Internal Workflow for Executing Base Language Models with Extensions (DBX Files)

```
#import "for-loop"

module forLoopUse {

    void main() {
        list(int) is;
        int i;

        place 15 into is;
        place 42 into is;

        for (i in is with i > 0) {
            print i + "\n";
        }
    }
}
```

Fig. 1. DBX model as text

1. Create XMI from DBX Model

An XMI representation of a model is created automatically when the model is saved after a modification.

```
<statements xsi:type="dbl:ForLoop" concreteSyntax="for (i in is with i > 0) ...
  <it concreteSyntax="i" referencedElement="//@modules.0/@procedures.0/@statements.1"/>
  <set xsi:type="dbl:IdExpr" concreteSyntax="is"
       referencedElement="//@modules.0/@procedures.0/@statements.0"/>
  <condition xsi:type="dbl:Greater" concreteSyntax="i > 0">
    ...
  </condition>
```

Fig. 2. DBX model as XMI

2. Generate EMF Java Code

Generates EMF Java classes for all meta-classes, which were added by extensions.

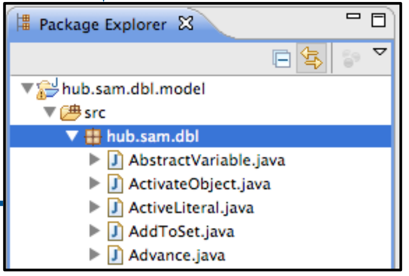

Fig. 3. Extended EMF model plugin for DBL

3. Create Working Copy from Original XMI

<input_model>_base.xmi

```
<statements xsi:type="dbl:ForLoop" concreteSyntax="for (i in is with i > 0) ...
  <it concreteSyntax="i" referencedElement="//@modules.0/@procedures.0/@statements.1"/>
  <set xsi:type="dbl:IdExpr" concreteSyntax="is"
       referencedElement="//@modules.0/@procedures.0/@statements.0"/>
  <condition xsi:type="dbl:Greater" concreteSyntax="i > 0">
    ...
  </condition>
```

Fig. 4. DBX model as XMI (working copy)

```
[template public extensionsToJava(model : Model)]
[for (ext : Extension | model.eAllContents(Extension)->asSet())]
    [for (extDef : ExtensionDefinition
            | model.imports.model.modules.extensionDefs)]
        [if (extDef.name = ext.eClass().name)]
            [genGlobalElements(extDef.eContainer().oclAsType(Module))/]
            [file (extDef.name + 'Semantics.java', false, 'UTF-8')]
                import hub.sam.dmx.AbstractExtensionSemantics;
                import hub.sam.dbl.*;

                public class [extDef.name/]Semantics extends Abstract...
```

Fig. 5. ExtensionsToJava.mtl

4. Invoke M2T Transformation for DBX Model (XMI)

First, find all extension instances in the DBX model. In the next step, make the semantics description of the corresponding extension definitions executable by translating them to Java. For each extension definition, a Java class named <ExtensionDefinitionName>Semantics is created.

```
extension ForLoop {
    Statement -> ForLoop;
    ForLoop -> "for" "(" it:$Variable "in"
        set:Expression "with" condition:Expression ")"
        "{" ManyStatements "}";
    ...
}
semantics {
        gen "for (" it ":" set ") {"; ...
```

Fig. 6. Excerpt Extension Definition

5. Execute *Semantics

For each extension instance, the corresponding *Semantics Java program is executed providing the extension instance as an argument. Each such invocation will create a so called modification, which is a DBL code replacement for the extension instance.

```
public class ForLoopSemantics extends AbstractExtensionSemantics {
    public static void main(String[] args) {
        (new ForLoopSemantics()).doGenerate(args);
    }

    public void doGenerate(EObject extensionInstance) {
        ForLoop self = (ForLoop) extensionInstance;
        gen("for (" + self.getIt().getConcreteSyntax() + ":"
            + self.getSet().getConcreteSyntax() + ") {");
        ...
```

Fig. 7. <ExtensionDefintion>Semantics.java

```
for (i: is) { if (i > 0) { print i + "\n"; } }
```

Fig. 8. Modificiations

6. Apply Modifications to the DBX Model

The applier works on the text of the DBX model. The result is a DBL model which consists of base language constructs only.

```
void main() {
    list(int) is;
    int i;

    place 15 into is;
    place 42 into is;

    for (i:is) {if (i > 0) {print i + "\n";} }
}
```

Fig. 9. Resulting DBL model as text

7. Parse DBL Model & Save as XMI

8. Invoke M2T Transformation for DBL Model (XMI)

```
[template public baseToJava(model : Model)]
[for (mod : Module | model.modules)]
    [if (mod.procedures->notEmpty()
            or mod.variables->notEmpty())]
        [genGlobalElements(mod)/]
    [/if]
    [genClasses(mod)/]

    [file ('JavaMain.java', false, 'UTF-8')]
        public class JavaMain {
```

Fig. 10. BaseToJava.mtl

9. Compile and Execute Target Language Model

```
***** DESMO-J version 2.2.0 *****
DefaultSimulation Experiment starts at ...
 ...please wait...
15
42
DefaultSimulation Experiment stopped at ...
Execution time: 0.107 seconds
```

Fig. 11. Execution Output