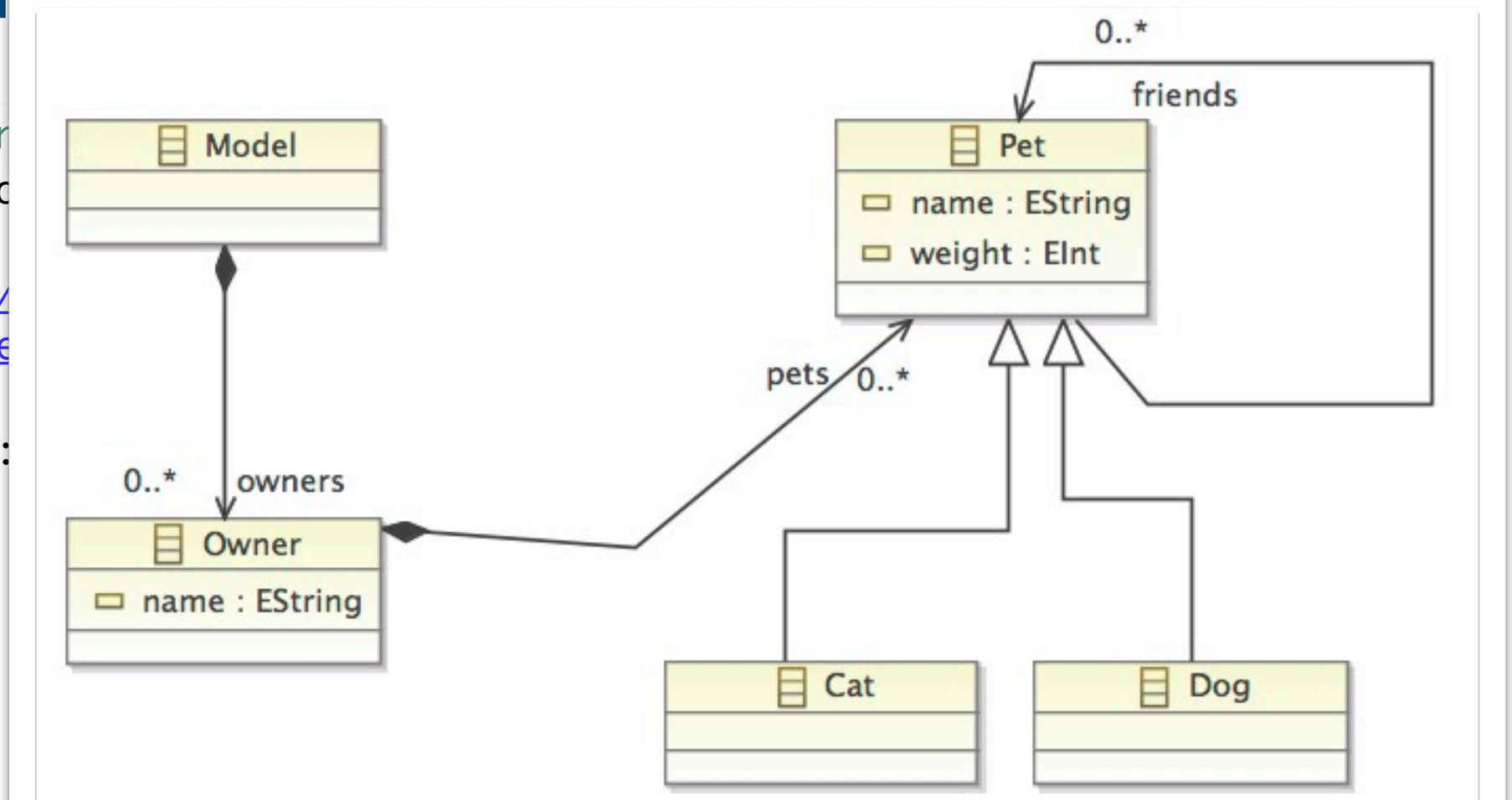
EString returns ecore::EString:
      STRING | ID;

EInt returns ecore::EInt:
```

Reasons for Non Generated Meta-Model

- Notation for existing meta-model
- Meta-Model with more elements as covered by textual notation
- Operations, derived features

Gramm




```
// automatically ger
grammar de.hub.modsc

import "http://fido/
import "http://www.e

Model returns Model:
    {Model}
    'Model'
    '{'
        ('owner' '{'
    '}';


Pet returns Pet:
    Dog | Cat;

Owner returns Owner:
    {Owner}
    'Owner'
    '{'
        ('pets' '{' pets+=Pet ( "," pets+=Pet)* '}' )?
    '}';

EString returns ecore::EString:
    STRING | ID;
```

Grammar Generated from Meta-Model

```
// automatically generated by Xtext
grammar de.hub.modsoft.fido.Notation with org.eclipse.xtext.common.Terminals

import "http://fido/1.0"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

Model returns Model:
    {Model}
    'Model'
    '{'
        ('owner' '{' owner+=Owner ( "," owner+=Owner)* '}' )?
    '}';


Pet returns Pet:
    Dog | Cat;

Owner returns Owner:
    {Owner}
    'Owner'
    '{'
        ('pets' '{' pets+=Pet ( "," pets+=Pet)* '}' )?
    '}';

EString returns ecore::EString:
    STRING | ID;
```

```
    'Owner'
    '{'
       ('pets' '{' pets+=Pet ( "," pets+=Pet)* '}' )?
    '}';

EString returns ecore::EString:
    STRING | ID;

EInt returns ecore::EInt:
    '-'? INT;

Dog returns Dog:
    {Dog}
    'Dog'
    name=EString
    '{'
       ('weight' weight=EInt)?
       ('friends' friends=[Pet|EString])?
    '}';

Cat returns Cat:
    {Cat}
    'Cat'
    name=EString
    '{'
       ('weight' weight=EInt)?
       ('friends' friends=[Pet|EString])?
    '}';
```

Generated/Import Syntax

- `generate <packageName> “<nsURI>”`
- `import “<nsURI>” as <packageID>`

Grammar Mixins

- Combine several grammars
  - for reuse, e.g. common Terminals
  - modularization
  - language extensibility, e.g. xBase

```
grammar de.hub.modsoft.fido.Notation with org.eclipse.xtext.common.Terminals

import "http://fido/1.0"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

Model returns Model:
    (owner += Owner)*;

EString returns ecore::EString:
    STRING | ID;

EInt returns ecore::EInt:
    '-'? INT;
```

Grammar Mixins

```
grammar org.eclipse.xtext.common.Terminals hidden(WS, ML_COMMENT, SL_COMMENT)

import "http://www.eclipse.org/emf/2002/Ecore" as ecore

terminal ID      : '^'?('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')*;
terminal INT returns ecore::EInt: ('0'..'9')+;
terminal STRING  :
          '"' ( '\\' ('b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\') | !('\\'|'"') )* '"' |
          "'" ( '\\' ('b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\') | !('\\'|"'") )* "'"
        ;
terminal ML_COMMENT : '/*' -> '*/';
terminal SL_COMMENT     : '//' !('\n'|'\r')* ('\r'? '\n')?;

terminal WS          : (' '|'\t'|'\r'|'\n')+;

terminal ANY_OTHER: .;
```

```
    (owner += Owner)*;

EString returns ecore::EString:
    STRING | ID;

EInt returns ecore::EInt:
    '-'? INT;
```

White Spaces

- Can be hidden, i.e. made unavailable for rule calls

```
grammar org.eclipse.xtext.common.Terminals hidden(WS, ML_COMMENT, SL_COMMENT)

import "http://www.eclipse.org/emf/2002/Ecore" as ecore

terminal ID      : '^'?('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')*;
terminal INT returns ecore::EInt: ('0'..'9')+;
terminal STRING  :
          '"' ( '\\' ('b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\') | !('\\'|'"') )* '"' |
          "'" ( '\\' ('b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\') | !('\\'|"'") )* "'"
       ;
terminal ML_COMMENT : '/*' -> '*/';
terminal SL_COMMENT    : '//' !('\n'|'\r')* ('\r'? '\n')?;

terminal WS         : (' '|'\t'|'\r'|'\n')+;

terminal ANY_OTHER: .;
```

Terminals

- terminal rule: ‘terminal’ (‘fragment’)? <name> (‘returns’ <ecore type>) ‘:’ <EBNF expression with keywords and terminals> ‘;’
- examples
  - `terminal INT returns ecore::EInt : (‘0’..’9’)+;`
  - `terminal DOUBLE returns ecore::EDouble : INT ‘.’ INT;`
  - `terminal fragment ESCAPED_CHAR : '\\' ('n'|'t'|'r'|'\\');`

```
terminal STRING  :
    '"' ( ESCAPED_CHAR | !('\\'|'"') )* '"' |
    "'" ( ESCAPED_CHAR | !('\\'|"'") )* "'";
```

- semantics: `EFactory#createFromString(EDataType, String)`

Enums

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

Limitations: LL(*) and Ambiguous Grammars

- xText uses ANTLR a LL(*) parser generator for Java
  - no left recursion, e.g. `Expr : Expr '+' Expr` is not allowed
  - *dangling else* problem, e.g.

```
If: 'if' Condition 'then' Block ( 'else' Block)?;

Block : Statement*;

Statement: IF | ...;
```

```
if a then
print("a");
if b then
print("b");
else
print("-");
```

Limitations: LL(*) and Ambiguous Grammars

- xText uses ANTLR a LL(*) parser generator for Java
  - no left recursion, e.g. `Expr : Expr '+' Expr` is not allowed
  - *dangling else* problem, e.g.

```
If: 'if' Condition 'then' Block ( 'else' Block)?;

Block : Statement*;

Statement: IF | ...;
```

```
if a then
    print("a");
    if b then
        print("b");
    else
        print("-");
```

Limitations: LL(*) and Ambiguous Grammars

- xText uses ANTLR a LL(*) parser generator for Java
  - no left recursion, e.g. `Expr : Expr '+' Expr` is not allowed
  - *dangling else* problem, e.g.

```
If: 'if' Condition 'then' Block ( 'else' Block)?;

Block : Statement*;

Statement: IF | ...;

if a then
   print("a");
   if b then
      print("b");
 else
      print("-");
```

Predicates

- xText uses ANTLR which uses syntactic predicates
  - *syntactic predicate* operator =>

```
If: ‘if’ Condition ‘then’ Block (=> ‘else’ Block)?;

Block : Statement*;

Statement: IF | ...;
```

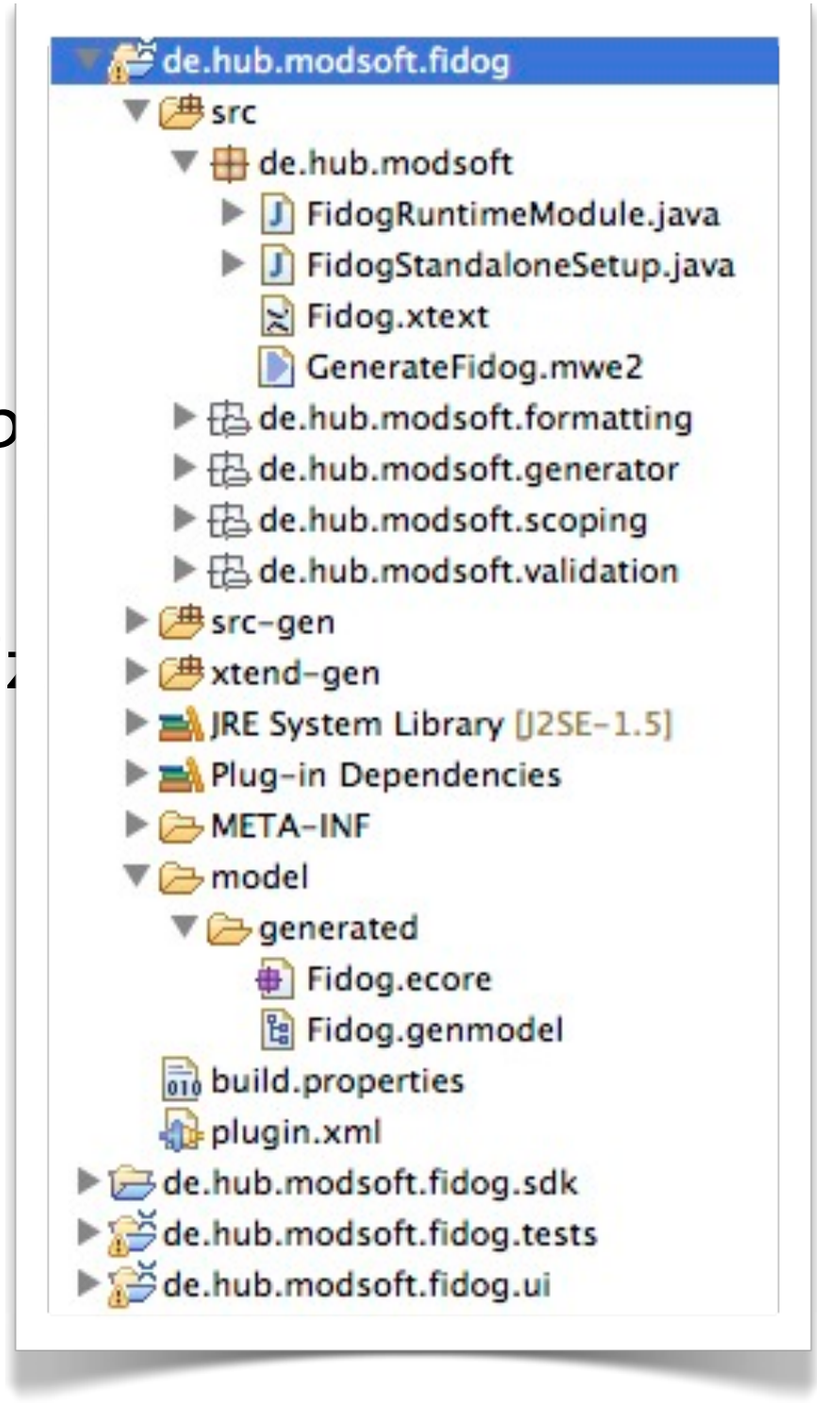
xText – Configuration

Compile Time Configuration

- Generate<name>.mwe2
  - *Modeling Workflow Engine 2*: DSL to describe configuration, validation, and code generation steps as a repeatable receipt
- Will be generated by xText's new project wizzards.
- Usually does not need to be touched.

Compile Time Configuration

- Generate<name>.mwe2
  - *Modeling Workflow Engine 2*: DSL to describ
    validation, and code generation steps as a
  - Will be generated by xText's new project wi
  - Usually does not need to be touched.

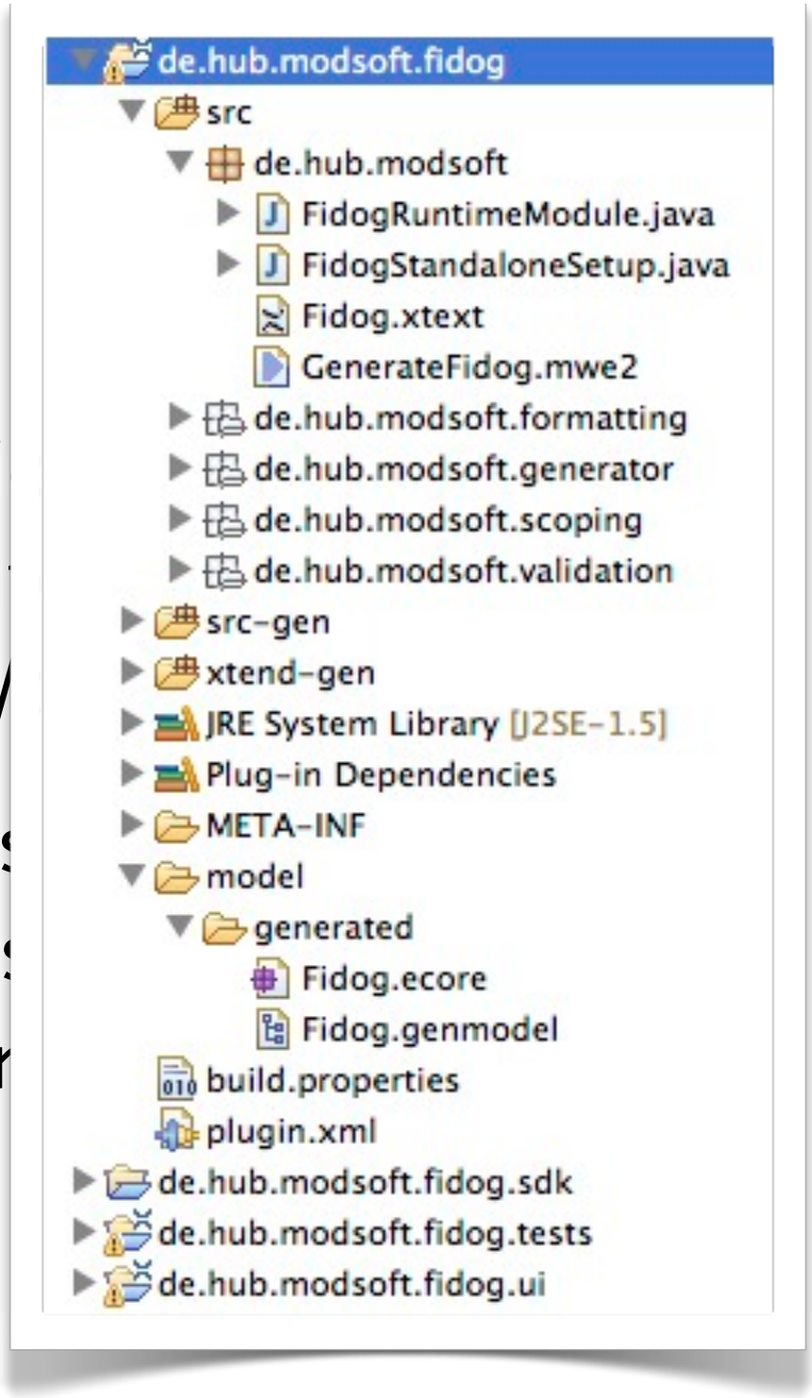


Runtime Configuration

- Different xText components (e.g. validation, reference resolution, pretty printing, ...) have an interface and multiple implementations for different strategies/defaults
- Dependency injection allows clients to assign concrete implementation or own implementations to those components from within a single class in the project.

Runtime Configuration

- Different xText components (e.g. validati
  resolution, pretty printing, ...) have an in
  implementations for different strategies/
- Dependency injection allows clients to as
  implementation or own implementations
  components from within a single class ir



Dependency Injection

Dependency Injection – Wikipedia

- **Dependency injection** (**DI**) in object-oriented computer programming is a technique that indicates to a part of a program which other parts it can use, i.e. to supply an external dependency – a reference – to a software component. In technical terms, it is a design pattern that separates behavior from dependency resolution, thus decoupling highly dependent components.

Example without DI

```
public class Messenger {
    private MessageService service;
    // dependency resolution
    public Messenger(MessageService svc){
        this.service=svc;
    }
    public boolean sendMessage(String msg, String rec){
        // behaviour
        return service.sendMessage(msg, rec);
    }
}
```

```
public class ClientApplication {

    public static void main(String[] args) {
        // dependency resolution
        Messenger app = new Messenger(new SMSService());
        // behaviour
        app.sendMessage("Hi Pankaj", "pankaj@abc.com");
    }
 }
```

Example with DI

```
public class MyApplication {
    @Inject
    private MessageService service;

    public boolean sendMessage(String msg, String rec){
        // behaviour
        return service.sendMessage(msg, rec);
    }
}
```

```
public class ClientApplication {
    public static void main(String[] args) {
        Injector injector = Guice.createInjector(new MyModule());
        Messeger app = injector.getInstance(Messenger.class);

        // behaviour
        app.sendMessage("Hi Pankaj", "pankaj@abc.com");
    }
}
```

Separation of Concerns

- Only implement business logic in your classes
- Dependencies and their resolution is defined else were
- Dependency resolution is easy to modify, replace
- Example use cases
  - Configuration of complex applications
  - Writing tests and replacing implementations with mock-ups

Guice Dependency Injection for Java by Google

- Code concepts
  - Injector, creates objects and resolves dependencies based on a module
  - Module, describes dependency resolutions with bindings and, can be nested
  - Bindings bind concrete implementations to dependencies (i.e. types)
  - Injections declare fields and parameters that need to be injected (resolved)

```
public class FidoNotationRuntimeModule extends GeneratedFidoNotationRuntimeModule {
    void configure(Binder binder) {
        super.configure(binder)
        binder.bind(IScopeProvider.class).to(FidoNotationScopeProvider.class);
    }
}
```

xText – Runtime

Two Distinct Scenarios

- *Eclipse* with UI and generated xText editor
  - Eclipse workspace is present
  - outline, code completion, etc. are important
- *standalone parser*

xText's Runtime Concepts

- Validation
- Linking
- Scoping
- Value Converter
- Serialization
- Formatting
- Fragments
- Encoding
- default behavior for all those can be overwritten via dependency injection

Linking

- ILinker is responsible to create all links corresponding to cross references
- LazyLinker: ILinker
  - creates proxies first
  - uses LazyLinkingResource which delegates proxy resolution to ILinkingService
  - default ILinkingService delegates to xText's scoping API

IScope

```
java.lang.Iterable<IEObjectDescription> getAllElements()

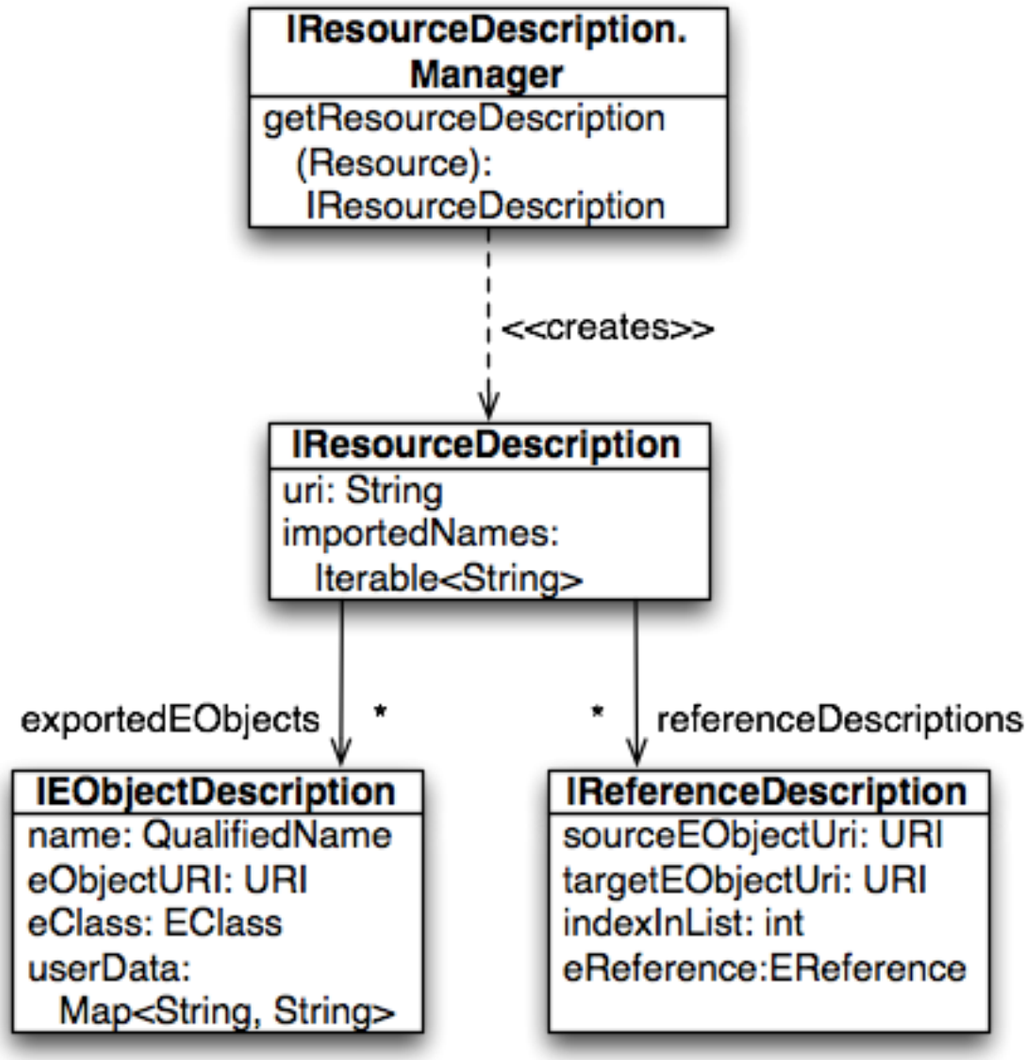
java.lang.Iterable<IEObjectDescription> getElements(EObject object)

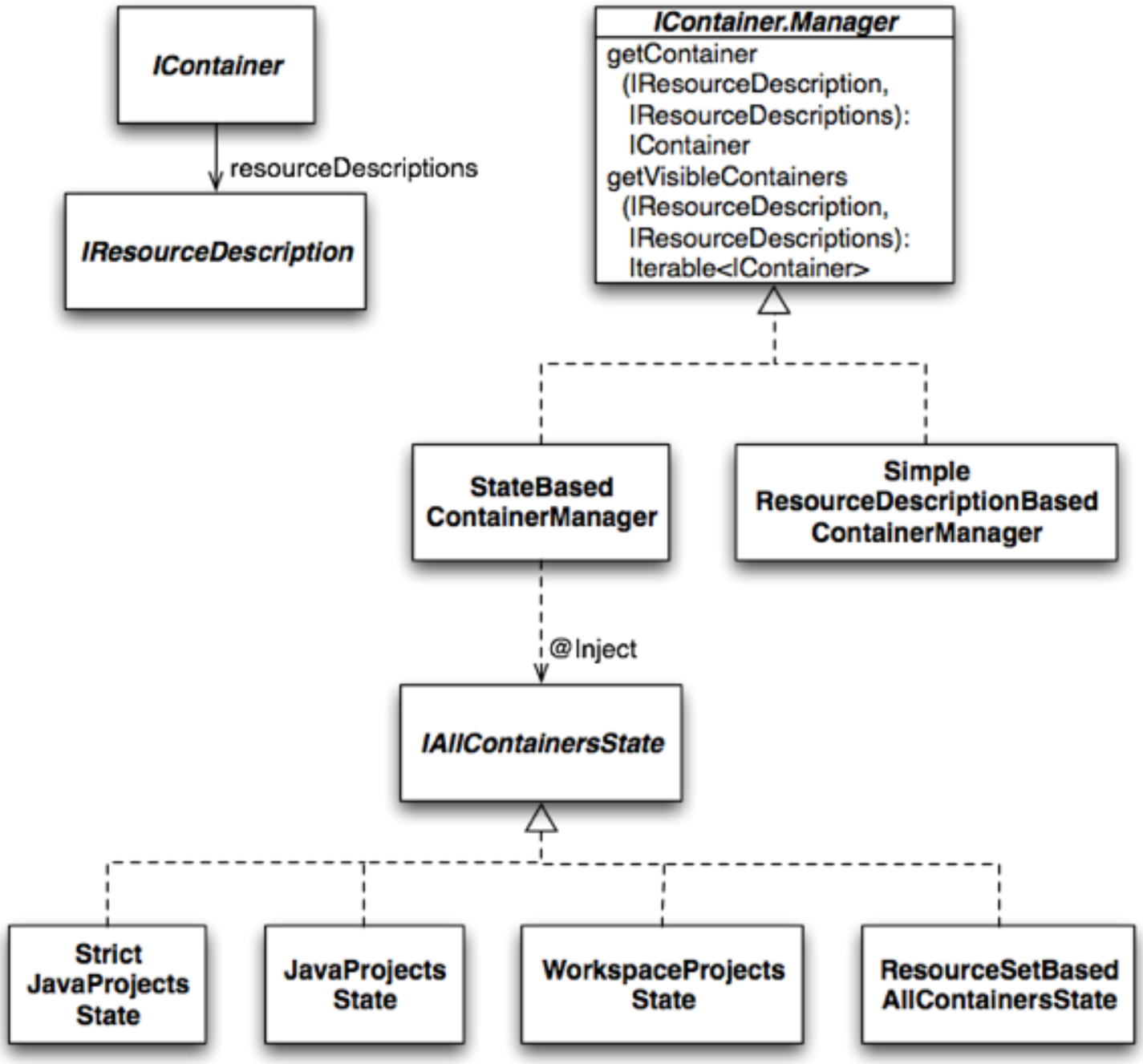
java.lang.Iterable<IEObjectDescription>getElements(QualifiedName name)

IEObjectDescription getSingleElement(org.eclipse.emf.ecore.EObject object)

IEObjectDescription getSingleElement(QualifiedName name)
```

EObject and Resource Descriptions, Containers





Scoping – Global vs. Local

- Global scopes are used to identify elements from other resources (e.g. other text files, other models)
- Local scopes are used to identify elements within the same resource
- xText provides different strategies for both

Global Scopes

- IGlobalScopeProvider
  - AbstractGlobalScopeProvider
    - DefaultGlobalScopeProvider
    - ImportUriGlobalScopeProvider
    - ResourceSetGlobalScopeProvider
  - AbstractTypeScopeProvider
    - ClasspathBasedTypeScopeProvider
    - JdtBasedSimpleTypeScopeProvider

Global Scope – Example

```
public class FidoNotationRuntimeModule extends
         de.hub.modsoft.fido.AbstractFidoNotationRuntimeModule {

    @Override
    public Class<? extends IGlobalScopeProvider> bindIGlobalScopeProvider() {
        return ImportUriGlobalScopeProvider.class;
    }
}
```

```
Model returns Model:
    ('import' importURI=STRING)*
    (owner+=Owner)*;
```

```
import "platform:/resource/test/source.fido"

Markus ownes
    a dog called Fido (20) who is friends with Monday and a cat called Ham (2)
```

```
Stephan ownes a dog called Monday (25)
```

Local Scopes

- IScopeProvider
  - AbstractScopeProvider
    - AbstractDeclarativeScopeProvider
    - DelegatingScopeProvider
    - XtextScopeProvider

Local Scope Example

```
public class FidoNotationRuntimeModule extends
          de.hub.modsoft.fido.AbstractFidoNotationRuntimeModule {
    @Override
    public Class<? extends IScopeProvider> bindIScopeProvider() {
        return FidoNotationScopeProvider.class;
    }
}
```

```
public class NotationScopeProvider extends AbstractDeclarativeScopeProvider {
    public FilteringScope scope_Pet_friends(final Pet context, EReference friendsRef) {
        Predicate<IEObjectDescription> predicate = new Predicate<IEObjectDescription>() {
            @Override
            public boolean apply(IEObjectDescription object) {
                return object.getEObjectOrProxy().eContainer() == context.eContainer()
                        && object.getEObjectOrProxy() != context;
            }
        };
        return new FilteringScope(getScope(context.eContainer(), friendsRef), predicate);
    }
}
```

Value Converter

- Allows to programmatically define values for strings reduced/consumed by (terminal) rules
- IValueConverterService
  - AbstractDeclarativeValueConverterService
    - DefaultTerminalConverters

Value Converter

- Allows to programmatically define values for strings reduced/consumed by (terminal) rules
- IValueConverterService
  - AbstractDeclarativeValueConverterService
    - DefaultTerminalConverters

```
@ValueConverter(rule = "WEIGHT")
public IValueConverter<TYPE> getMyRuleNameConverter() {
  return new IValueConverter<TYPE>() {
    @Override
    public TYPE toValue(String string, INode node) throws ValueConverterException { ... }

    @Override
    public String toString(TYPE value) throws ValueConverterException { ... }
  };
}
```

Value Converter Example

```
Pet returns Pet:
    (Dog | Cat)
    'called' name=ID '(' weight=WEIGHT ')'
    ('who' 'is' 'friends' 'with' friends+=[Pet|ID] ('and' friends+=[Pet|ID])*)?
;

WEIGHT returns ecore::EInt : INT ("kg"|"g");
```

```
public class FidoNotationValueConverters extends Ecore2XtextTerminalConverters {

  @ValueConverter(rule = "WEIGHT")
  public IValueConverter<Integer> getWeightConverter() {
    return new IValueConverter<Integer>() {

      @Override
      public Integer toValue(String string, INode node)
            throws ValueConverterException {
        if (string.endsWith("kg")) {
          return Integer.parseInt(string.substring(0,
                string.length() - 2).trim())*1000;
        } else if (string.endsWith("g")) {
          return Integer.parseInt(string.substring(0,
                string.length() - 1).trim());
        } else {
          throw new IllegalArgumentException();
```

```
public IValueConverter<Integer> getWeightConverter() {
  return new IValueConverter<Integer>() {

    @Override
    public Integer toValue(String string, INode node)
           throws ValueConverterException {
       if (string.endsWith("kg")) {
           return Integer.parseInt(string.substring(0,
                  string.length() - 2).trim())*1000;
       } else if (string.endsWith("g")) {
           return Integer.parseInt(string.substring(0,
                  string.length() - 1).trim());
       } else {
           throw new IllegalArgumentException();
       }
    }

       @Override
       public String toString(Integer value) throws ValueConverterException {
           if (value > 1000) {
               return (int)Math.round(value/1000.0) + "kg";
           } else {
               return value + "g";
           }
       }
     };
   }
}
```

Summary

- Simple EBNF grammar, xText is based on ANTLR, LL(*)
- Minimal mapping information to relate to or infer a meta-model
- Terminals and Grammars can be reused
- Various aspects can be modified, configuration via DI

Other Textural Editing Frameworks based on Parsing

- TCS/TCSSL, allows complex mappings with multiple passes, based on ANTLR and Jet
- HUTN, OMG Standard for creating generic textual notations
- TEF, academic, allows changes to syntax at runtime
- MontiCore, Gymnast, Spoofax, Frodo, ...
- xText and TCS are maintained under Eclipse TMF