

Agile Language Engineering Tutorial – Creating a DSL for Telephone Control

with MMUnit (<http://mmunit.sf.net>),
EProvide (<http://eprovide.sf.net>), and
TEF (<http://tef.berlios.de>)

2008-09-16 at the University of Agder

by Daniel A. Sadilek <sadilek@informatik.hu-berlin.de>
Guido Wachsmuth <guwac@informatik.hu-berlin.de>

Agenda

- **Eclipse setup** (09:00 – 09:30)
- Domain-specific modelling (09:30 – 09:45)
- Agile language engineering (09:45 – 10:00)
- Metamodel testing (10:00 – 10:45)
 - Introduction
 - Demo
- **Test-first static metamodel** (10:45 – 12:15)
 - DTMF Control DSL introduction
 - Test specification
 - Static metamodel
- <lunch> (12:15 – 13:15)
- Operational semantics 1 (13:15 – 13:30)
- **Test-first dynamic metamodel** (13:30 – 14:00)
 - Adapt Test specification
 - Extend metamodel with runtime data structures
- Visual interpretation and debugging (14:00 – 14:15)
- **Create a textual editor** (14:15 – 14:30)
- Operational semantics 2 (14:30 – 14:45)
- **Describe semantics** (14:45 – 16:15)
 - Build initialisation transformation
 - Build transition transformation
 - Make it extensible with new external functions
 - Hook it up to a test GUI

Set up Eclipse

- Install and start eclipse-rcp-ganymede
- Add update sites:
 - http://mmunit.sf.net/update
 - http://eprovide.sf.net/updatesite
 - http://tef.berlios.de/updatesite
- Select all features from all three update sites;
from the "Ganymede" update site,
select "Models and Model Development"->"Ecore Tools SDL"
and "Models and Model Development"->"EMF SDK"
and "Models and Model Development"->"Graphical Modelling Framework SDK"
- Click "Install..." to install your selection

Setup DSL plugin

- Create new plugin project: hub.metrik.lang.dtmfcontrol
- Create a new source folder test
- Add necessary dependencies to the project
 - Open META-INF/MANIFEST.MF
 - On the dependencies tab, add:
 - de.hu_berlin.sam.mmunit (2.0.2)
 - hub.metrik.lang.eprovide (2.1.0)
 - hub.metrik.lang.eprovide.java (2.1.0)
 - hub.sam.tef (1.0.5)
 - org.eclipse.emf.transaction (1.2.0)
 - org.eclipse.emf.workspace (1.2.0)
 - org.eclipse.ui (3.4.0)
 - org.eclipse.ui.ide (3.4.0)
 - org.junit4 (4.3.1)
 - On the MANIFEST.MF tab, modify the following line to add the singleton-directive:
Bundle-SymbolicName: hub.metrik.lang.dtmfcontrol;singleton:=true

Create the DSL metamodel

- Create Ecore model in model/dtmfcontrol.ecore
- Set the packages properties:
 - Name: dtmfcontrol
 - Ns Prefix: dtmfcontrol
 - Ns URI: http://informatik.hu-berlin.de/dtmfcontrol
- Create new MMUnit Model in model/tests/example.mmunit
- Initialize the model's diagram file in model/tests/example.mmunit_diagram (look in the model file's context menu)
- Model an object hierarchy describing the following example program (example.txt):

```
main {
  say "main menu";
  listen {
    1: "control lights" -> lights;
  }
}

lights {
  say "your lights are";
  say lights_status();
  listen {
    1: "switch your lights" -> switch_lights;
    0: "return to main menu" -> main;
  }
}

switch_lights {
```

```

lights_switch();
say "your lights are now";
say lights_status();
goto main;
}

```

- Generate a JUnit-Test for the example.mmunit test
 - In the example.mmunit's context menu, select MMUnit -> Add As Test Model
 - In the dtmfcontrol.ecore's context menu, select MMUnit -> Set As Metamodel
 - In the source folder tests' context menu, select MMUnit -> Create JUnit Tests Here
 - Name the file ExampleTest
- Run the generated test as a JUnit Test. The test should fail and tell you the reason.
- Modify the metamodel and rerun the test until it succeeds
 - To edit the metamodel graphically, select "Initialize Ecore Diagram File..." from its context menu
- Add existence specifications to example.mmunit to ensure that
 - the a listen block must have at least one option
 - that a goto statement must point to exactly one label
- Rerun the test and modify the metamodel until the test succeeds

- Uncomment the commented line in the generated JUnit test and change it as follows:

```

test.saveInstantiatedModel(test.getInstantiatedModels().get(0), "model/tests/example.xmi");

```

This saves an instance of the metamodel as described in example.mmunit under model/tests/example.xmi.

Model runtime data structures

- Think about which data structures are needed at runtime to execute the example program and model in example.mmunit how they should look like. You probably need
 - some form of instruction pointer
 - a list containing the keys that the user presses during the program's run (for this example, we store them in the model, in a real project, the inputs would probably be read from a GUI showing the system's output and keys for generating input)
 - an initially empty list that gets filled during execution with the output messages the system generates
 - a boolean value to store the current state of the lights
- Rerun the test and modify the metamodel until the test succeeds

Create a textual editor

- Generate Java code for the metamodel
 - In the dtmfcontrol.ecore's context menu, select New->Other->Eclipse Modeling Framework->EMF Model and accept all defaults
 - In the generated dtmfcontrol.genmodel, edit the package's properties: set "Base Package" to "hub.metrik.lang"
 - Select "Generate All" from the package's context menu
 - Open META-INF/MANIFEST.MF, on the Runtime tab, click "Add..." and select all packages
- Open META-INF/MANIFEST.MF, on the plugin.xml tab, add the following extension after the <plugin> tag (extension.txt):

```

<extension
    point="org.eclipse.emf.transaction.editingDomains">
    <editingDomain
        factory="org.eclipse.emf.workspace.WorkspaceEditingDomainFactory"
        id="hub.metrik.lang.dtmfcontrol.editingDomain">
    </editingDomain>
</extension>

```

```

<extension
    point="org.eclipse.ui.editors">
    <editor
        class="hub.metrik.lang.dtmfcontrol.editor.DTMFEditor"
        default="false"
        extensions="dtmfcontrol"
        id="hub.metrik.lang.dtmfcontrol.editor"
        name="DTMF Control Editor">
    </editor>
</extension>

```

- Create the class `hub.metrik.lang.dtmfcontrol.editor.DTMFEditor` (`editor.txt`):

```

package hub.metrik.lang.dtmfcontrol.editor;

import hub.metrik.lang.dtmfcontrol.DtmfcontrolPackage;
import hub.metrik.lang.dtmfcontrol.util.DtmfcontrolAdapterFactory;
import hub.metrik.lang.eprovide.StepNotification;
import hub.sam.tef.editor.model.ModelDocumentProvider;
import hub.sam.tef.editor.model.TransactionModelEditor;
import hub.sam.tef.layout.AbstractLayoutManager;
import hub.sam.tef.layout.BlockLayout;

import java.util.ArrayList;
import java.util.List;

import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.Platform;
import org.eclipse.emf.common.notify.Adapter;
import org.eclipse.emf.common.notifyAdapterFactory;
import org.eclipse.emf.common.notify.Notification;
import org.eclipse.emf.common.notify.impl.AdapterImpl;
import org.eclipse.emf.ecore.EPackage;
import org.eclipse.jface.text.IDocument;
import org.eclipse.jface.text.TextAttribute;
import org.eclipse.jface.text.rules.IRule;
import org.eclipse.jface.text.rules.SingleLineRule;
import org.eclipse.jface.text.rules.Token;
import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Color;
import org.eclipse.swt.graphics.RGB;
import org.eclipse.swt.widgets.Display;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.PlatformUI;
import org.osgi.framework.Bundle;

/**
 * An editor for the dtmf control DSL.
 */
public class DTMFEditor extends TransactionModelEditor {
    private static final String PLUGIN_ID = "hub.metrik.lang.dtmfcontrol";
    private static final String EDITING_DOMAIN_ID =
"hub.metrik.lang.dtmfcontrol.editingDomain";
    private Adapter stepListener;

    @Override
    public AbstractLayoutManager createLayout() {
        return new BlockLayout();
    }

    @Override
    protected AdapterFactory[] createItemProviderAdapterFactory() {
        return new AdapterFactory[] { new DtmfcontrolAdapterFactory() };
    }
}

/**

```

```

* Override super method to use a {@link ModelDocumentProvider} that adds
* (and also removes) an adapter, which updates the editor text when an
* EProvide step is performed.
*
* @see hub.sam.tef.editor.model.ModelEditor#initialiseDocumentProvider()
*/
@Override
protected void initialiseDocumentProvider() {
    setDocumentProvider(new ModelDocumentProvider(this) {
        @Override
        protected boolean setDocumentContent(final IDocument document,
            IEditorInput editorInput) throws CoreException {
            boolean result = super
                .setDocumentContent(document, editorInput);
            stepListener = new AdapterImpl() {
                @Override
                public void notifyChanged(Notification notification) {
                    if (notification instanceof StepNotification) {
                        PlatformUI.getWorkbench().getDisplay().syncExec(
                            new Runnable() {
                                public void run() {
                                    try {
                                        updateDocumentContents(
                                            document,
                                            getCurrentModel());
                                    } catch (CoreException ce) {
                                        throw new RuntimeException(
                                            "Could not update editor text after EProvide step.",
                                            ce);
                                    }
                                }
                            }
                        );
                    }
                }
            };

            waitForReconciliation();
        }
    });
    getCurrentModel().eAdapters().add(stepListener);

    return result;
}

@Override
protected void modelDispose() {
    super.modelDispose();
    getCurrentModel().eAdapters().remove(stepListener);
}
});
}

@Override
protected EPackage[] createMetaModelPackages() {
    return new EPackage[] { DtmfcontrolPackage.eINSTANCE };
}

@Override
protected Bundle getPluginBundle() {
    return Platform.getBundle(PLUGIN_ID);
}

@Override
protected String getSyntaxPath() {
    return "syntax/dtmfcontrol.etslt";
}

@Override

```

```

public List<IRule> getAdditionalPresentationRules() {
    List<IRule> result = new ArrayList<IRule>();
    result.add(new SingleLineRule("#", null, new Token(new TextAttribute(
        new Color(Display.getCurrent(), new RGB(255, 46, 248)), null,
        SWT.NORMAL))));
    return result;
}

protected String getEditingDomainId() {
    return EDITING_DOMAIN_ID;
}
}

```

- Create the syntax description for the editor in syntax/dtmfcontrol.etslt
- Describe a grammar for your language including the runtime data structures. The textual representation of runtime data structures should be preceded by #. Thus, they will be colored pink. The example program from above including runtime states could, for example, look like this (notice the comment behind the second listen command) (exampleRuntime.txt):

```

# key presses: 1 1 0
# current index: 1
# light status: on
# output: "main menu"
# output: "press 1 to control lights"
# output: "your lights are"
# output: "on"
# output: "press 1 to switch your lights"
# output: "press 0 to return to main menu"

```

```

main {
    say "main menu";
    listen {
        1: "control lights" -> lights;
    }
}

lights {
    say "your lights are";
    say lights_status();
    listen { # <-- next
        1: "switch your lights" -> switch_lights;
        0: "return to main menu" -> main;
    }
}

switch_lights {
    lights_switch();
    say "your lights are now";
    say lights_status();
    goto main;
}

```

- Your grammar could look like this:

```

syntax(app) "model/dtmfcontrol.ecore" {
    app:element(Application) -> (runtimeStatus ws(statement) ws(statement))?
        (block:composite(blocks))*;

    runtimeStatus -> keyPresses currentIndex lightStatus outputs;
    keyPresses -> "#" ws(space) "key" ws(space) "presses" ":" ws(space)
        (keyPress:composite(keyPresses) ("," ws(space)
keyPress:composite(keyPresses))*? ws(statement);
    keyPress:element(KeyPress) -> INTEGER:composite(key);
    currentIndex -> "#" ws(space) "current" ws(space) "index" ws(empty) ":"
ws(space)

```

```

        INTEGER:composite(currentKeyPress) ws(statement);
    lightStatus -> "#" ws(space) "light" ws(space) "status" ws(empty) ":"
ws(space)
        "on" ws(statement);
    outputs -> (output:composite(outputs))*;
    output:element(Output) -> "#" ws(space) "output" ws(empty) ":" ws(space)
        STRINGDEF:composite(text) ws(statement);

    block:element(Block) -> IDENTIFIER:composite(name) ws(empty) "{"
        ws(blockstart) (command:composite(commands))*
ws(blockend)
        "}" ws(statement) ws(statement);
    blockRef:element(Block) -> IDENTIFIER:composite(name);

    command -> ws(indent) sayText;
    command -> ws(indent) sayExternal;
    command -> ws(indent) external;
    command -> ws(indent) goto;
    command -> ws(indent) listen;

    sayText:element(SayText) -> "say" ws(space) STRINGDEF:composite(text)
commandEnd;
    external:element(External) -> IDENTIFIER:composite(function) ws(empty) "("
ws(empty) ")" commandEnd;
    sayExternal:element(SayExternal) -> "say" ws(space)
IDENTIFIER:composite(function) ws(empty) "(" ws(empty) ")" commandEnd;
    goto:element(Goto) -> "goto" ws(space) blockRef:reference(target)
commandEnd;
    listen:element(Listen) -> "listen" ws(space) "{" (activity)?
        ws(blockstart) (listenOption:composite(options))*
ws(blockend)
        ws(indent) "}" ws(statement);

    listenOption:element(ListenOption) ->
        ws(indent) INTEGER:composite(key) ws(empty) ":" ws(space)
        sayTextInOption:composite(say) ws(space) "->" ws(space)
blockRef:reference(target) ws(empty) ";" ws(statement);
    sayTextInOption:element(SayText) -> STRINGDEF:composite(text);

    commandEnd -> ws(empty) ";" (activity)? ws(statement);
    activity -> ws(space) "#" ws(space) "<--" ws(space)
activityComment:composite(next);
    activityComment:constant("true":EBoolean) -> "next command";
}

```

- This grammar uses a derived attribute called "nextCommand" in the class for Applications that is implemented as follows (nextCommand.txt):

```

/**
 * @generated NOT
 */
public Command basicGetNextCommand() {
    for (Block block : getBlocks())
        for (Command command : block.getCommands())
            if (command.isNext()) return command;
    return null;
}

/**
 * @generated NOT
 */
public void setNextCommand(Command newNextCommand) {
    for (Block block : getBlocks())
        for (Command command : block.getCommands())
            command.setNext(newNextCommand == command);
}

```

- Execute the editor by selecting Run As->Eclipse Application from the plugin's context menu
- In the new Eclipse instance, import the plugin project
- Rename model/tests/example.xmi to model model/tests/example.dtmfcontrol and open it.
- The TEF editor you just created should open and pretty print the example model.

Create a simulator/debugger

- Open MANIFEST.MF, on tab extension add an extension for hub.metrik.lang.eprovide.java.semantics
 - unique identifier: hub.metrik.lang.dtmfcontrol.semantics
 - name: DTMF Control Semantics
 - file extension: dtmfcontrol
 - editing domain: hub.metrik.lang.dtmfcontrol.editingDomain
 - create a semantics provider class
hub.metrik.lang.dtmfcontrol.semantics.Simulator (follow the link in the extension tab)
- Implement initialisation
 - implement reset(Resource resource)
 - navigate to the model contained by the resource
 - modify the model to get an initial state (use the generated API)
 - think about initialisation of instruction pointers, pressed keys, outputs, and light status
 - have multiple instances running different models in mind
- Implement transition step
 - implement step(Resource resource)
 - realise different cases for transition steps
 - think about typical error situations (hint: unsupported key, end of block, missing external functions)
 - hint: to create new model elements, obtain the static instance of DtmfcontrolFactory
- Test your implementation
 - debug as Eclipse Application
 - in the new Eclipse instance, open a DTMF Control model
 - run it as executable model
 - observe the behaviour and correct your implementation (hint: set breakpoints in your implementation)
 - run the model without animation (hint: runtime configuration)
 - debug the model, you can run the model stepwise (even backwards)
- Refactor your implementation
 - avoid duplicated code
 - take care of structure and readability
 - make it easy to provide external functions for different application domains