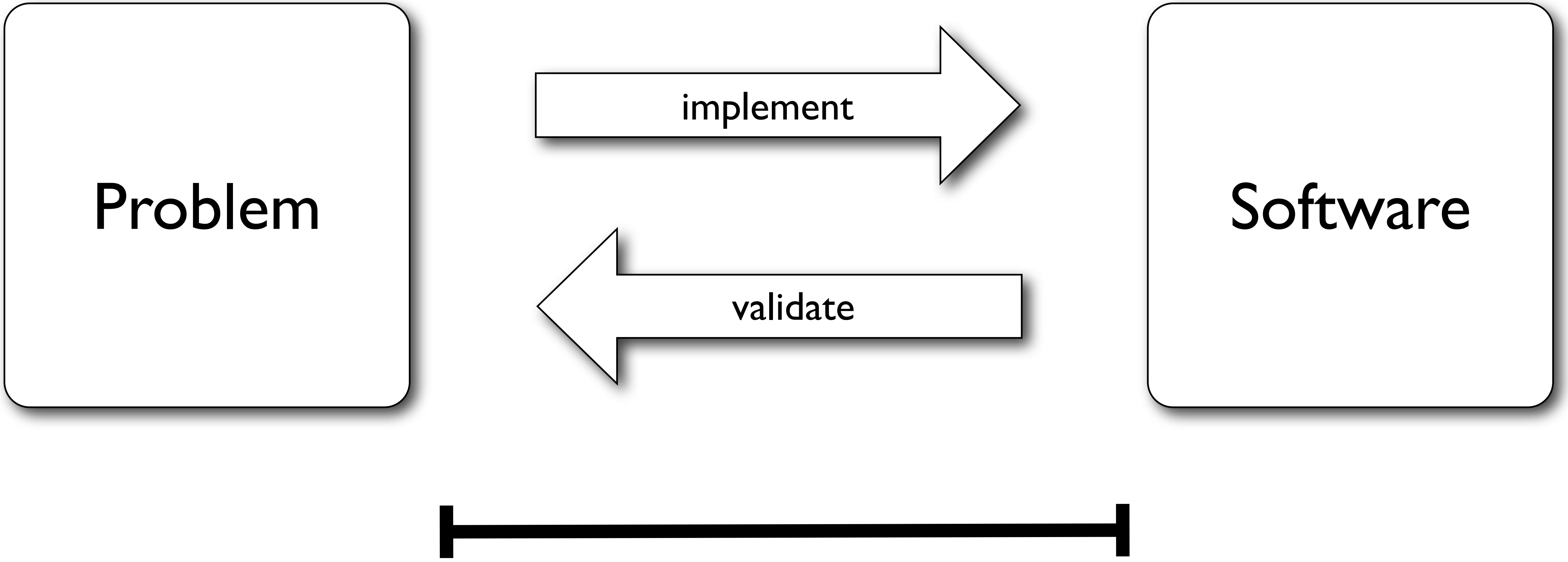
The Spoofax Language Workbench

Lennart Kats Eelco Visser

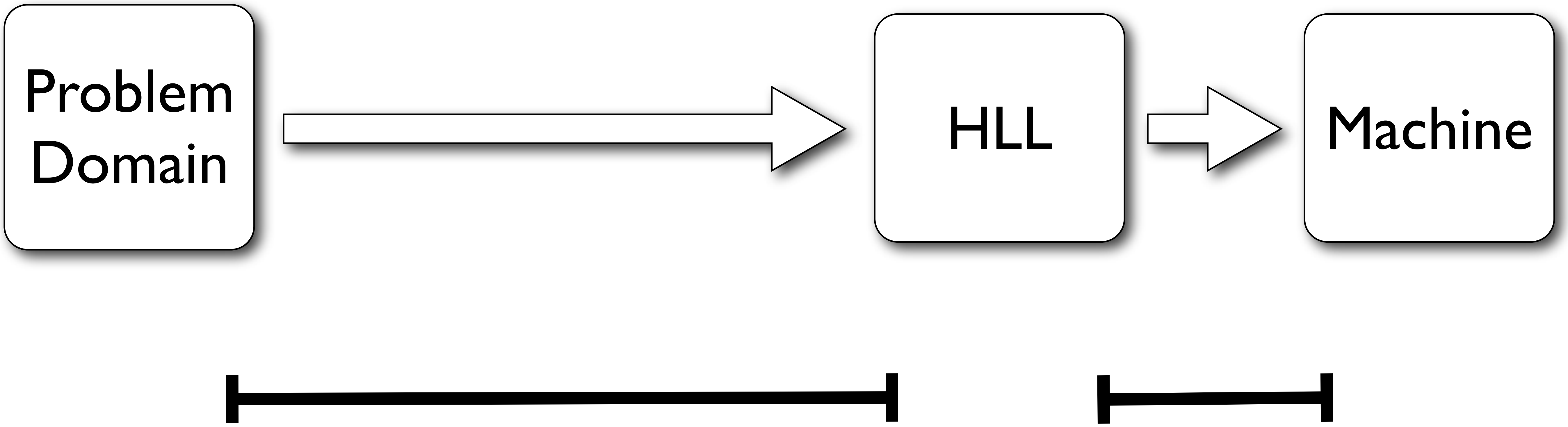
TU Delft

Delft University of Technology

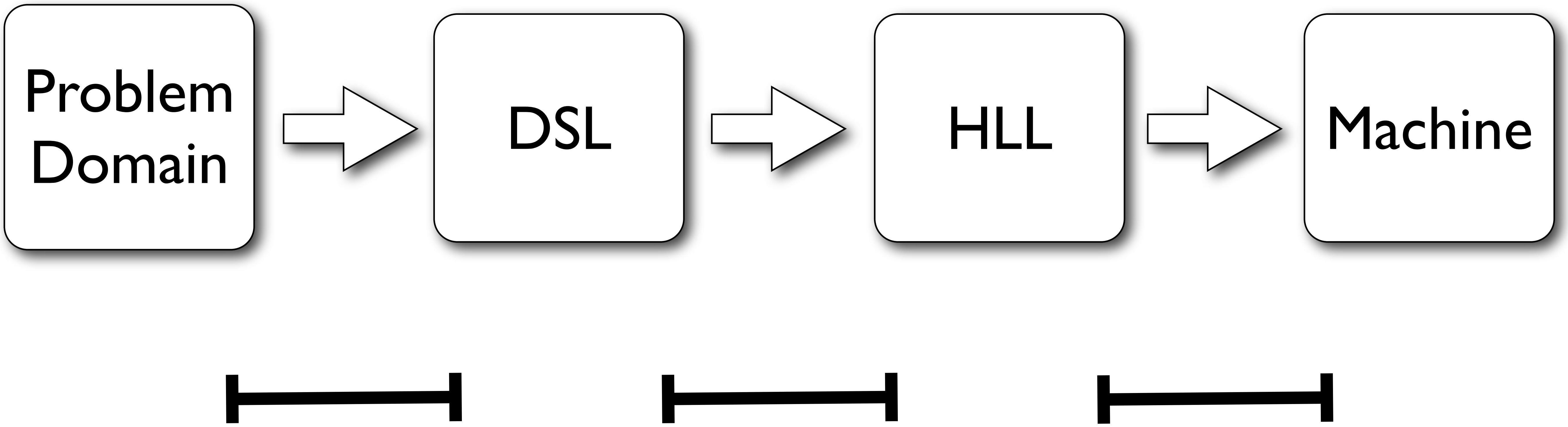
Software Engineering



High-level languages reduce problem/solution gap

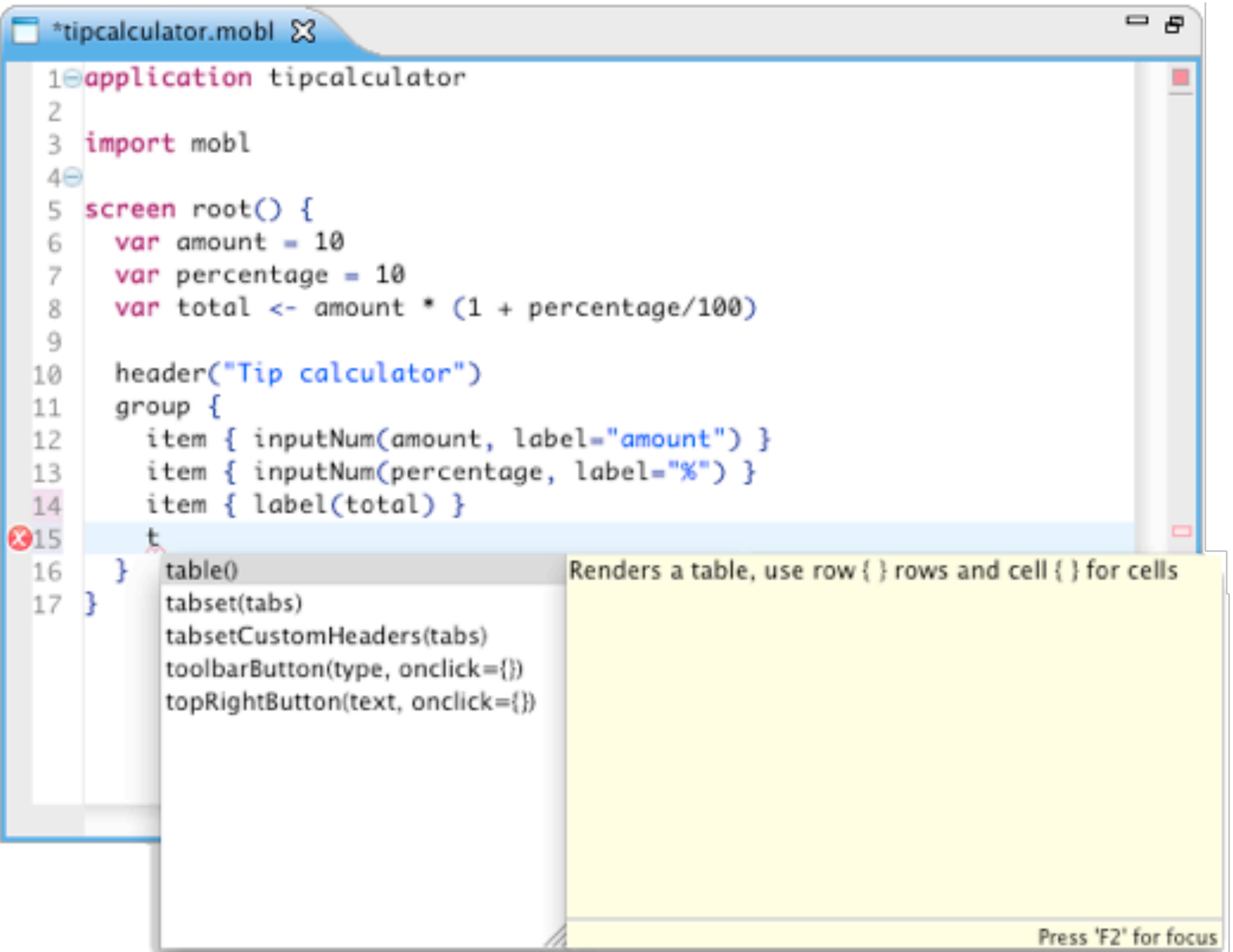


Domain-Specific Languages



DSLs further reduce gap between problem domain and implementation

Example DSL: mobl



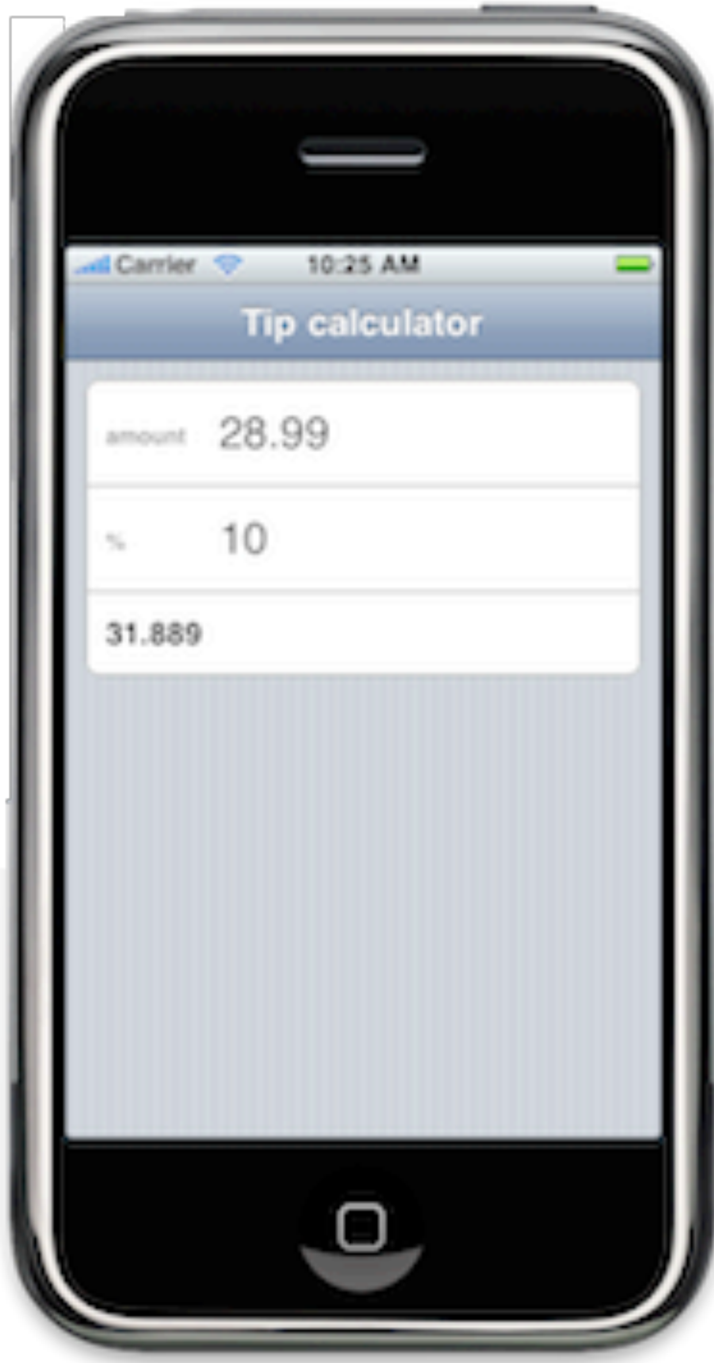
*tipcalculator.mobl

```
application tipcalculator

import mobl

screen root() {
  var amount = 10
  var percentage = 10
  var total <- amount * (1 + percentage/100)

  header("Tip calculator")
  group {
    item { inputNum(amount, label="amount") }
    item { inputNum(percentage, label="%") }
    item { label(total) }
    t
  }
}
```

table()
tabset(tabs)
tabsetCustomHeaders(tabs)
toolbarButton(type, onclick={})
topRightButton(text, onclick={})

Renders a table, use row { } rows and cell { } for cells

Press 'F2' for focus



Tip calculator

amount 28.99

% 10

31.889

paradigm: linguistic abstraction

making languages should be cheap

Compiler Ingredients

Syntax definition

- ★ concrete syntax
- ★ abstract syntax

Static semantics

- ★ error checking
- ★ name resolution
- ★ type analysis

Model-to-model tra

- ★ express constructs in co guage

Code generation

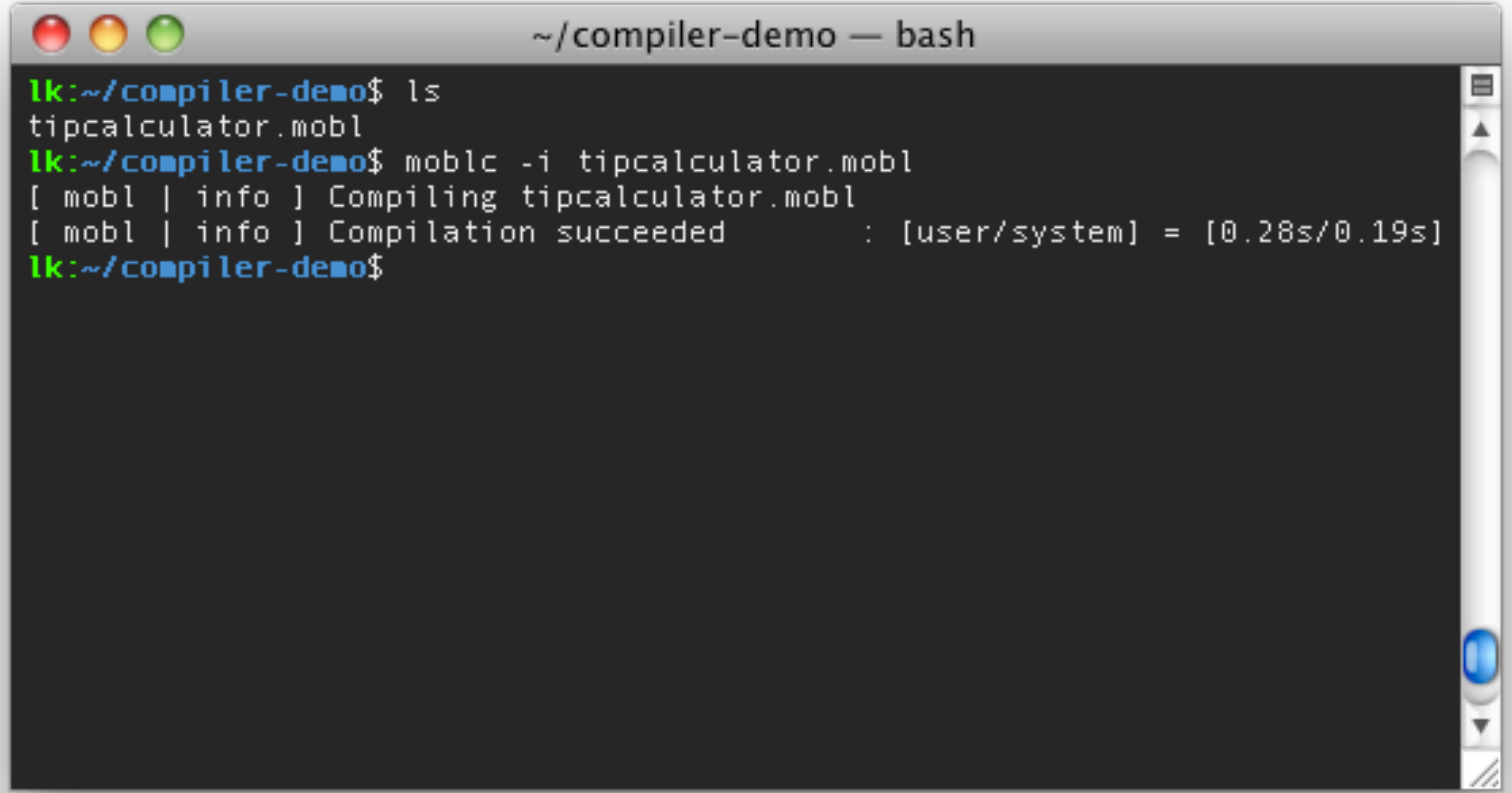
- ★ translate core language models to implementation

parser generators

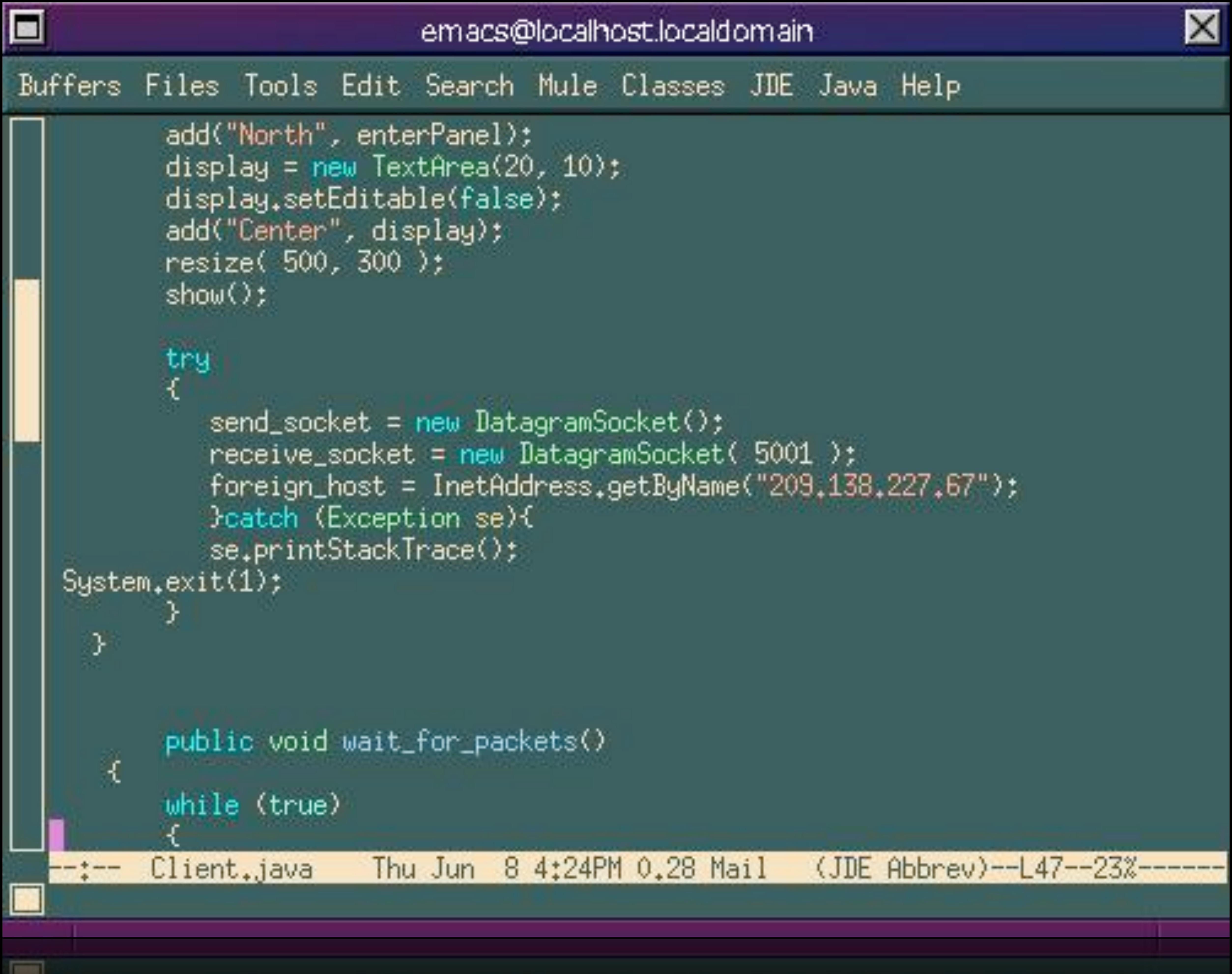
meta-programming libraries

meta-programming languages

template engines

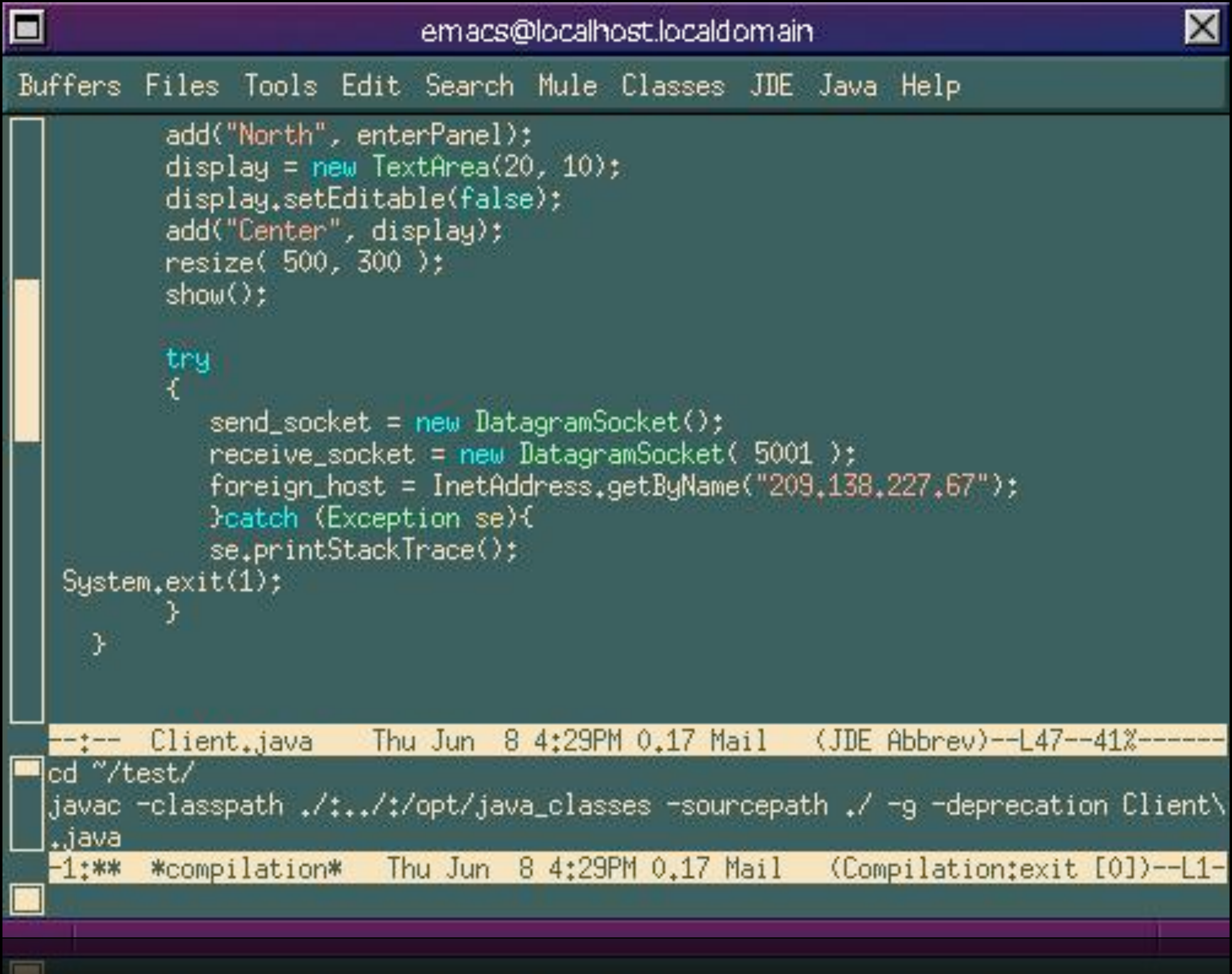


```
lk:~/compiler-demo$ ls
tipcalculator.mobl
lk:~/compiler-demo$ moblc -i tipcalculator.mobl
[ mobl | info ] Compiling tipcalculator.mobl
[ mobl | info ] Compilation succeeded        : [user/system] = [0.28s/0.19s]
lk:~/compiler-demo$
```

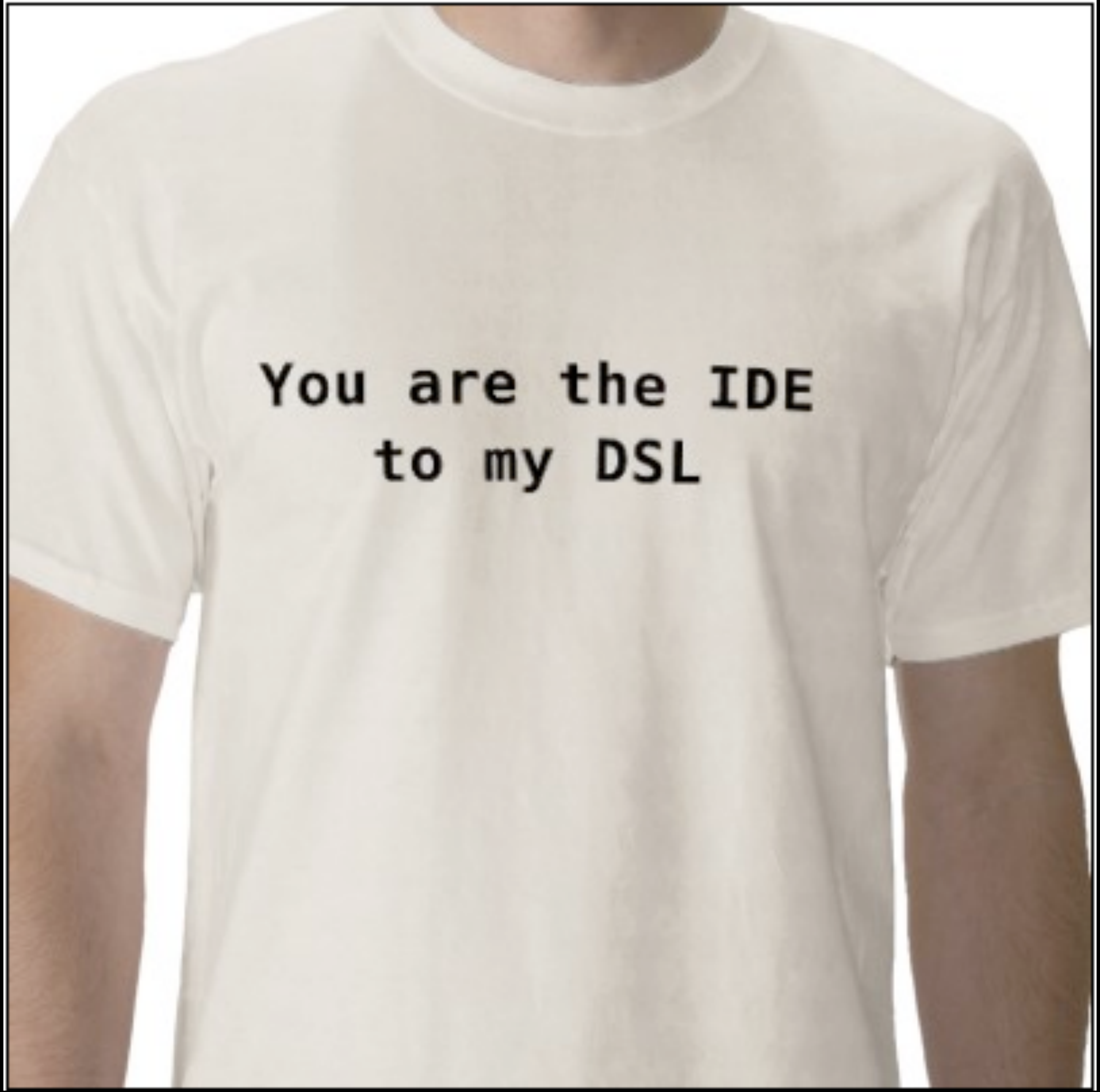


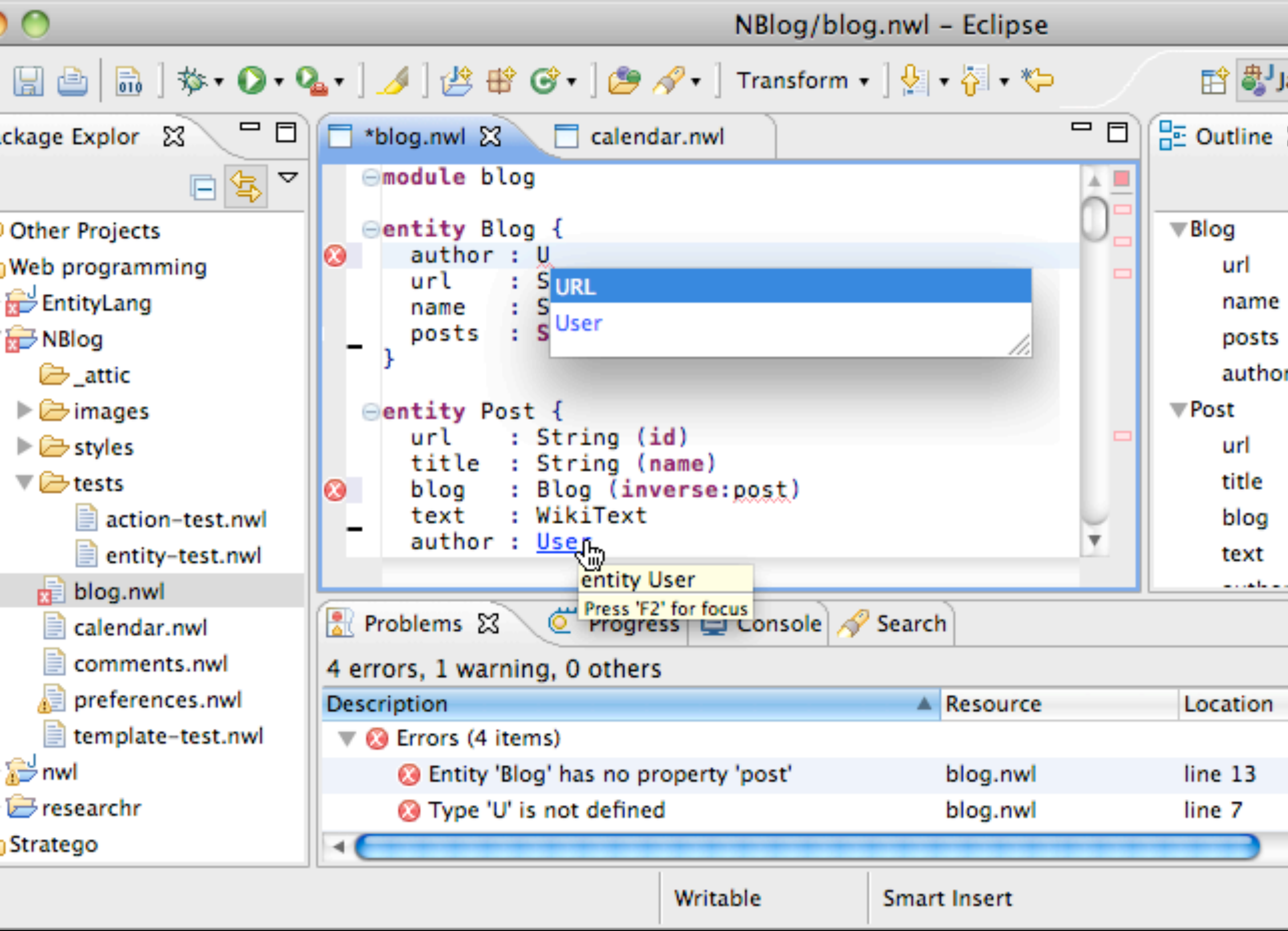
emacs@localhost.localdomain
Buffers Files Tools Edit Search Mule Classes JDE Java Help
--:-- Client.java   Thu Jun 8 4:24PM 0.28 Mail   (JDE Abbrev)--L47--23%-----

```
       add("North", enterPanel);
       display = new TextArea(20, 10);
       display.setEditable(false);
       add("Center", display);
       resize( 500, 300 );
       show();

       try
       {
          send_socket = new DatagramSocket();
          receive_socket = new DatagramSocket( 5001 );
          foreign_host = InetAddress.getByName("209.138.227.67");
          }catch (Exception se){
          se.printStackTrace();
System.exit(1);
       }
  }


       public void wait_for_packets()
   {
       while (true)
       {
```


emacs@localhost.localdomain
Buffers Files Tools Edit Search Mule Classes JDE Java Help

```
        add("North", enterPanel);
        display = new TextArea(20, 10);
        display.setEditable(false);
        add("Center", display);
        resize( 500, 300 );
        show();

        try
        {
           send_socket = new DatagramSocket();
           receive_socket = new DatagramSocket( 5001 );
           foreign_host = InetAddress.getByName("209.138.227.67");
           }catch (Exception se){
           se.printStackTrace();
 System.exit(1);
        }
   }

--:--  Client.java    Thu Jun  8 4:29PM 0.17 Mail   (JDE Abbrev)--L47--41%-----
cd ~/test/
javac -classpath ./:../:/opt/java_classes -sourcepath ./ -g -deprecation Client\
.java
-1:**  *compilation*   Thu Jun  8 4:29PM 0.17 Mail   (Compilation:exit [0])--L1-
```


You are the IDE
to my DSL



```
module blog

entity Blog {
  author : U
  url    : S
  name   : S
  posts  : S
}

entity Post {
  url    : String (id)
  title  : String (name)
  blog   : Blog (inverse:post)
  text   : WikiText
  author : User
```

URL
User

entity User
Press 'F2' for focus

4 errors, 1 warning, 0 others

Description	Resource	Location
Errors (4 items)		
Entity 'Blog' has no property 'post'	blog.nwl	line 13
Type 'U' is not defined	blog.nwl	line 7

Editor Services

syntactic editor services

- syntax highlighting
- syntax checking
- outline view
- bracket matching, insertion
- automatic indentation
- syntax completion
- ...

semantic editor services

- error marking
- reference resolving
- hover help
- mark occurrences
- content completion
- refactoring
- ...

Syntax definition

Static semantics

Model-to-model transformation

Code generation

Syntactic Editor Services

Semantic Editor Services

}

Language workbenches [Fowler ’05]

how can we make these things cheaply?

Language Workbench:

integrated environment for language definition

Holy Grail of Software Language Definition

Automatically derive *efficient, scalable, incremental* ***compiler*** + *usable* ***IDE*** *from* ***high-level, declarative language definition***

Stratego

SDF

Spoofax

Eclipse

IMP

Stratego

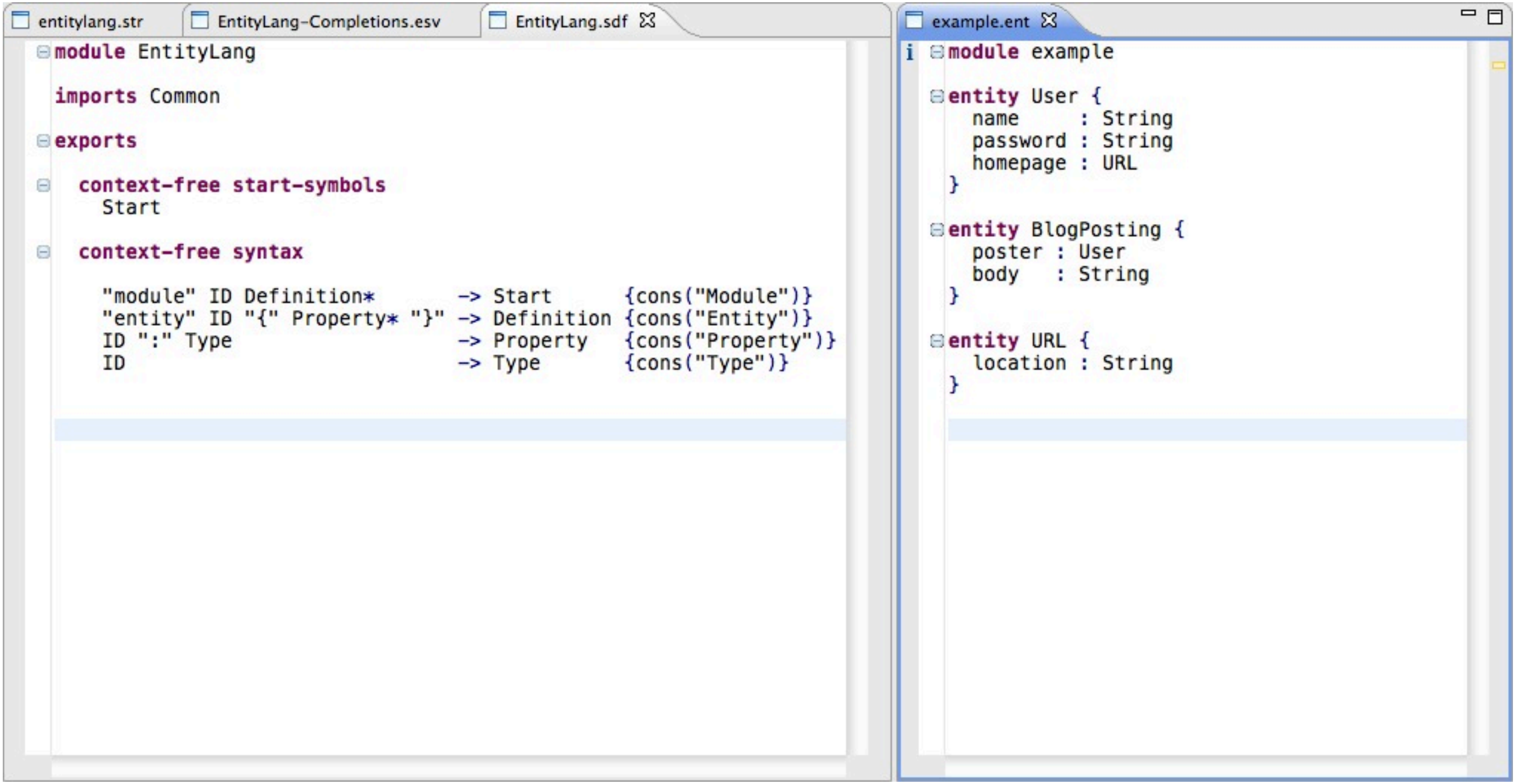
SDF

Eclipse

IMP

Language Definition by Transformation

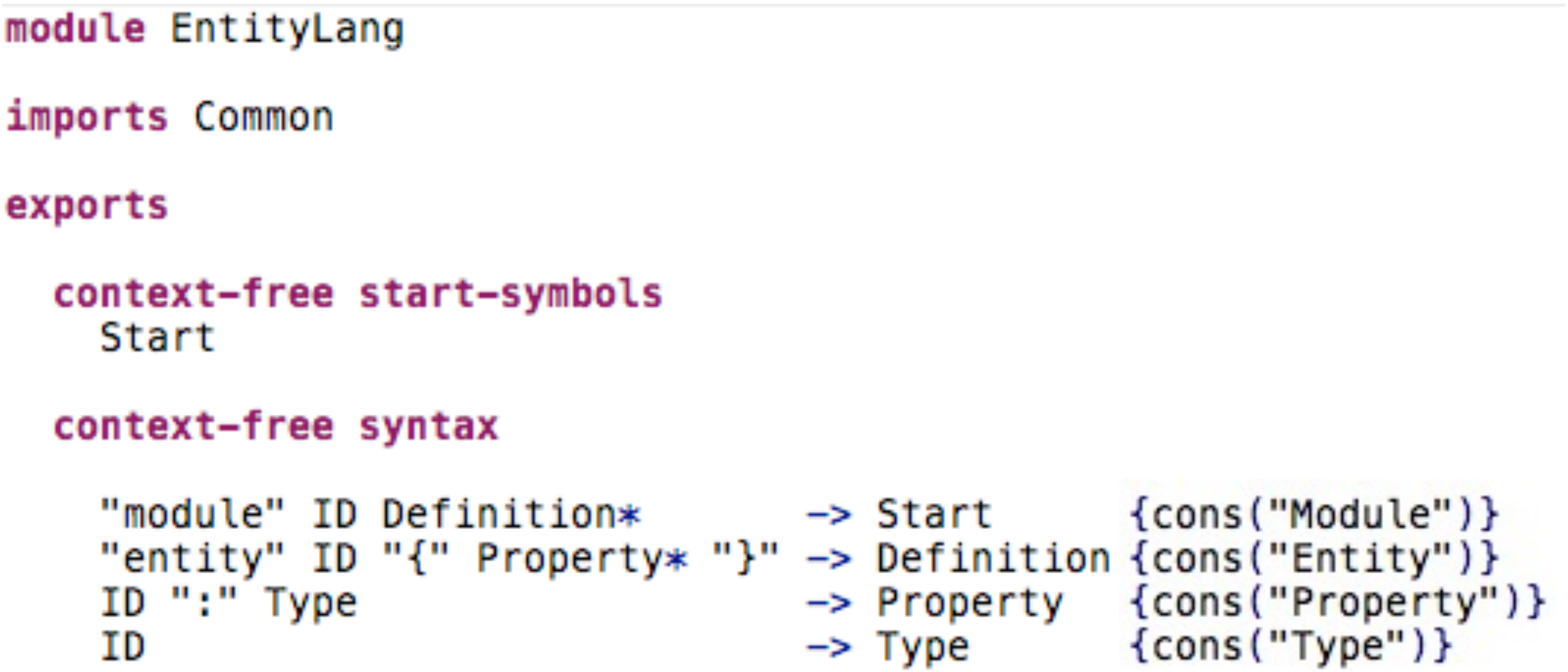
Spoofax



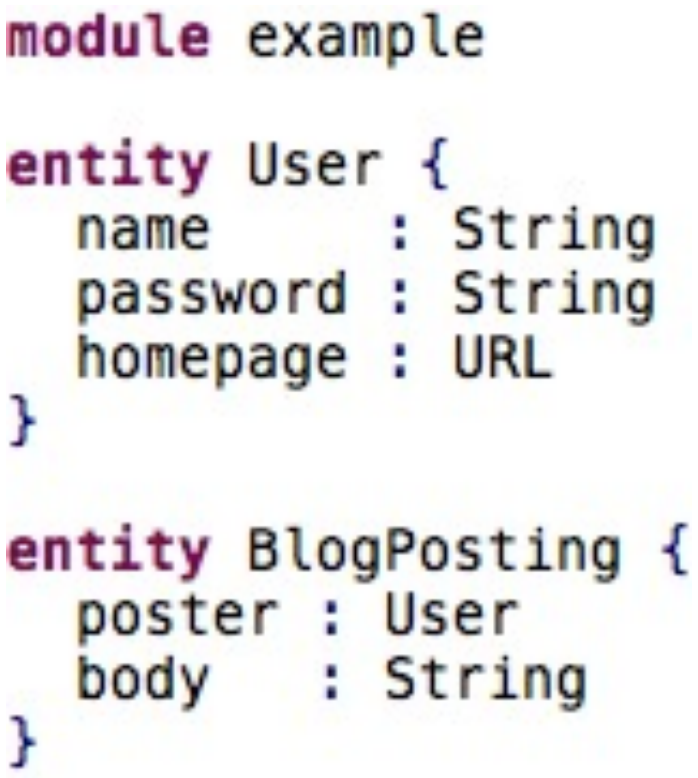
```
module EntityLang

imports Common

exports

  context-free start-symbols
    Start

  context-free syntax

    "module" ID Definition*        -> Start      {cons("Module")}
    "entity" ID "{" Property* "}"  -> Definition {cons("Entity")}
    ID ":" Type                    -> Property   {cons("Property")}
    ID                             -> Type       {cons("Type")}
```

```
module example

entity User {
  name     : String
  password : String
  homepage : URL
}

entity BlogPosting {
  poster : User
  body   : String
}

entity URL {
  location : String
}
```

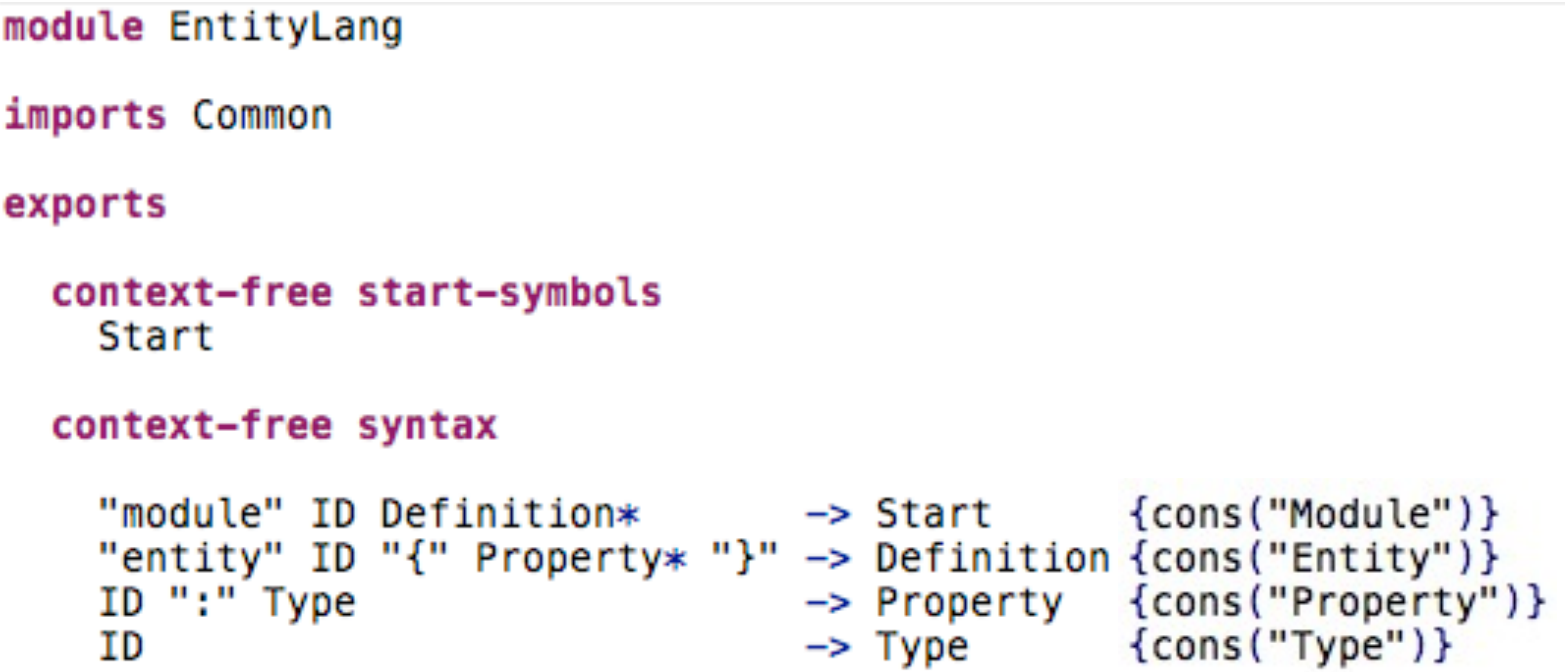
SDF: Declarative Syntax Definition



```
module EntityLang

imports Common

exports

  context-free start-symbols
    Start

  context-free syntax

    "module" ID Definition*         -> Start      {cons("Module")}
    "entity" ID "{" Property* "}"   -> Definition {cons("Entity")}
    ID ":" Type                     -> Property   {cons("Property")}
    ID                              -> Type       {cons("Type")}
```



```
module example

entity User {
  name     : String
  password : String
  homepage : URL
}

entity BlogPosting {
  poster : User
  body   : String
}
```



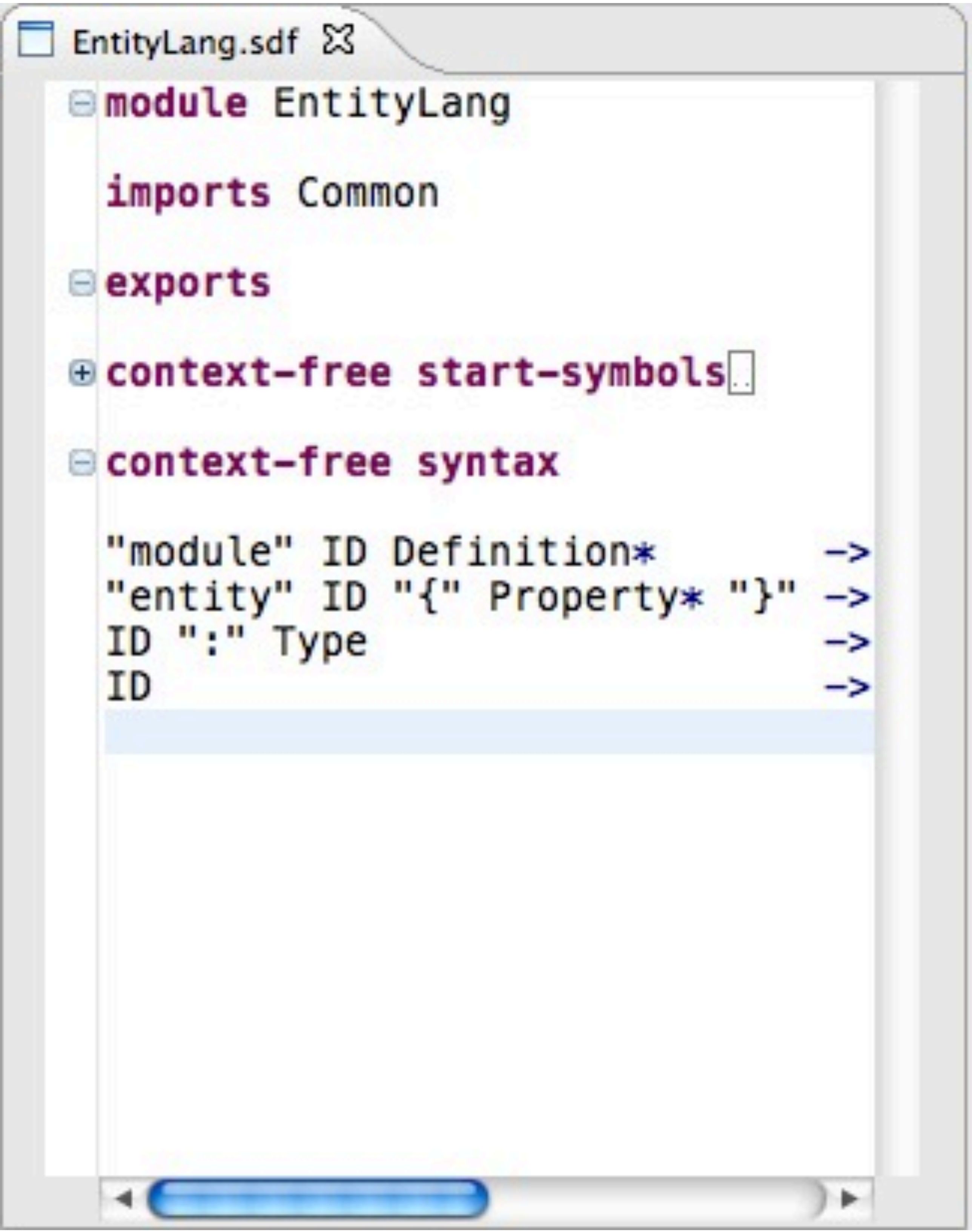
```
module EntityLang

imports Common

exports

  context-free start-symbols
    Start

  context-free syntax

    "module" ID Definition*       -> Start      {cons("Module")}
    "entity" ID "{" Property* "}" -> Definition {cons("Entity")}
    ID ":" Type                   -> Property   {cons("Property")}
    ID                            -> Type       {cons("Type")}
```

A **syntax** definition

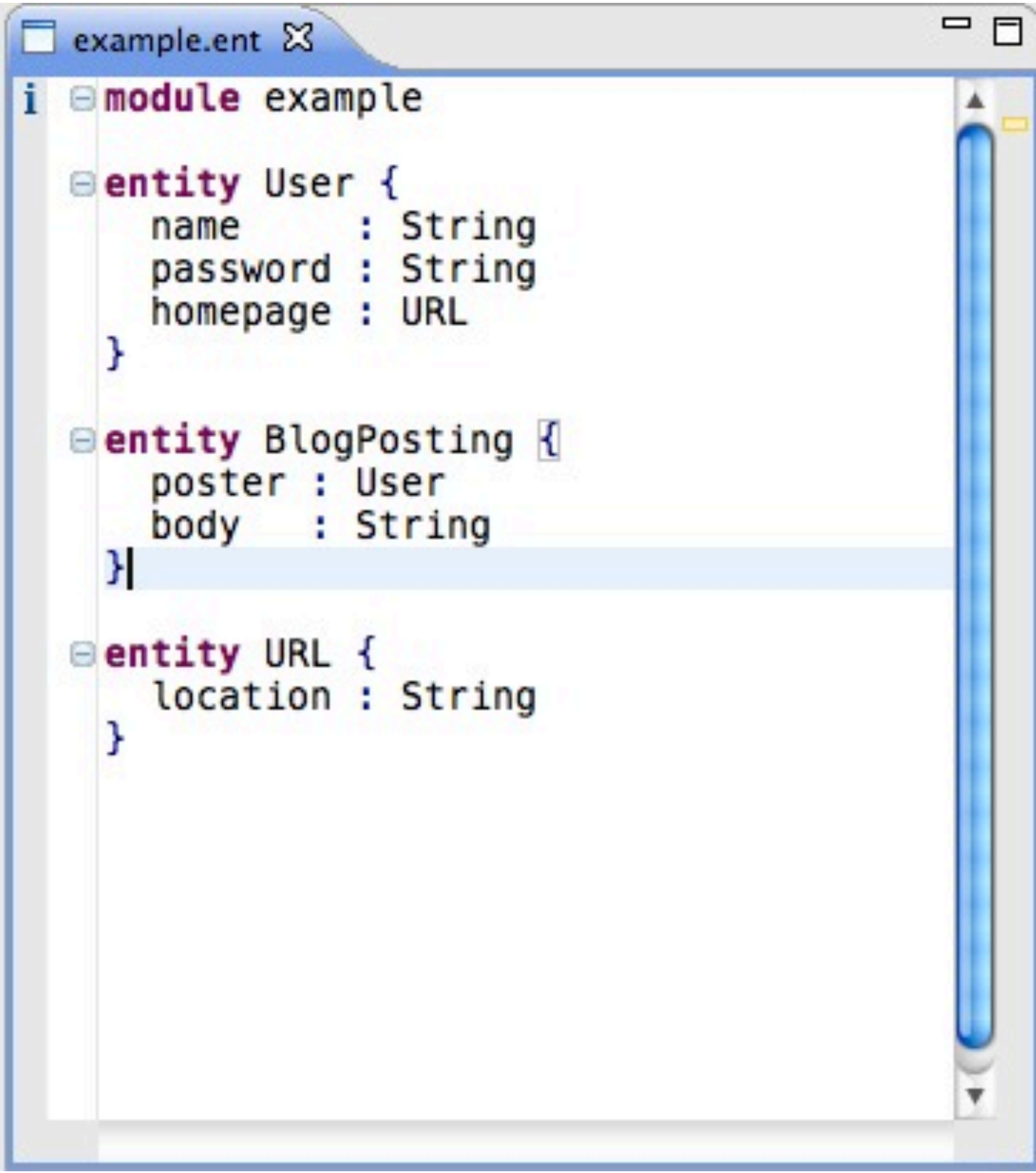
specifies a ***transformation***

from *text to trees*

Syntax as Transformation

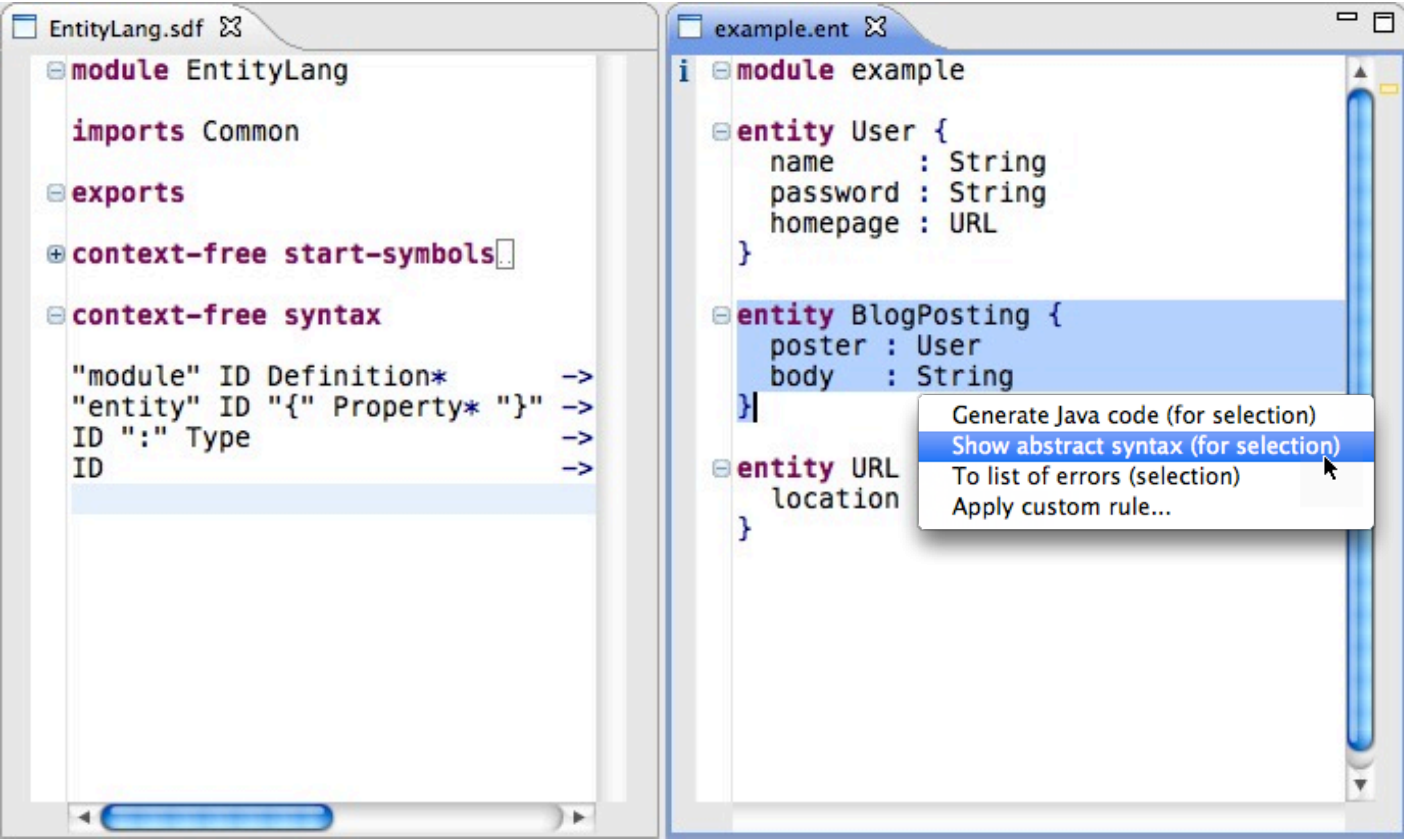


```
module EntityLang

imports Common

exports

context-free start-symbols

context-free syntax

"module" ID Definition*        ->
"entity" ID "{" Property* "}"  ->
ID ":" Type                    ->
ID                             ->
```



```
module example

entity User {
  name     : String
  password : String
  homepage : URL
}

entity BlogPosting {
  poster : User
  body   : String
}

entity URL {
  location : String
}
```

Syntax as Transformation

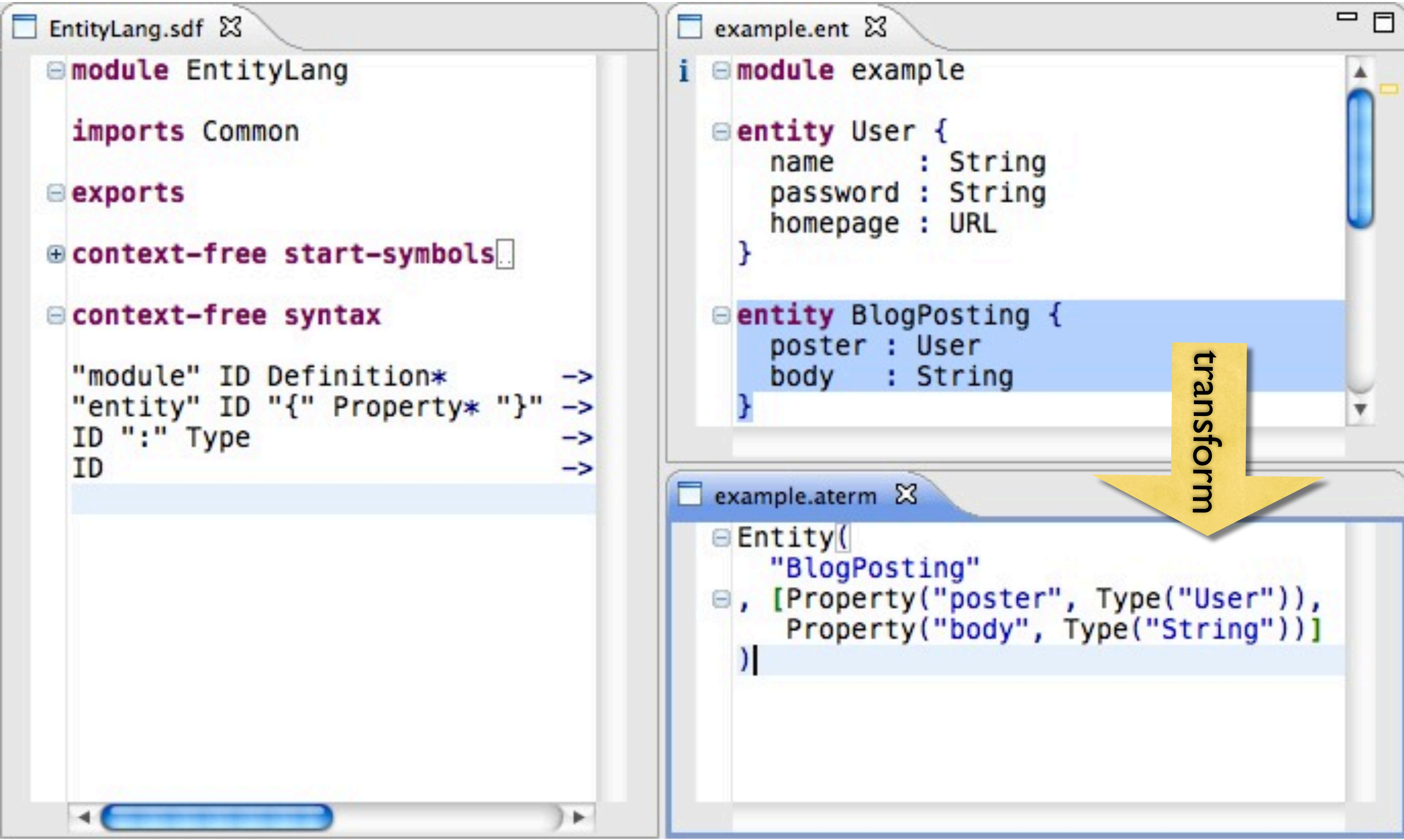

EntityLang.sdf
example.ent

```
module EntityLang

imports Common

exports

context-free start-symbols

context-free syntax

"module" ID Definition*        ->
"entity" ID "{" Property* "}"  ->
ID ":" Type                    ->
ID                             ->
```

```
module example

entity User {
  name     : String
  password : String
  homepage : URL
}

entity BlogPosting {
  poster : User
  body   : String
}

entity URL
  location
}
```

Generate Java code (for selection)
Show abstract syntax (for selection)
To list of errors (selection)
Apply custom rule...

Syntax as Transformation

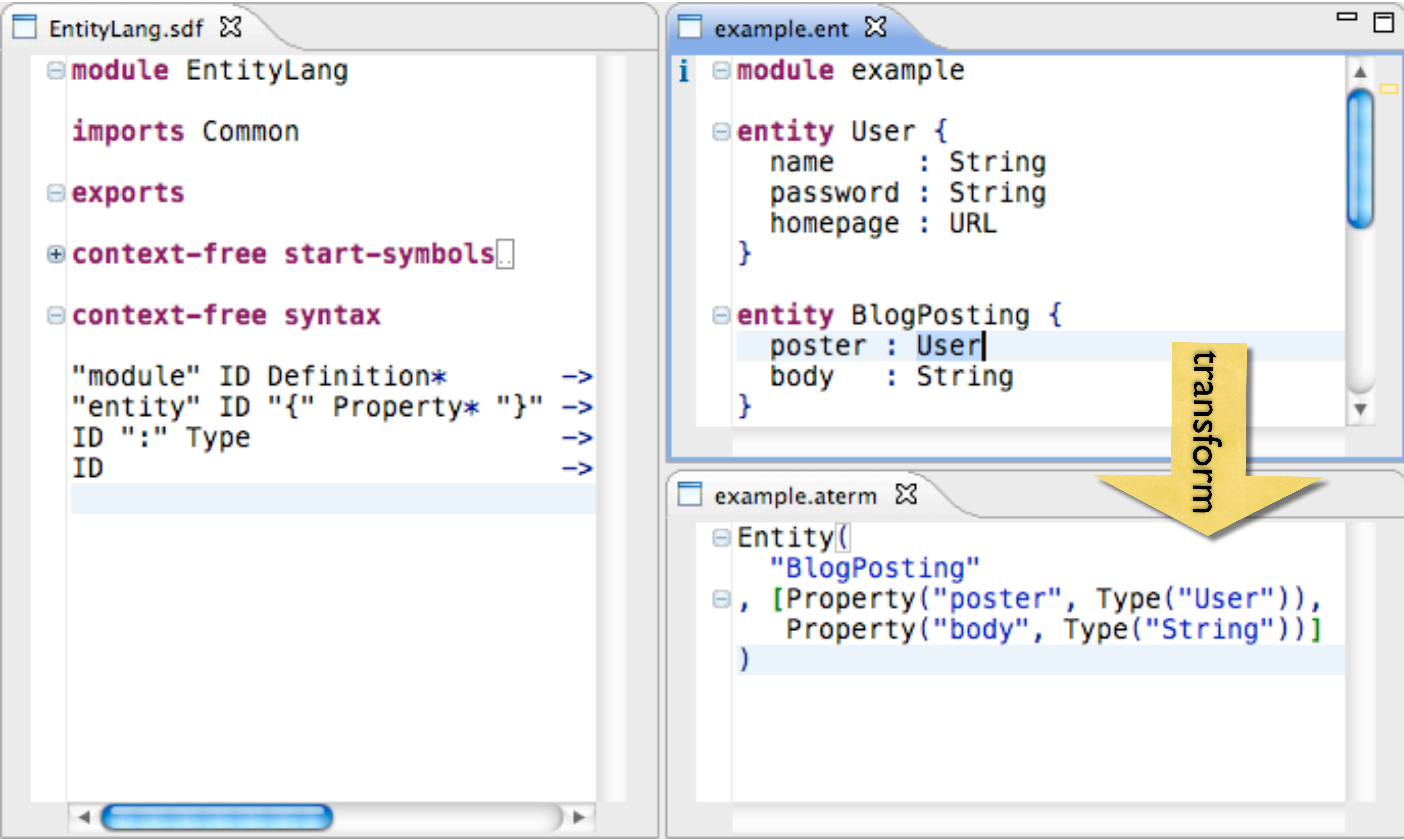


```
module EntityLang

imports Common

exports

context-free start-symbols

context-free syntax

"module" ID Definition*           ->
"entity" ID "{" Property* "}"     ->
ID ":" Type                       ->
ID                                ->
```

```
module example

entity User {
  name     : String
  password : String
  homepage : URL
}

entity BlogPosting {
  poster : User
  body   : String
}
```

```
Entity(
  "BlogPosting"
, [Property("poster", Type("User")),
   Property("body", Type("String"))]
)
```

Syntax as Transformation

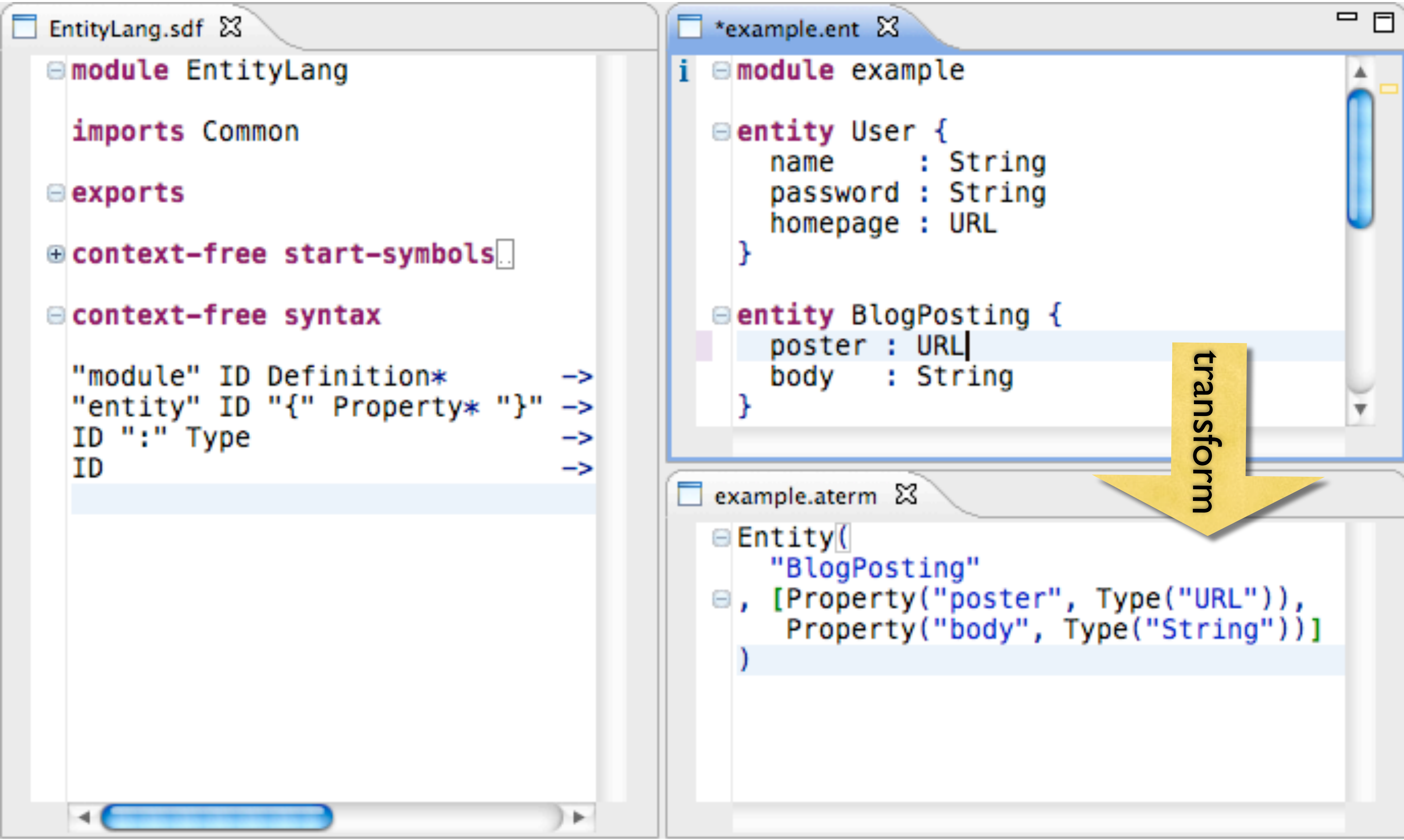


```
module EntityLang

imports Common

exports

context-free start-symbols

context-free syntax

"module" ID Definition*       ->
"entity" ID "{" Property* "}" ->
ID ":" Type                   ->
ID                            ->
```

```
module example

entity User {
  name     : String
  password : String
  homepage : URL
}

entity BlogPosting {
  poster : User
  body   : String
}
```

```
Entity(
  "BlogPosting"
, [Property("poster", Type("User")),
   Property("body", Type("String"))]
)
```

Syntax as Transformation



```
module EntityLang

imports Common

exports

context-free start-symbols

context-free syntax

"module" ID Definition*       ->
"entity" ID "{" Property* "}" ->
ID ":" Type                   ->
ID                            ->
```

```
module example

entity User {
  name     : String
  password : String
  homepage : URL
}

entity BlogPosting {
  poster : URL
  body   : String
}
```

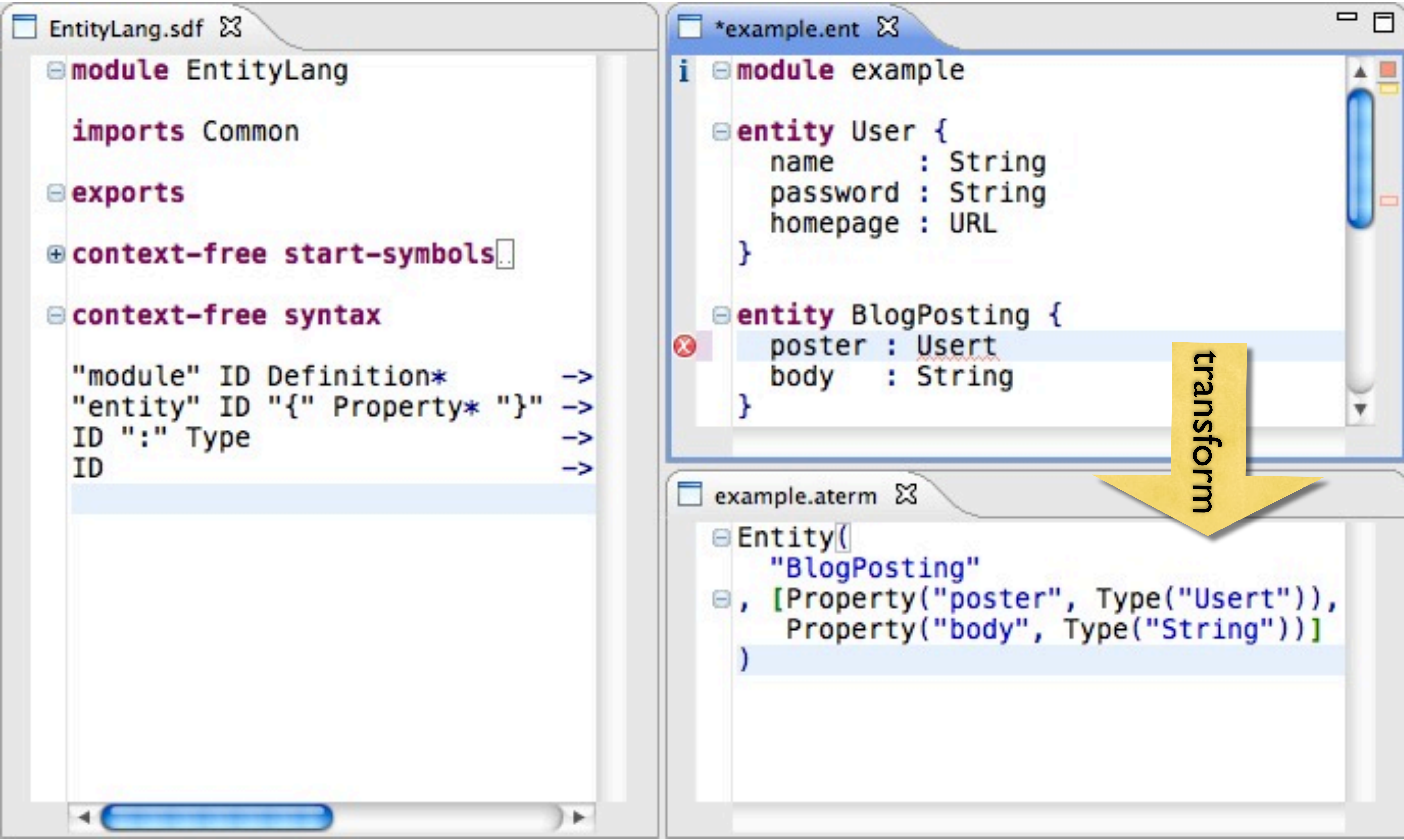
```
Entity(
  "BlogPosting"
, [Property("poster", Type("URL")),
   Property("body", Type("String"))]
)
```

Semantics

=

transformation

Error Marking is a Transformation

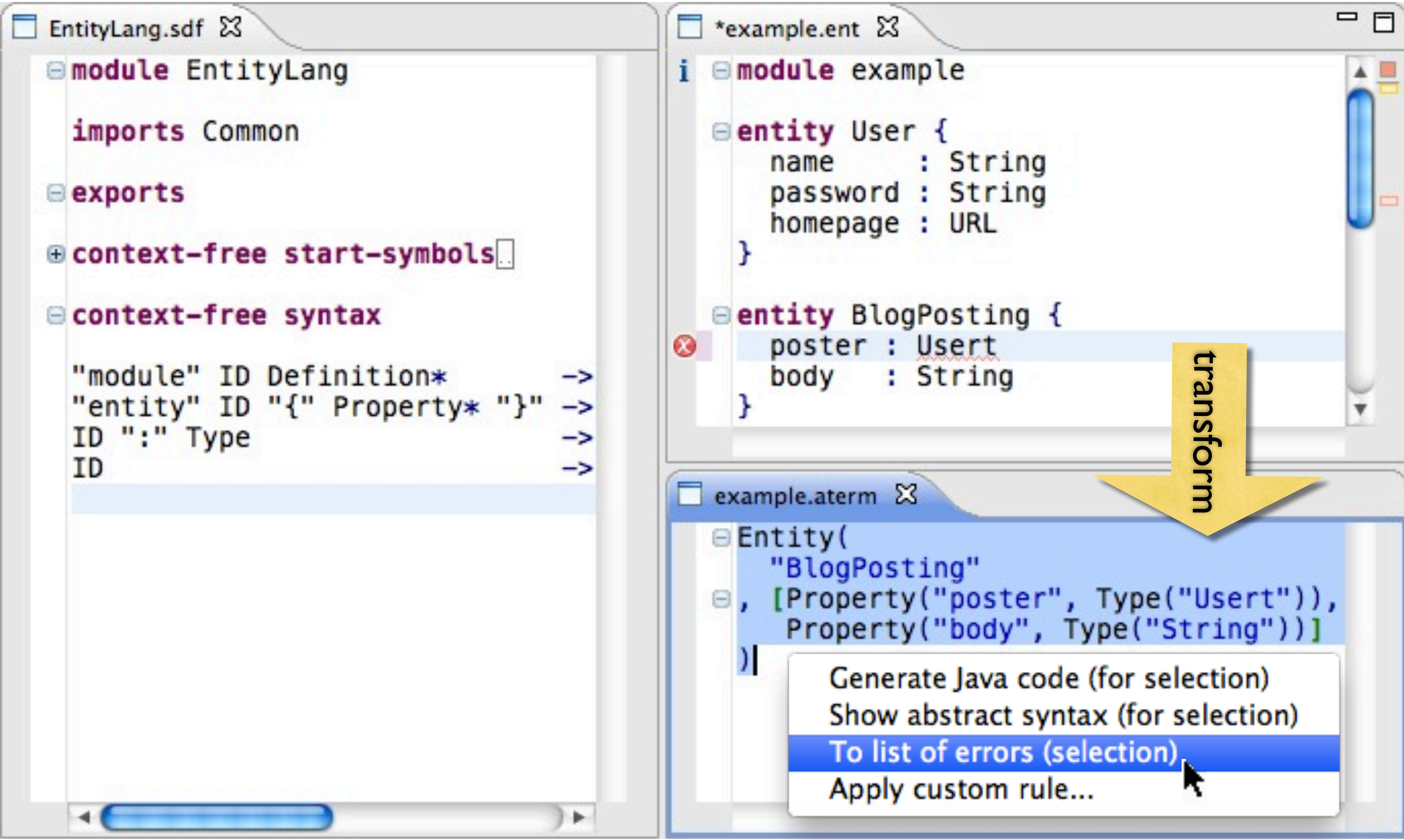


```
module EntityLang

imports Common

exports

context-free start-symbols

context-free syntax

"module" ID Definition*         ->
"entity" ID "{" Property* "}"   ->
ID ":" Type                     ->
ID                              ->
```

```
module example

entity User {
  name     : String
  password : String
  homepage : URL
}

entity BlogPosting {
  poster : Usert
  body   : String
}
```

```
Entity(
  "BlogPosting"
, [Property("poster", Type("Usert")),
   Property("body", Type("String"))]
)
```

Error Marking is a Transformation

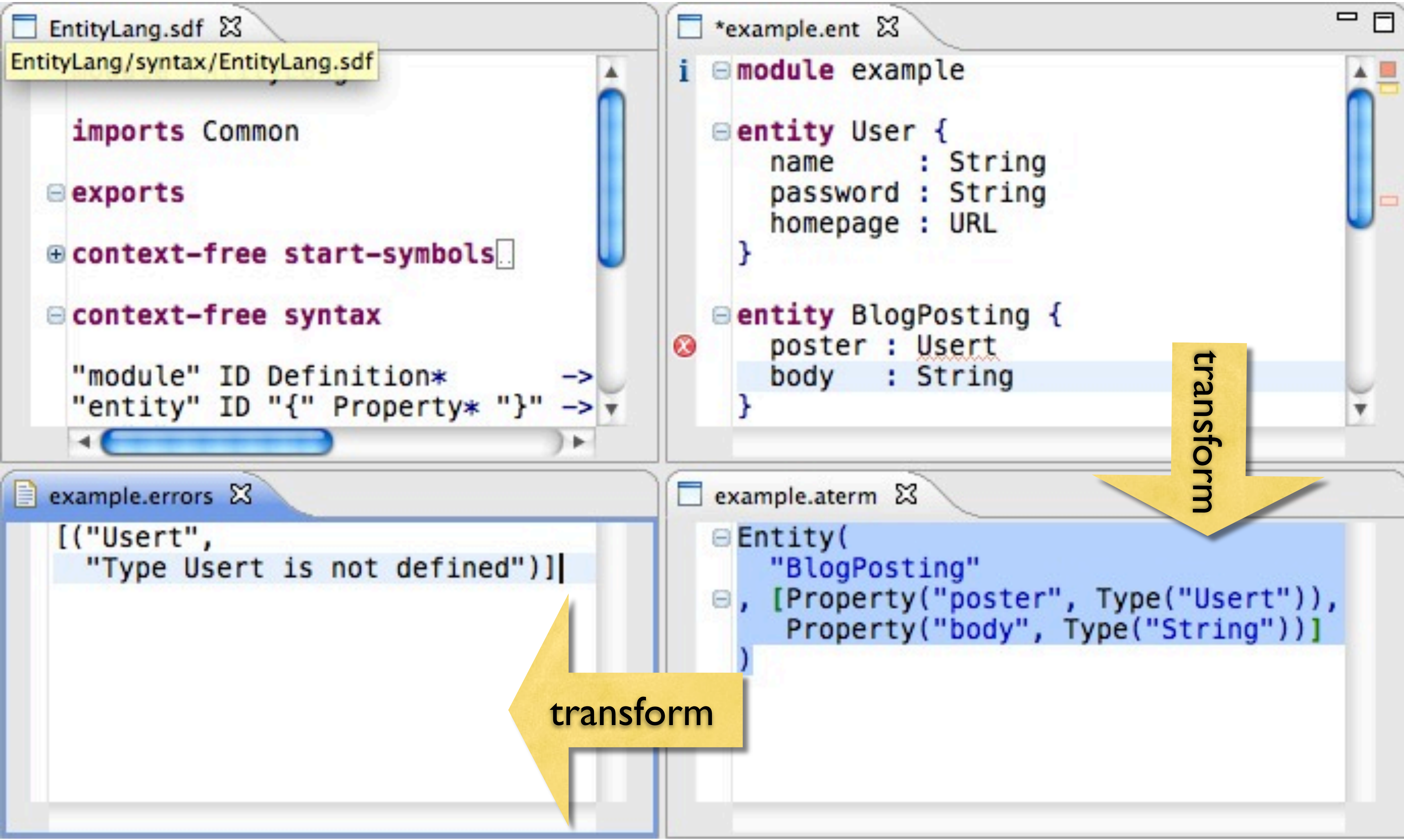

EntityLang.sdf
*example.ent
example.aterm
transform
Generate Java code (for selection)
Show abstract syntax (for selection)
To list of errors (selection)
Apply custom rule...

```
module EntityLang

imports Common

exports

context-free start-symbols

context-free syntax

"module" ID Definition*            ->
"entity" ID "{" Property* "}"      ->
ID ":" Type                        ->
ID                                 ->
```

```
module example

entity User {
  name     : String
  password : String
  homepage : URL
}

entity BlogPosting {
  poster : Usert
  body   : String
}
```

```
Entity(
  "BlogPosting"
, [Property("poster", Type("Usert")),
   Property("body", Type("String"))]
)
```

Error Marking is a Transformation



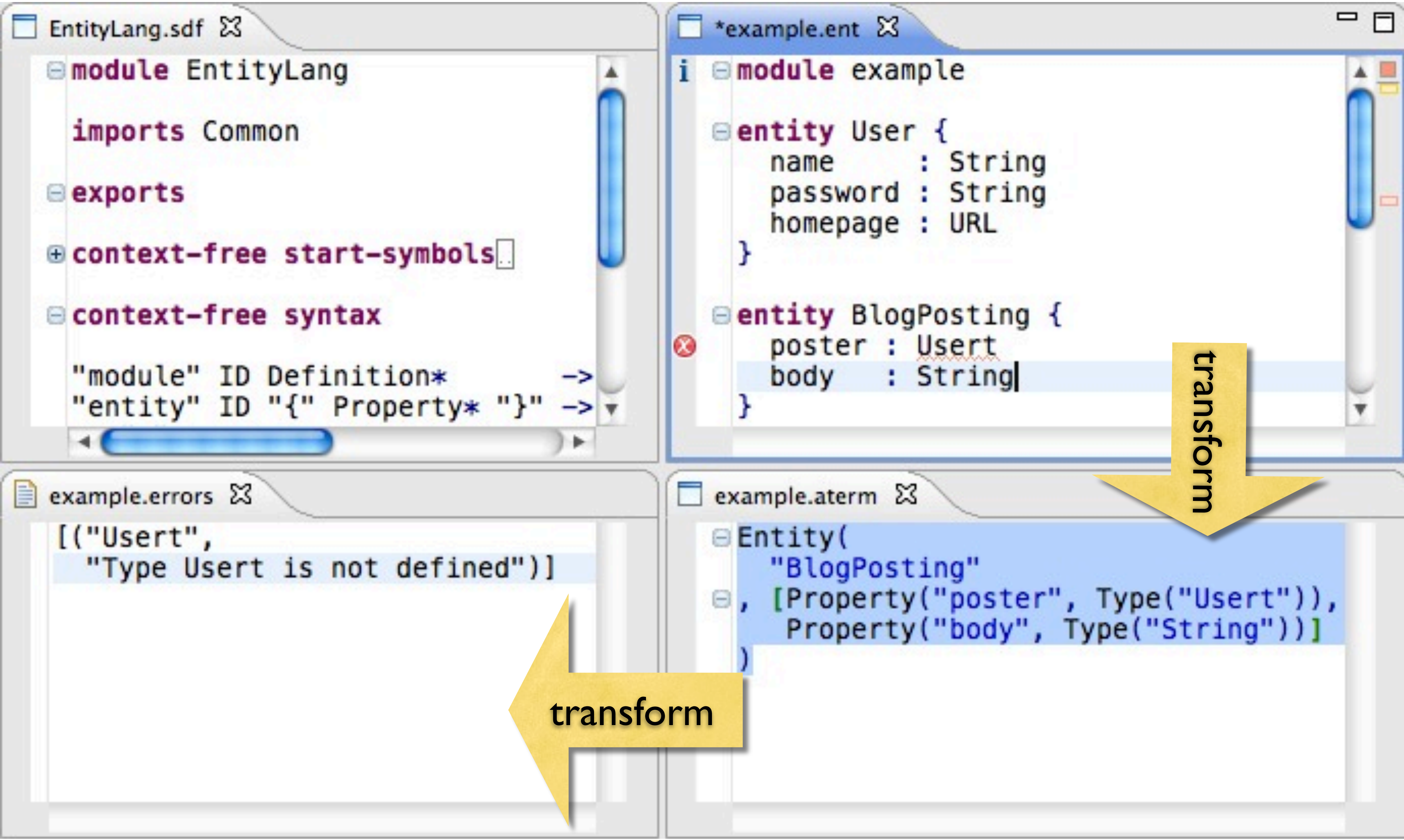
```
imports Common

exports

context-free start-symbols

context-free syntax

"module" ID Definition*        ->
"entity" ID "{" Property* "}"  ->
```

```
module example

entity User {
  name     : String
  password : String
  homepage : URL
}

entity BlogPosting {
  poster : Usert
  body   : String
}
```

```
[("Usert",
  "Type Usert is not defined")]
```

```
Entity(
  "BlogPosting"
, [Property("poster", Type("Usert")),
   Property("body", Type("String"))]
)
```

Error Marking is a Transformation



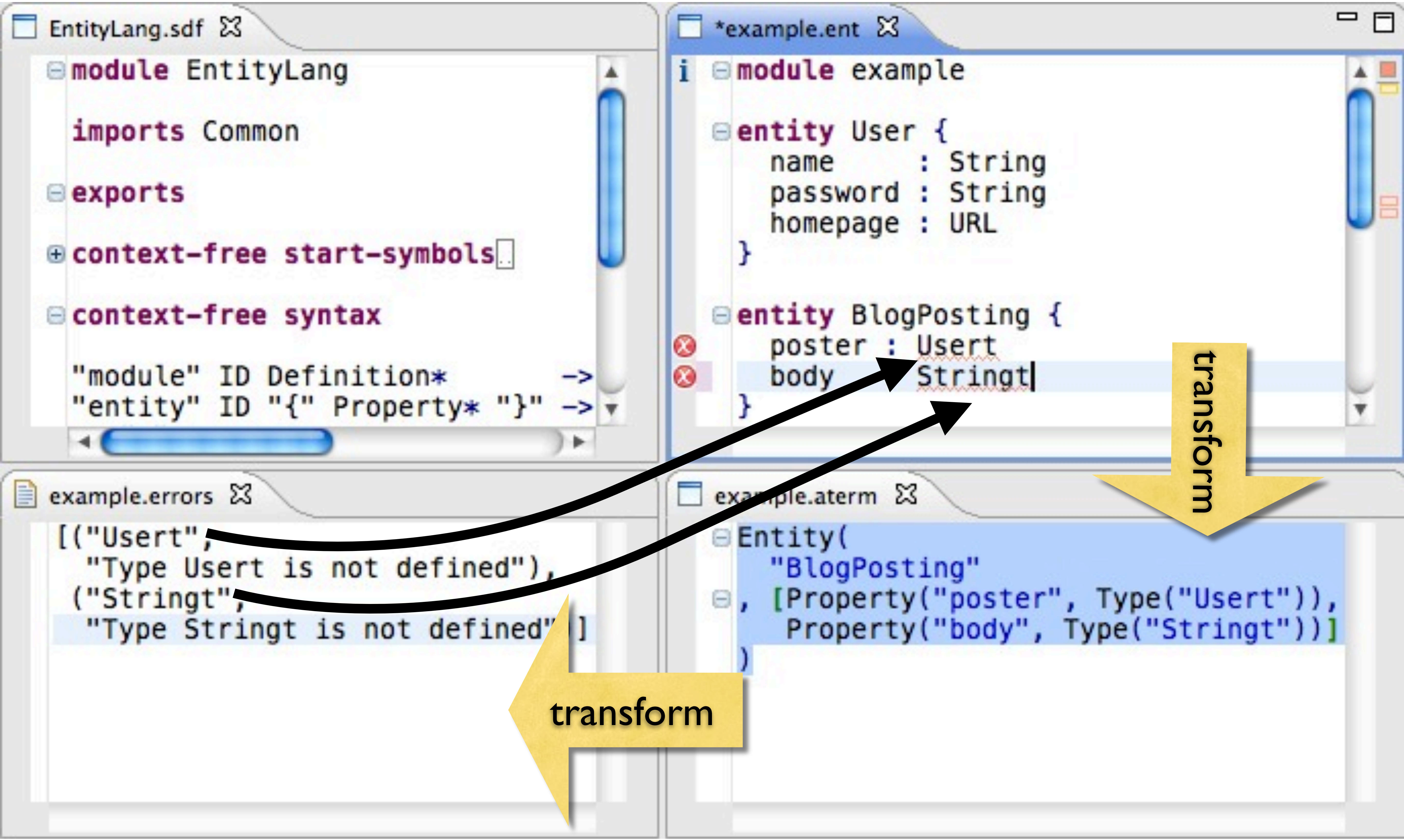
```
module EntityLang

imports Common

exports

context-free start-symbols

context-free syntax

"module" ID Definition*          ->
"entity" ID "{" Property* "}"  ->
```

```
module example

entity User {
  name     : String
  password : String
  homepage : URL
}

entity BlogPosting {
  poster : Usert
  body   : String
}
```

```
[("Usert",
  "Type Usert is not defined")]
```

```
Entity(
  "BlogPosting"
, [Property("poster", Type("Usert")),
   Property("body", Type("String"))]
)
```

Error Marking is a Transformation



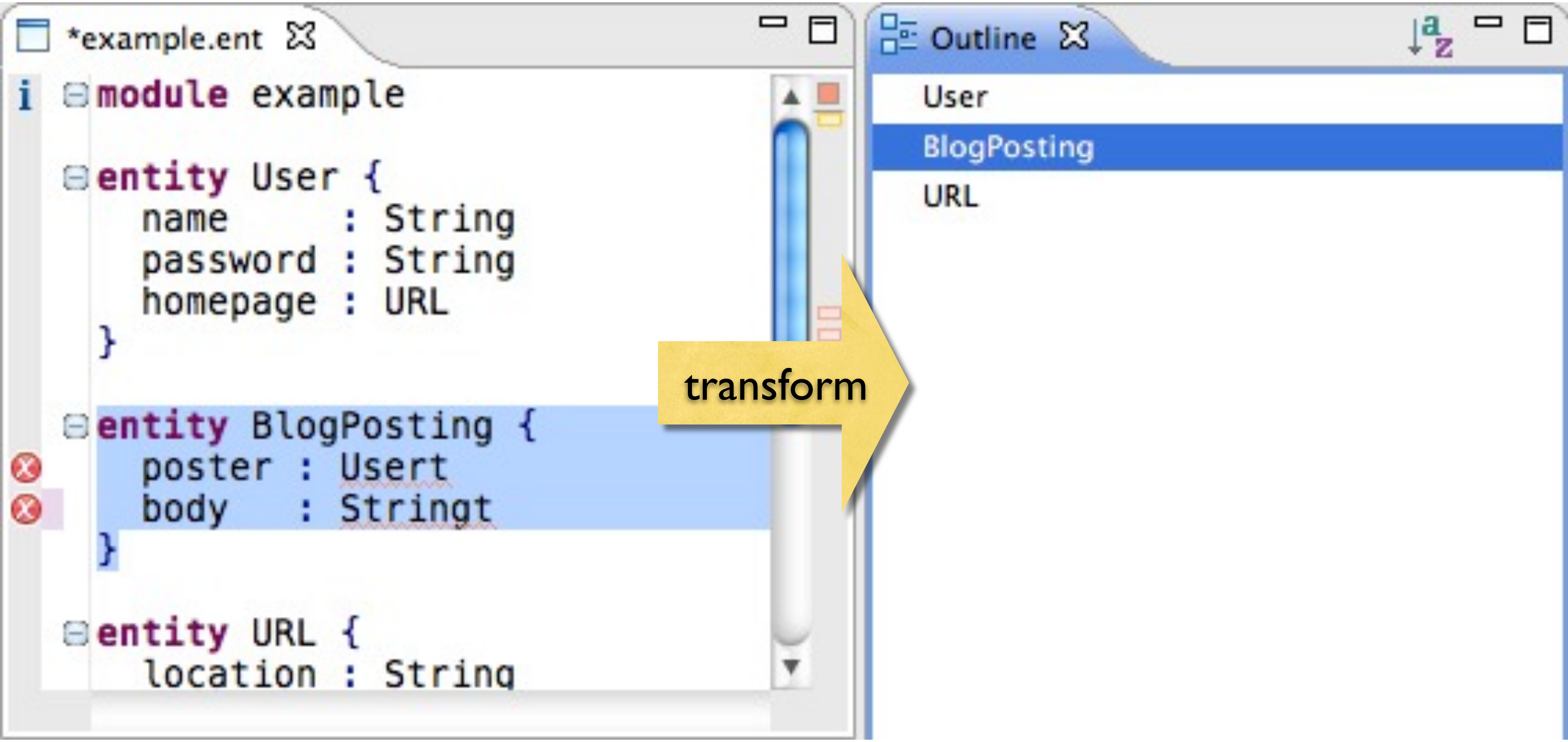
```
module EntityLang

imports Common

exports

context-free start-symbols

context-free syntax

"module" ID Definition*        ->
"entity" ID "{" Property* "}"  ->
```

```
module example

entity User {
  name     : String
  password : String
  homepage : URL
}

entity BlogPosting {
  poster : Usert
  body   : Stringt
}
```

```
[("Usert",
  "Type Usert is not defined"),
 ("Stringt",
  "Type Stringt is not defined")]
```

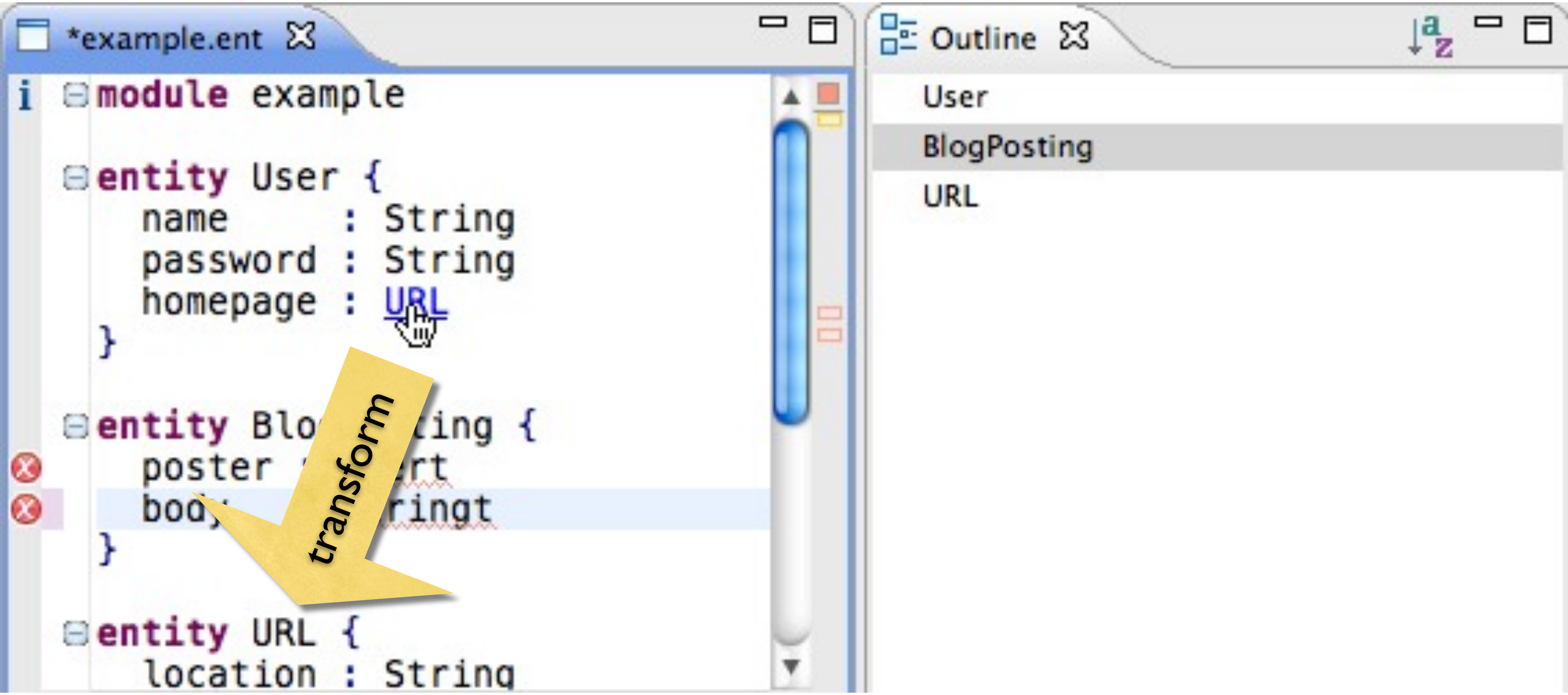
```
Entity(
  "BlogPosting"
, [Property("poster", Type("Usert")),
   Property("body", Type("Stringt"))]
)
```

Outline View is a Transformation



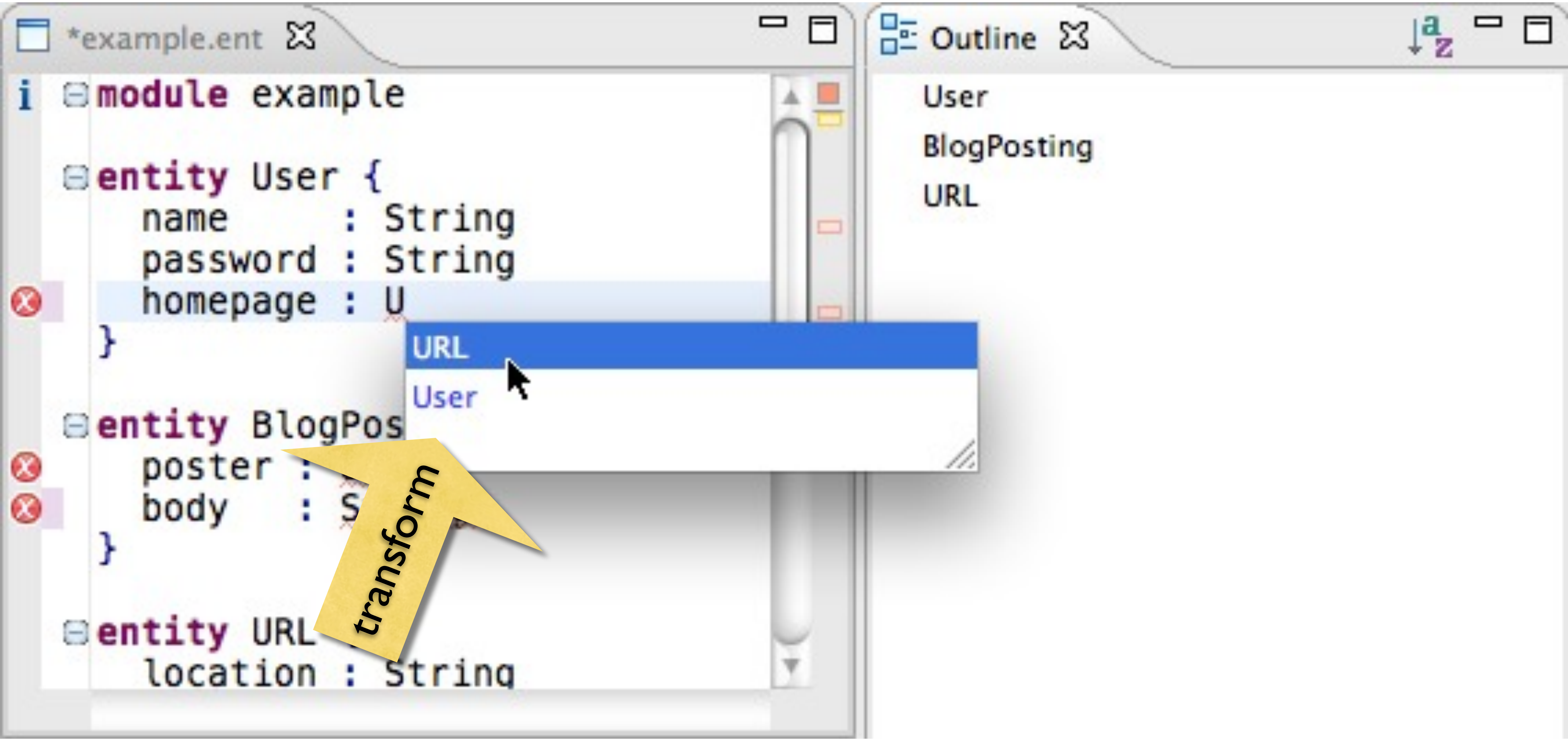
```
module example

entity User {
   name     : String
   password : String
   homepage : URL
}

entity BlogPosting {
   poster : Usert
   body   : Stringt
}

entity URL {
   location : String
```

Hyperlinking is a Transformation



```
module example

entity User {
  name     : String
  password : String
  homepage : URL
}

entity Blo     ing {
  poster      rt
  bod       ringt
}

entity URL {
  location : String
```

Content Completion is a Transformation

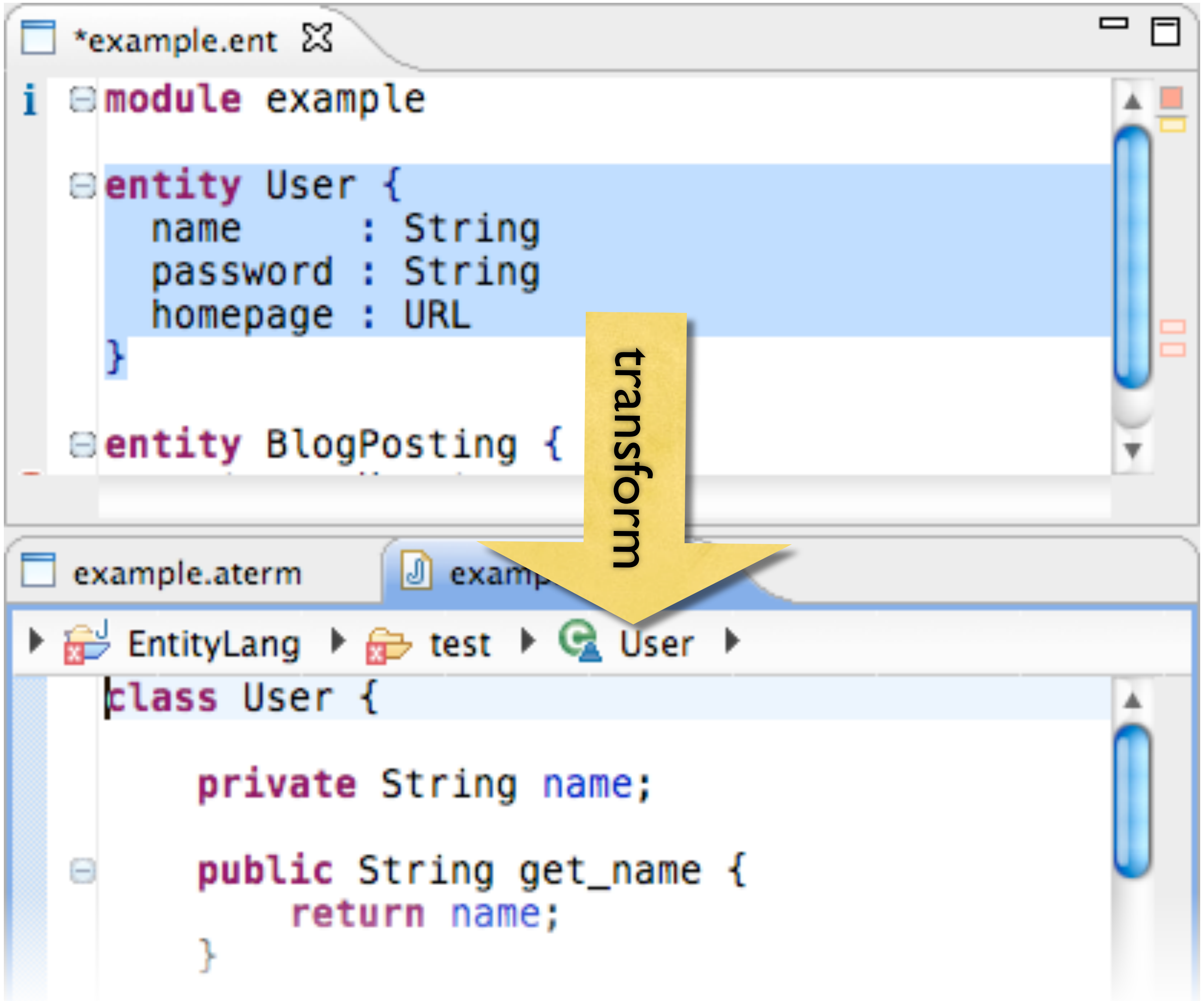


```
module example

entity User {
  name     : String
  password : String
  homepage : U
}

entity BlogPos
  poster :
  body   : S
}

entity URL
  location : String
```

URL
User

User
BlogPosting
URL

Code Generation is a Transformation



```
module example

entity User {
  name     : String
  password : String
  homepage : URL
}

entity BlogPosting {
```

```
class User {

    private String name;

    public String get_name {
        return name;
    }
```

Need for single, unified language specification:

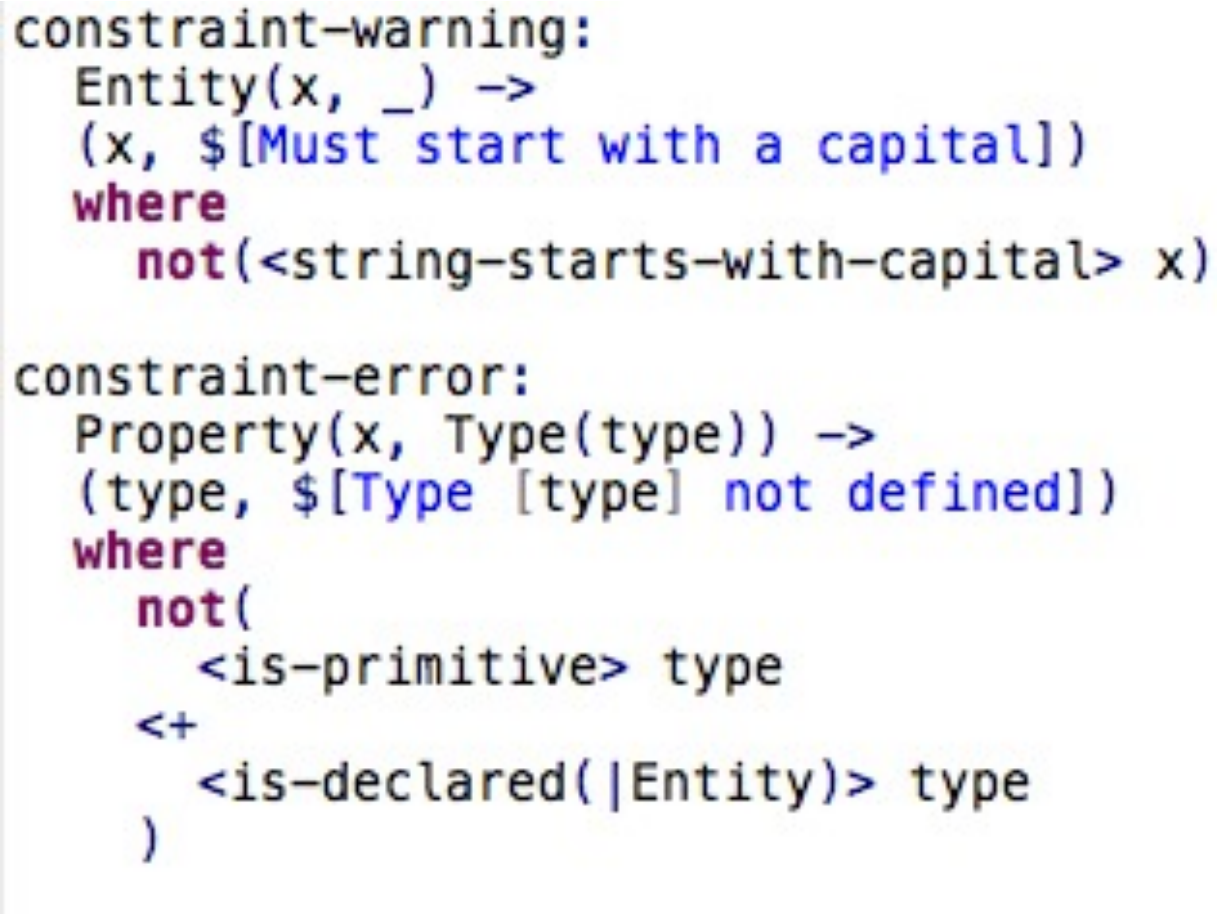
- Editor services
- Model transformations
- Code generation

Stratego: Rewriting Language

Rewrite rules

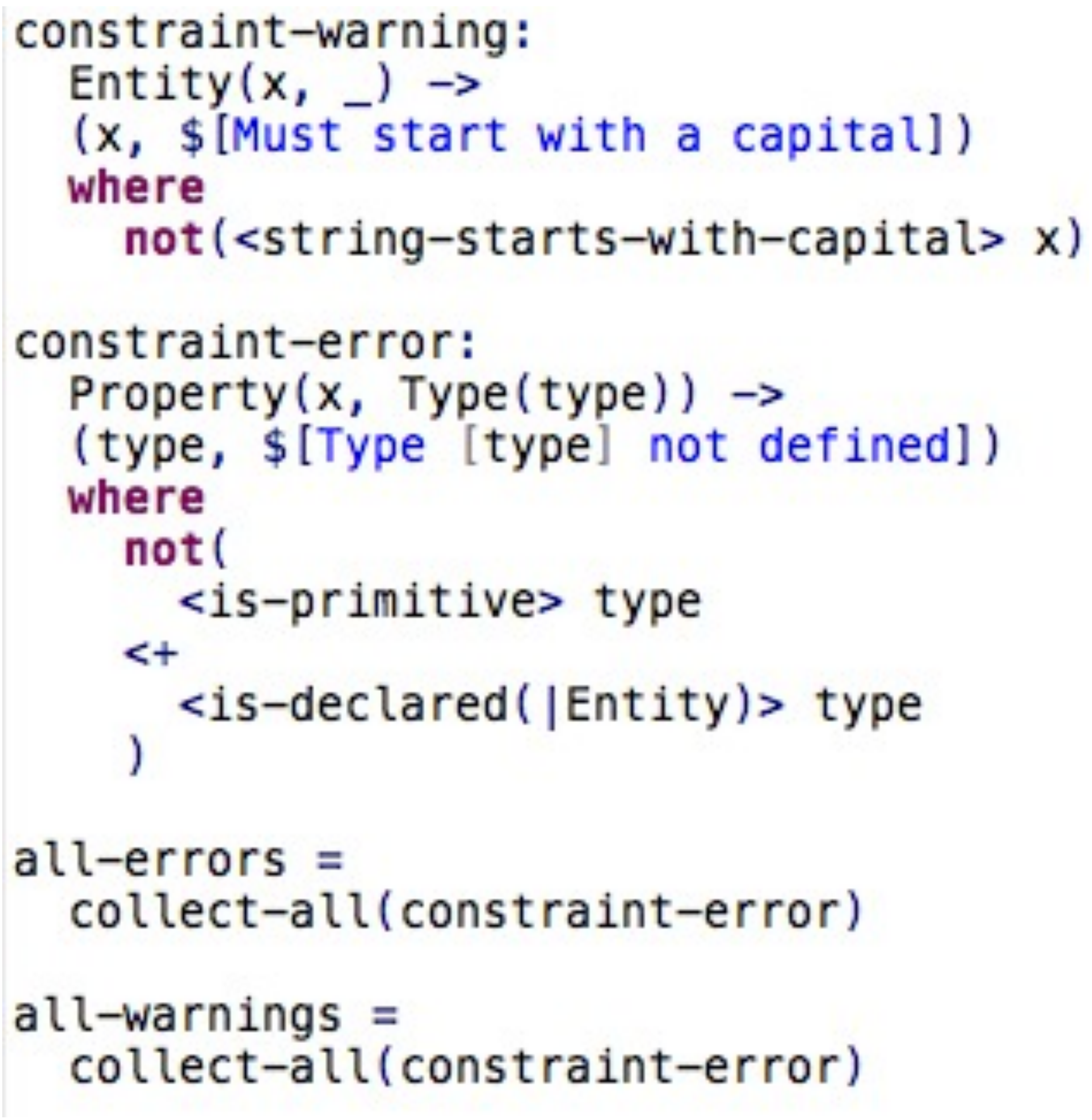
Strategies

Error Marking with Rewrite Rules



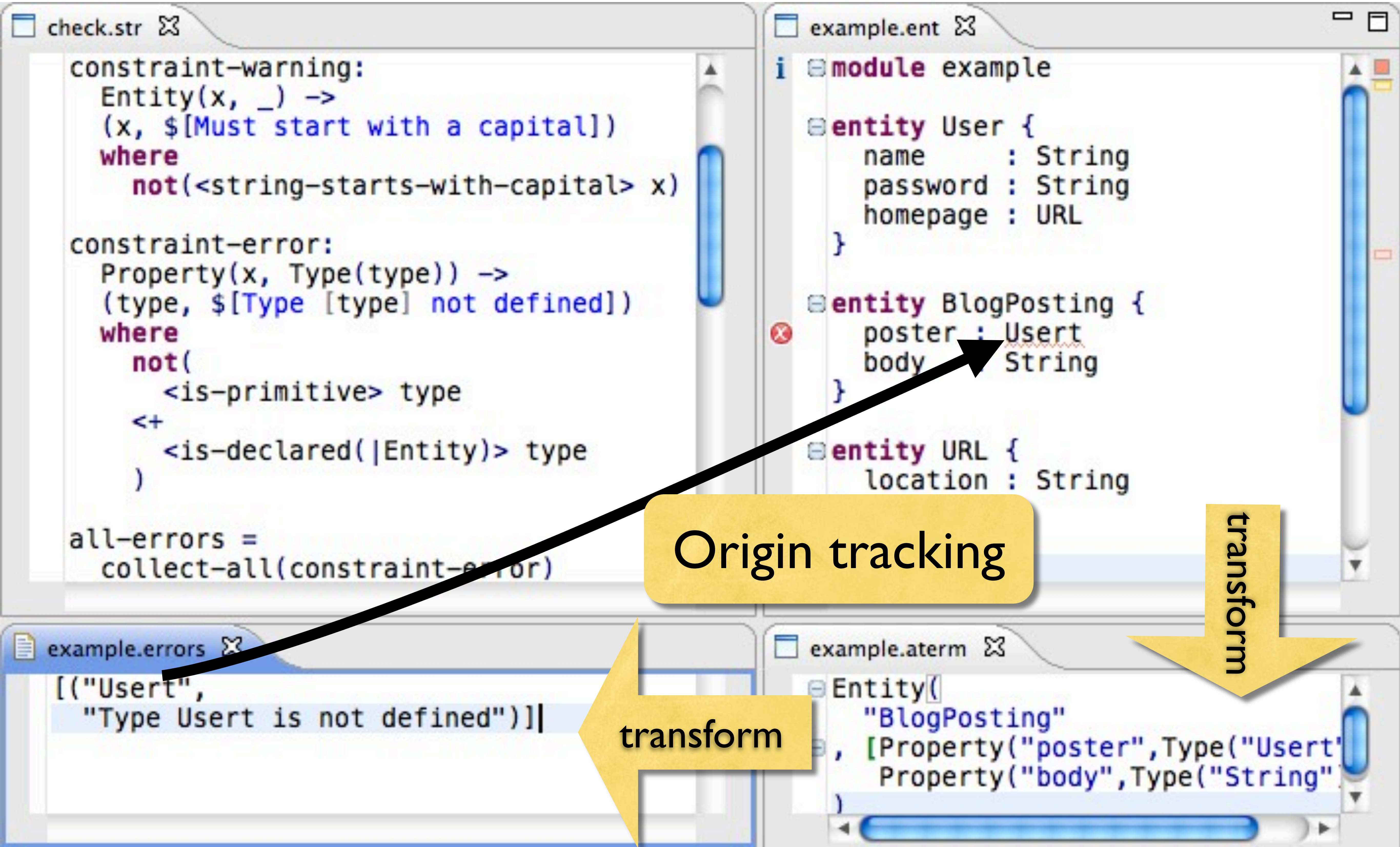
```
constraint-warning:
  Entity(x, _) ->
  (x, $[Must start with a capital])
  where
    not(<string-starts-with-capital> x)

constraint-error:
  Property(x, Type(type)) ->
  (type, $[Type [type] not defined])
  where
    not(
      <is-primitive> type
    <+
      <is-declared(|Entity)> type
    )
```

Error Marking with Rewrite Rules



```
constraint-warning:
  Entity(x, _) ->
  (x, $[Must start with a capital])
  where
    not(<string-starts-with-capital> x)

constraint-error:
  Property(x, Type(type)) ->
  (type, $[Type [type] not defined])
  where
    not(
      <is-primitive> type
    <+
      <is-declared(|Entity)> type
    )

all-errors =
  collect-all(constraint-error)

all-warnings =
  collect-all(constraint-error)
```

Error Marking with Rewrite Rules



check.str

```
constraint-warning:
  Entity(x, _) ->
  (x, $[Must start with a capital])
  where
    not(<string-starts-with-capital> x)

constraint-error:
  Property(x, Type(type)) ->
  (type, $[Type [type] not defined])
  where
    not(
      <is-primitive> type
    <+
      <is-declared(|Entity)> type
    )

all-errors =
  collect-all(constraint-error)
```

example.ent

```
module example

entity User {
  name     : String
  password : String
  homepage : URL
}

entity BlogPosting {
  poster : Usert
  body   : String
}

entity URL {
  location : String
```

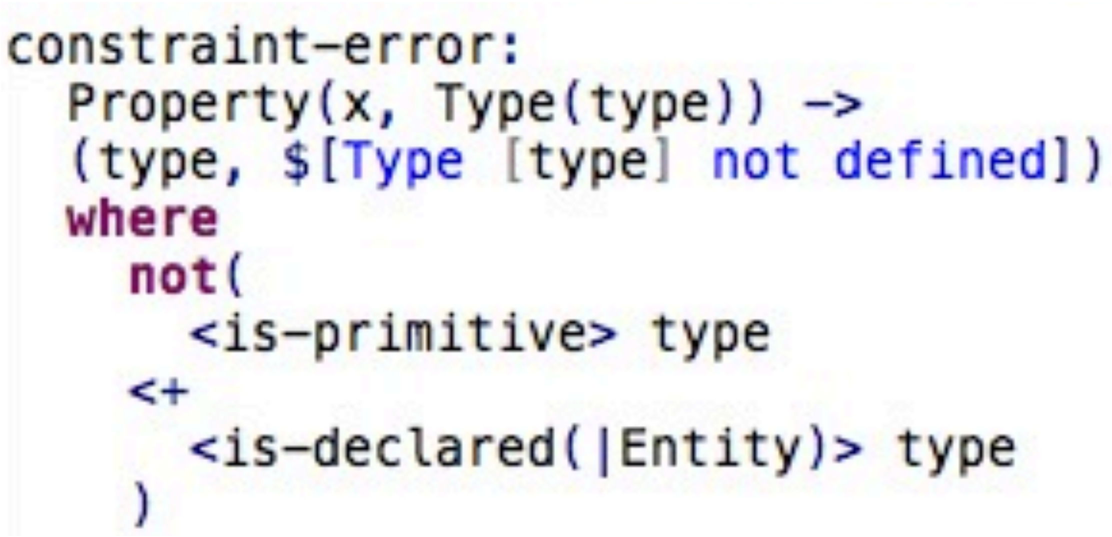
example.errors

```
[("Usert",
  "Type Usert is not defined")]
```

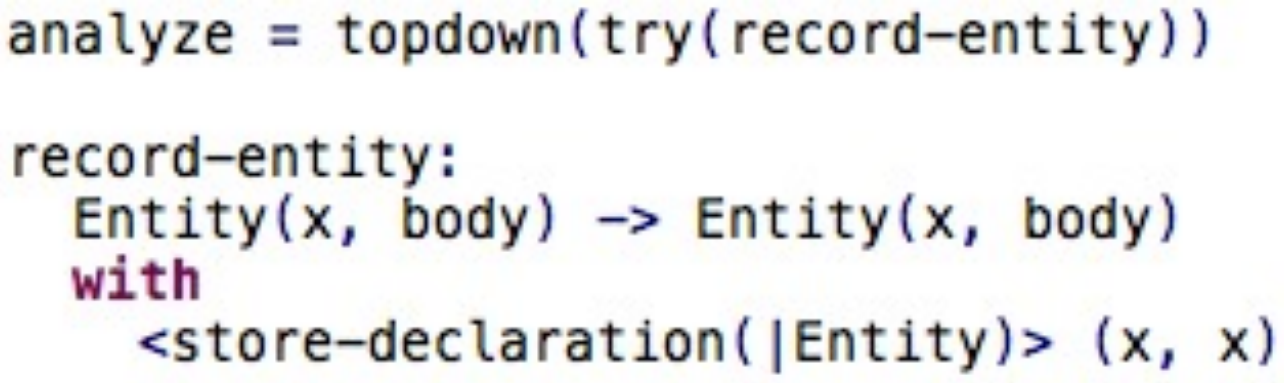
example.aterm

```
Entity(
  "BlogPosting"
, [Property("poster",Type("Usert"
   Property("body",Type("String")
)
```

Analysis with Rewrite Rules

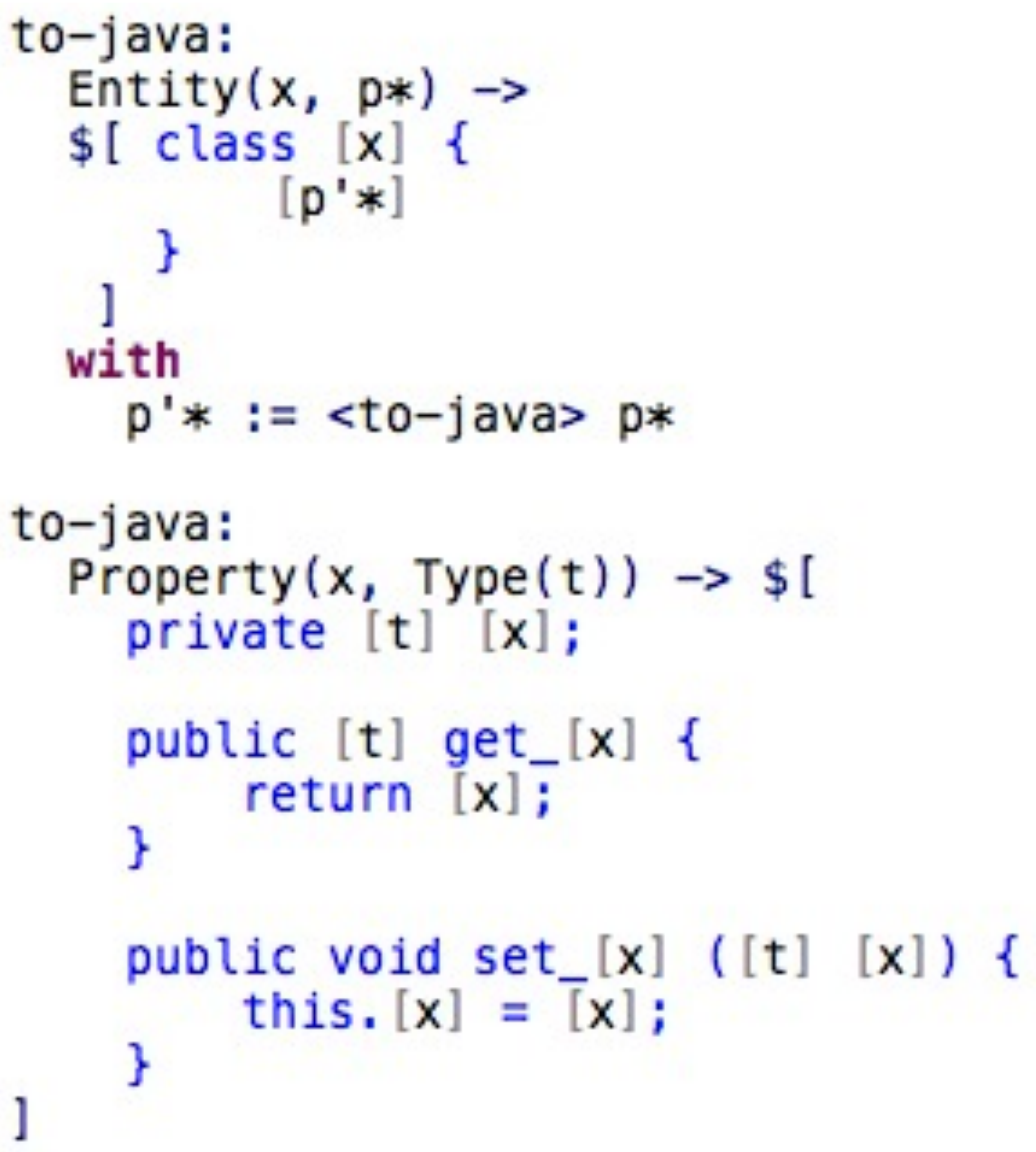


```
constraint-error:
  Property(x, Type(type)) ->
  (type, $[Type [type] not defined])
  where
    not(
      <is-primitive> type
    <+
      <is-declared(|Entity)> type
    )
```



```
analyze = topdown(try(record-entity))

record-entity:
  Entity(x, body) -> Entity(x, body)
  with
    <store-declaration(|Entity)> (x, x)
```

Code Generation with Rewrite Rules



```
to-java:
  Entity(x, p*) ->
  $[ class [x] {
         [p'*]
     }
   ]
  with
    p'* := <to-java> p*

to-java:
  Property(x, Type(t)) -> $[
    private [t] [x];

    public [t] get_[x] {
        return [x];
    }

    public void set_[x] ([t] [x]) {
        this.[x] = [x];
    }
]
```

Conclusion

- Co-evolution of language and IDE
- Pure and declarative syntax definition
- Language definition by transformation

- www.spoofax.org: papers, tour, download