Beyond Code: An Introduction to Model-Driven Software Development (CISC 836, Fall 2021)

Xtext Sample DSLs

The sample DSLs below assume the Eclipse IDE for Java and DSL Developers (version 2020-12 R, it will contain Xtext version 2.24 and all other software Xtext depends on). See the beginning of Assignment 4 for information on how to obtain and install it, and how use these sample DSLs. The origin of these DSLs is as follows:

- **Greetings, Arithmetics, State machines:** Part of the Xtext distribution
- **Person tasks:** Discussed in detail in the Xtext tutorial by Mooji and Hooman
- **Entities, State machines, SmallJava:** Used in Lorenzo Bettini's DSL book and available in the associated GitHub repo
- **Urml:** Developed by Keith Yip. For details on the language and its implementation, see his MSc thesis

1. **Greetings:**
   - 'Hello World' of Xtext
   - illustrates basic support for editing, validation, and generation
   - language infrastructure artifacts generated by Xtext from the DSL grammar to explore: ECore model (`mydsl/model/generated/MyDsl.ecore`), EMF code (`mydsl/src-gen/mydsl.myDsl`), Antlr parser (`mydsl/src-gen/mydsl.parser.antlr`), validation (`mydsl/src/mydsl.validation`), and generation (`mydsl/src/mydsl.generator`)
   - example:

greetings1.mydsl

```
Hello Al!
Hello Bo!
Hello Cy!
Hello Dee!
```

2. **Task assignment:**
   - language for specifying tasks and assigning people, priorities, and durations to them
   - illustrates generation of HTML and text artifacts from specifications, validation, support for arithmetic and boolean expressions in the grammar, and scoping(i.e., restricted visibility)
   - example:

spec3.tdsl

```
Planning DepartmentABC anonymous
Task: Report Strategy persons: Alice Carol priority: 5
Task: Pay 5000 euro persons: Bob priority: 2
Person: Alice Person: Fred
Task: Lunch Canteen persons: Fred priority: 8 duration: 1 hour
Task: Meeting "Demo" persons: Alice priority: 4 duration: 90 min
Person: Bob
Task: Meeting "Training" persons: Carol priority: 7 duration: 3 day
Task: Report Overview persons: Dave priority: 2 duration: 9 week
Task: Pay 3500 euro persons: Bob priority: 3
Person: Carol Person: Dave
```

3. **Entities:**
   - textual language for class modeling (i.e., 'class diagrams')
   - demonstrates Java code generation (`org.example.entities.generator`), validation (cyclic subclassing), and quick fixes (`org.example.entities.ui.quickfix`) to remove supertypes to break cycles and declare

missing entities
- example:

```
restaurant.entities
entity Restaurant {
    string name;
    boolean open;
    Menu menu;
    Table[20] table; }

entity Menu {
    MenuItem[30] item; }

entity MenuItem {..

entity Table { }

entity Employee {..

entity Server extends Employee {
    Table[] serves;
    Order[] takes;
    Bill issues; }

entity Bill {..

entity Customer {..

entity Order {..
```

4. **Arithmetics:**
   - calcuator for arithmetic expressions with support for user-defined functions
   - high-level description can be found here
   - illustrates the use of Xtext to implement an interactive interpreter (`arithmetics.interpreter`) using automatic edit of the specification to output evaluation results (`arithmetics.ui.autoedit`), content-assist to propose function arguments, and quick fixes to, e.g., normalize constant expressions
   - also showcases 'multiple dispatch' in Xtend, i.e., method resolution based on the runtime type of arguments (see method `internalEvaluate` in class `Calculator.xtend` in package `src/org.eclipse.xtext.example.arithmetics.interpreter`
   - example:

```
*test2.calc
module test
3+4;
// = 7
def inc(x):x+1;
inc(3);
// = 4
def f(x):2*(inc(x)+inc(x));
def n : 7;
f(n-2)+2;
// = 26
f(3)+
```

5. **State machines:**
   - simple textual state machine language originally due to Martin Fowler
   - high-level description can be found here
   - illustrates the use of Xtext for the generation of executable Java code from an artifact expressed in the DSL

- example:

SecretPanel1.statemachine

```
// sample state machine of secret door panel
events
    doorClosed  D1CL
    drawOpened  D2OP
    lightOn     L1ON
    doorOpened  D1OP
    panelClosed PNCL
end

resetEvents
    doorClosed
end

commands
    unlockPanel PNUL
    lockPanel   PNLK
    lockDoor    D1LK
    unlockDoor  D1UL
end

state idle
    actions {unlockDoor lockPanel}
    doorClosed => active
end

state active
    drawOpened  => waitingForLight
    lightOn     => waitingForDraw
end

state waitingForLight..

state waitingForDraw..

state unlockedPanel..
```

6. **Small Java**
   - sublanguage of Java
   - illustrates Java code generation, scoping, validation (type conformance, dead code, cyclic class hierarchy), testing (UI and generated code)
   - example:

SmallJavaExample.smalljava

```
package smalljava.examples;

import smalljava.examples2.*;

class SmallJavaExample {
    public SmallJavaExample m() {
        return new SmallJavaExample();
    }
    public String sayHello1() {
        return new AnotherSmallJavaExample().hello(true);
    }
    public String sayHello2() {
        return new AnotherSmallJavaExample().hello(false);
    }
}
```

7. **Urml:**
   - sublanguage of UML-RT as used in Assignment 4
   - illustrates interactive interpretation and the generation of executable Java code from artifacts expressed in the DSL

- example:

```
blinky.urml
/*
 * This example involves a yellow light that blinks.  The yellow light blinks by turning on
 * for 1 second and then off for 1 second.  Additionally, the yellow light proceeds a cycle
 * of alternately blinking for 5 seconds and stop blinking for 5 seconds.
 */
model Blinky {

     * The top of the model that consists of Blinky, the blinking yellow light,
    root capsule Top {

     * The blinking light blinks --- it turns on for 1 second and then off
    capsule BlinkingLight {
        external port connectToController : ControllerProtocol
        timerPort onAndOff_Timer
        logPort logger
        attribute int onAndOff_Period := 1000

        /*
         * This state machine describes the cycle alternating between
         * the light blinking and not blinking, which is coordinated by
         * the controller.
         */
        stateMachine {
    }

    /*
     * The controller makes sure that the yellow light blinks for 5 seconds and
     * then stops blinking for 5 seconds.
     */
    capsule Controller {
    protocol ControllerProtocol {
        incoming {
            start()
            stop()
        }
        outgoing {
        }
    }
}
```

Download

All artifacts are made available under the terms of the Eclipse Public Licence v1.0.

- DSL definitions: Projects defining the DSLs below and supporting tooling can be found here. Import into Eclipse using 'Existing Projects into Workspace'.
- Instances of the DSLs: Artifacts expressed in the DSLs (i.e., 'specifications', 'programs, or 'models') can be found here. Import into the 'runtime workspace' of the new Eclipse instance running the DSL plugins created by Xtext.

More information on DSLs with Xtext

- M. Barash. Introductory course on DSLs with Xtext and MPS
- M. Barash. Xtext grammar cheat sheet
- M. Barash. Xtend cheat sheet

Last modified: Tue Feb 09 2021 12:41:29