The Spoofax Language Workbench

Lennart Kats    Eelco Visser

TU Delft
Delft University of Technology
Software Engineering

Problem → implement → Software

Software → validate ← Problem
High-level languages reduce problem/solution gap.
Domain-Specific Languages

DSLs further reduce gap between problem domain and implementation
Example DSL: mobl

```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) } 
  }
}
```

Renders a table, use row {} rows and cell {} for cells.
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 transformation
★ express constructs in core language

Code generation
★ translate core language models to implementation

parser generators

meta-programming libraries

meta-programming languages

template engines
ls

moblc -i tipcalculator.mobl

Compilation succeeded : [user/system] = [0.28s/0.19s]
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 )
    {
    }
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);
You are the IDE
to my DSL
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
Automatically derive **efficient, scalable, incremental compiler + usable IDE from high-level, declarative language definition**
Language Definition by Transformation
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")}

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

definition User {
  name = "John Doe"
  password = "password123"
  homepage = "www.example.com"
}

definition BlogPosting {
  poster : User
  body : String
}

definition BlogPosting {
  poster = "John Doe"
  body = "Hello, world!"
}

definition URL {
  location : String
}

definition BlogPosting {
  location = "www.example.com"
}
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
}
A **syntax** definition specifies a **transformation** from **text** to **trees**
Syntax as Transformation

```EntityLang.sdf
module EntityLang

imports Common

exports

context-free start-symbols

context-free syntax

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

```example.ent
module example

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

entity BlogPosting {
    poster : User
    body : String
}

entity URL {
    location : String
}
```
Syntax as Transformation
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
Syntax as Transformation

```
module EntityLang

  imports Common

  exports

  context-free start-symbols

  context-free syntax

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

```
Semantics = transformation
Error Marking is a Transformation
Error Marking is a Transformation
Error Marking is a Transformation
Error Marking is a Transformation
Error Marking is a Transformation
Outline View is a Transformation
Hyperlinking is a Transformation
Content Completion is a Transformation
Code Generation is a Transformation
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

```plaintext
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

```pseudo
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)
```

Origin tracking

```pseudo
module example

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

entity BlogPosting {
  poster : User
  body : String
}

entity URL {
  location : String
}
```

```pseudo
example.errors

["User!
  "Type User is not defined"]

transform

"BlogPosting"
  , [Property("poster", Type("User")
    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

```plaintext
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](http://www.spoofax.org)