Design of Embedded Systems (DES)

Jozef Hooman

http://www.cs.ru.nl/~hooman/DES/

Course 10
19 November 2014
Planning

• 19 Nov: application of DSL to small Lego Rover examples of DSL applications
• 26 Nov: discuss DSL proposals & code generation start Mars Rover development: requirements, risks, deployment, DSL
• 3 Dec: feedback on requirements & risks, determine deployment
• 10 & 17 Dec: Mars Rover development separate discussions with groups
• 7 Jan: final demos (?) - development
• 14 Jan: final demos - afternoon
Meta level, for developing the general infrastructure

Concrete syntax (.xtex)

Abstract syntax (.ecore)

Code generation (.xtend)

Textual input (.mydsl)

Model (.xmi)

Generated code (.*

Instance level, for developing a specific system
grammar & ecore model

StateMachine:
  'State Machine' name = ID
  'Init' init = State
  'States' states += State+
  'Events' events += Event+
  'Transitions'
    transitions += Transition*
  'Final' final += State;

State: name = ID;
Event: name = ID;

Transition:
  SelfTransition | StateChange;

SelfTransition:
  'FromTo' state = [State]
  'Event' event = [Event];

StateChange:
  'From' from = [State]
  'Trigger' trigger = [Event]
  'To' to = [State];

better: references to state for init and final

concrete syntax  abstract syntax
Problems

- Cannot change type twice within a rule
- An unassigned rule call is not allowed, when the 'current' was already created.
- The rule 'Transition' may be consumed without object instantiation. Add an action to ensure object creation, e.g. '{Transition}'.

The rule 'Transition' may be consumed without object instantiation. Add an action to ensure object creation, e.g. '{Transition}'.

```plaintext
Transition: {Transition}
trans += StateChange*
```
Expressions

- The rule 'Expression' is left recursive.
- This rule call is part of a left recursive call graph.

Left-factorization:

Addition: Multiplication ( ‘+’ Multiplication )*;

Multiplication: Number (‘*’ Number )* ;

Number : INT ;

ignore brackets
Expressions

All rules return same type:

Addition returns Expression:
   Multiplication ( ‘+’ Multiplication )*;

Multiplication returns Expression:
   Number (‘*’ Number )* ;

Number returns Expression: INT ;

Finally, in Xtext we need assignment to construct
Abstract Syntax Tree based on previously parsed part
Expressions

Expression : Addition;

Addition returns Expression:
    Multiplication \{Addition.left=current\} '+' right = Multiplication *;

Multiplication returns Expression:
    Number \{Multiplication.left=current\} '*' right = Number *;

Number : value = INT;

State Machine tryExpression
Init Start
States Start Mid Last Some
Events Button TimeOut Exit
Transitions
    From Start Trigger TimeOut ( 1*2+3 ) To Mid
    From Mid Trigger TimeOut ( 1*2*3+4*5+6+7*8*9 ) To Last
Final Last
init & final now as references to state
The following two are equivalent:

```python
enum ChangeKind :
    add
    | move_something
    | removeSomething
;

enum ChangeKind :
    ADD = 'add'
    | MOVE_SOMETHING = 'move_something'
    | REMOVE_SOMETHING = 'removeSomething'
;
```

- **abstract syntax**
- **concrete syntax**
Alternatives

Drive: action = 'drive';
Turn: action = 'turn' degrees = INT;
TurnRandom: action = 'turnRandom';
Backwards: action = 'backwards' seconds = INT;
Stop: action = 'stop';
ConditionalAction: 'when' condition += Condition+ 'do' actionSequence = ActionSequence;

could be replaced by:

Drive: {Drive} 'drive';
Turn: 'turn' degrees = INT;
TurnRandom: {TurnRandom} 'turnRandom';
Backwards: 'backwards' seconds = INT;
Stop: {Stop} 'stop';
ConditionalAction: 'when' condition += Condition+ 'do' actionSequence = ActionSequence;

or an enum for action types + optional parameters
Enum Type (1)

• Optional enum value becomes 0 if not specified

```javascript
enum StatusVal:
    ON | 0
    OFF | 1

Action:
    'action:' name = ID
    ('Left wheel:' left = Expression)?
    ('Right wheel:' right = Expression)?
    ('Timer:' timer = StatusVal)?
    ('Lamp:' lamp = StatusVal)?
```

• Above: timer and lamp become “0”, so “ON” if not specified …
Enum Type (2)

• Possible solution: add optional value as first enum value

```python
Action:
' action: name = ID
( 'Left wheel:' left = Expression)?
( 'Right wheel:' right = Expression)?
( 'Timer:' timer = StatusVal)?
( 'Lamp:' lamp = StatusVal)?
;
enum StatusVal:
 CURRENT |
 ON     |
 OFF
```

• timer and lamp become “0”
  so “CURRENT” if not specified
Enum Type (3)

- Other solution: check on `null` for optional fields by adding explicit type

```plaintext
Action:

' action:' name = ID
('Left wheel:' left = Expression)?
('Right wheel:' right = Expression)?
('Timer:' timer = Status)?
('Lamp:' lamp = Status)?

;

Status: status = StatusVal
;

enum StatusVal:
    ON |
    OFF
;
```
Xtend

- [http://www.eclipse.org/xtend/documentation.html](http://www.eclipse.org/xtend/documentation.html)

Xtend is a statically-typed programming language which translates to comprehensible Java source code

Extends Java with:

- **Extension methods** - enhance closed types with new functionality
- **Lambda Expressions** - concise syntax for anonymous function literals
- **Operator overloading** - make your libraries even more expressive
- **Powerful switch expressions** - type based switching with implicit casts
- **Multiple dispatch** - a.k.a. polymorphic method invocation
- **Template expressions** - with intelligent white space handling
- **No statements** - everything is an expression
- **Properties** - shorthands for accessing and defining getters and setter
Template Expressions

```python
def someHTML(List<Paragraph> paragraphs) ""
    <html>
    <body>
    «FOR p : paragraphs BEFORE '<div>' SEPARATOR '</div><div>' AFTER '</div>'»
    «IF p.headLine != null»
    <h1>«p.headline»</h1>
    «ENDIF»
    <p>
    «p.text»
    </p>
    «ENDFOR»
    </body>
    ""
</html>
```

The for expression optionally allows to specify what to prepend (BEFORE), put in-between (SEPARATOR), and what to put at the end (AFTER) of all iterations. BEFORE and AFTER are only executed if there is at least one iteration. (SEPARATOR) is only added between iterations. It is executed if there are at least two iterations.
Path in code generation

```java
var path = "../";

doGenerateConfig(path, resource, fsa);
doGenerateOil(path, resource, fsa);
doGenerateC(path, resource, fsa);

def doGenerateConfig(String path, Resource resource, IFileSystemAccess fsa){
    fsa.generateFile(path+"config.h",
        generateConfig(resource));
}
```
Xtend

• Access an element of an enumeration type: Color::BLACK
• Check whether an optional attribute is set: (attr != null)

• String operations: (the full Java API can be used)
  – Concatenation: "hello" + " " + "world"
  – Equality check: "<=".equals(name)

• List operations:

  «FOR element: someList»
  Element: «element»
  «ENDFOR»

  «FOR i : (0.. someList.size-1)»
  Element «i»: «someList.get(i)»
  «ENDFOR»
Xtend

• Shortcuts:
  – <CTRL>-<SHIFT>-O Organize imports (and automatically extend)
  – <CTRL>-<SPACE> Content assist, e.g., to create «» symbols

• Print to console: (for debugging, shown at meta-level console)
  println("The value of this variable is now: " + someVariable)

• Breakpoints
  – Double click in the grey border at the left side of the statement
  – A blue dot will appear in the grey border
  – Only effective when running the instance workspace in debug mode!
Assignment

Develop DSL & code generation for missions with first NXT robot (light, touch, and ultrasonic sensors)

- define concrete syntax (Bla.xtext)
- define code generation (BlaBla.xtend) for LeJOS
- experiment at instance level with some missions

Send me the main files:

From meta level:
- concrete syntax (Bla.xtext)
- code generation files (BlaBla.xtend + other files used)
- when used, validation file(s)

From runtime:
- (renamed) zip of project in runtime workspace; export using section 4.3.1 of manual

before Tuesday 25 November 18:00