PINEAPPLE DATAFLOWNET MODELER
QUICK GUIDE

Created by Ákos Hajdu, József Makai and Márton Búr as a homework for the "Model Driven Software Development" course at the Budapest University of Technology and Economics (http://www.inf.mit.bme.hu/en)

Technologies used

- Eclipse Modeling Framework 2.0.2 for creating the metamodel (Has to be installed in Eclipse)
- EMF IncQuery 0.8.0 (http://viatra.inf.mit.bme.hu/incquery) for validation (Has to be installed in Eclipse)
- Xtext 2.5.3 for creating textual language (Has to be installed in Eclipse)
- Xtend 2.5.3 for generating GraphML and Java code (Has to be installed in Eclipse)
- yED 3.12.2 for visualization (http://www.yworks.com/en/products_yed_about.html)
- Akka 2.2.2 (http://akka.io/) and Paho 0.4.0 (http://www.eclipse.org/paho/) for remote communication (Included as dependecies)

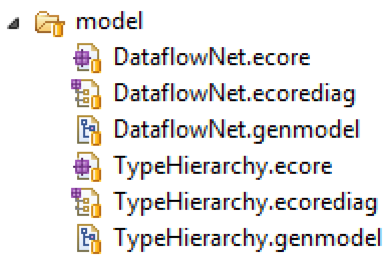
How to use it

Load metamodel and sample model

Clone the project and load the following projects into the workspace:

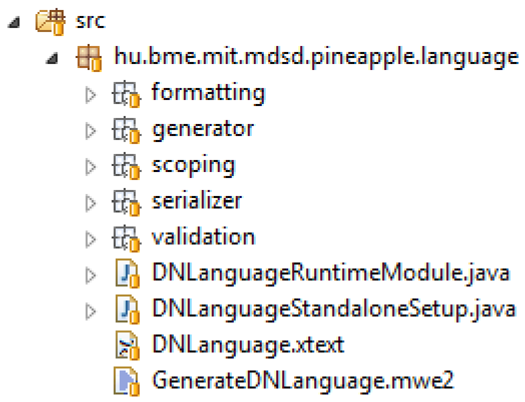
- hu.bme.mit.mdsd.pineapple
- hu.bme.mit.mdsd.pineapple.dataflownet.common
- hu.bme.mit.mdsd.pineapple.dataflownet.generator
- hu.bme.mit.mdsd.pineapple.dataflownet.generator.incquery
- hu.bme.mit.mdsd.pineapple.dataflownet.graphml
- hu.bme.mit.mdsd.pineapple.dataflownet.incquery
- hu.bme.mit.mdsd.pineapple.dataflownet.ui.generator
- hu.bme.mit.mdsd.pineapple.language
- hu.bme.mit.mdsd.pineapple.ui.buttons

The metamodel can be found in the hu.bme.mit.mdsd.pineapple project.



Generate the metamodels using the .genmodel files. Generate TypeHierarchy.genmodel first, and then DataflowNet.genmodel.

The textual language (Xtext) can be found in the hu.bme.mit.mdsd.pineapple.language project.

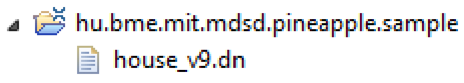

src
hu.bme.mit.mdsd.pineapple.language
formatting
generator
scoping
serializer
validation
DNLanguageRuntimeModule.java
DNLanguageStandaloneSetup.java
DNLanguage.xtext
GenerateDNLanguage.mwe2

Right click on the file GenerateDNLanguage.mwe2 and select Run as → MWE2 Workflow.

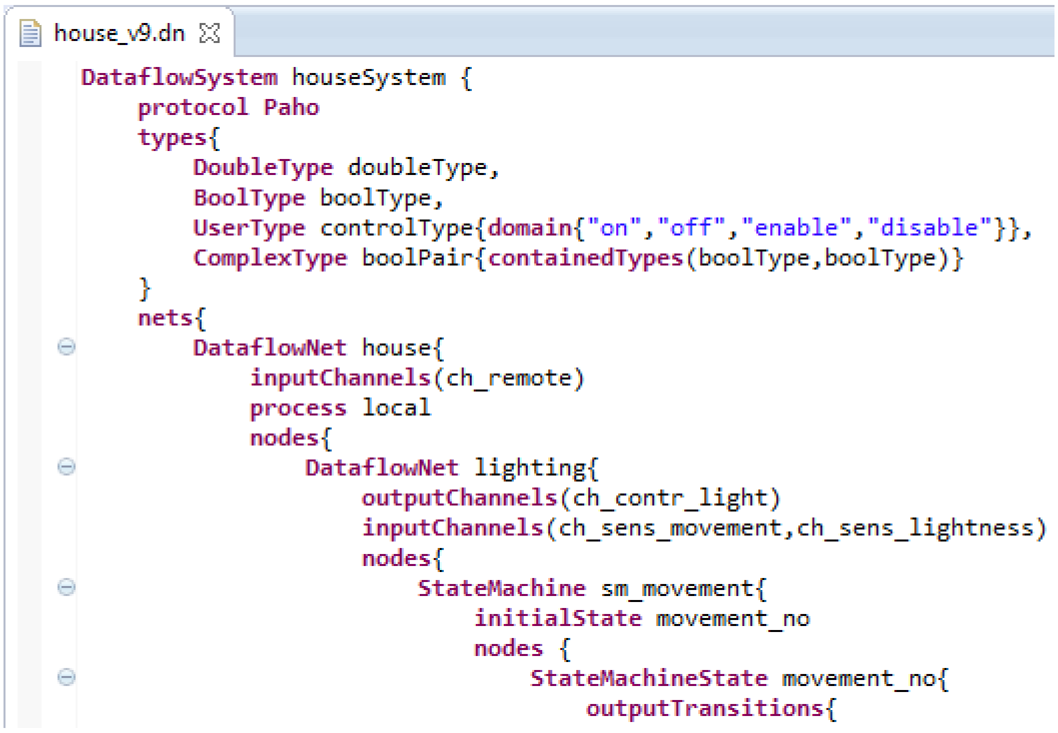
Start a runtime Eclipse application and load the following projects:

- hu.bme.mit.mdsd.pineapple.sample
- hu.bme.mit.mdsd.pineapple.dataflownet.bundle

The sample instance model can be found in the file house_v9.dn in the project hu.bme.mit.mdsd.pineapple.sample.



```
hu.bme.mit.mdsd.pineapple.sample
    house_v9.dn
```

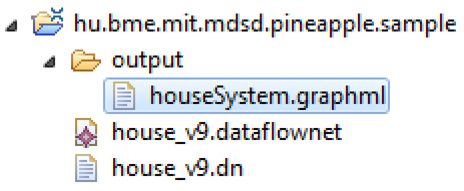


```
house_v9.dn
DataflowSystem houseSystem {
    protocol Paho
    types{
        DoubleType doubleType,
        BoolType boolType,
        UserType controlType{domain{"on","off","enable","disable"}},
        ComplexType boolPair{containedTypes(boolType,boolType)}
    }
    nets{
        DataflowNet house{
            inputChannels(ch_remote)
            process local
            nodes{
                DataflowNet lighting{
                    outputChannels(ch_contr_light)
                    inputChannels(ch_sens_movement,ch_sens_lightness)
                    nodes{
                        StateMachine sm_movement{
                            initialState movement_no
                            nodes {
                                StateMachineState movement_no{
                                    outputTransitions{
```

Right click on the file house_v9.dn and select “Convert textual dataflownet to XMI” to get the file house_v9.dataflownet. This file can also be edited using the tree editor.

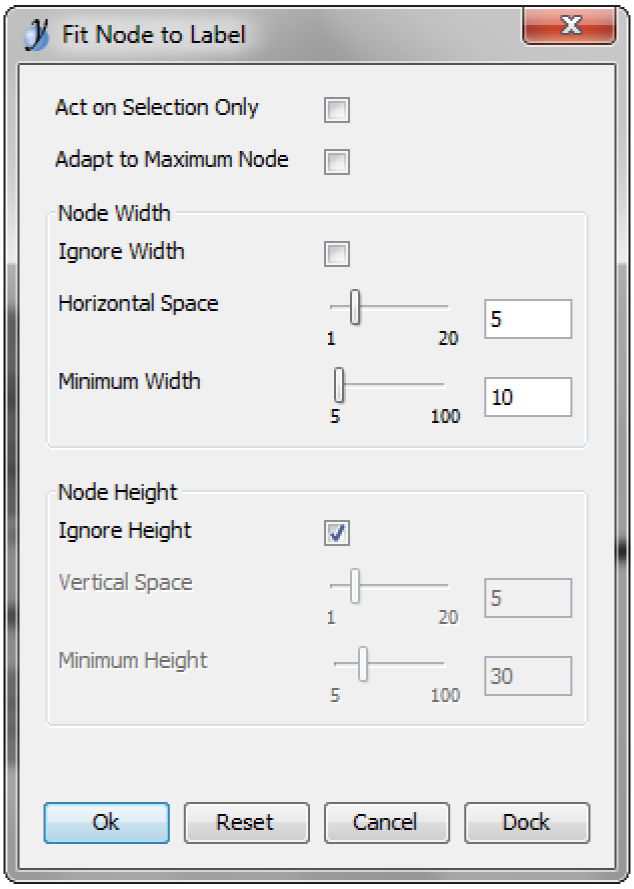
Generate GraphML code

Right click on the house_v9.dataflownet file and select “Generate GraphML Representation”. The .graphml file will be generated in the sample project under a folder named output.



```
hu.bme.mit.mdsd.pineapple.sample
    output
        houseSystem.graphml
    house_v9.dataflownet
    house_v9.dn
```

Open the .graphml file in yED and select Tools → Fit Node to Label. Use the following settings and click ok.



Fit Node to Label

Act on Selection Only ☐

Adapt to Maximum Node ☐

Node Width

Ignore Width ☐

Horizontal Space 1 20 5

Minimum Width 5 100 10

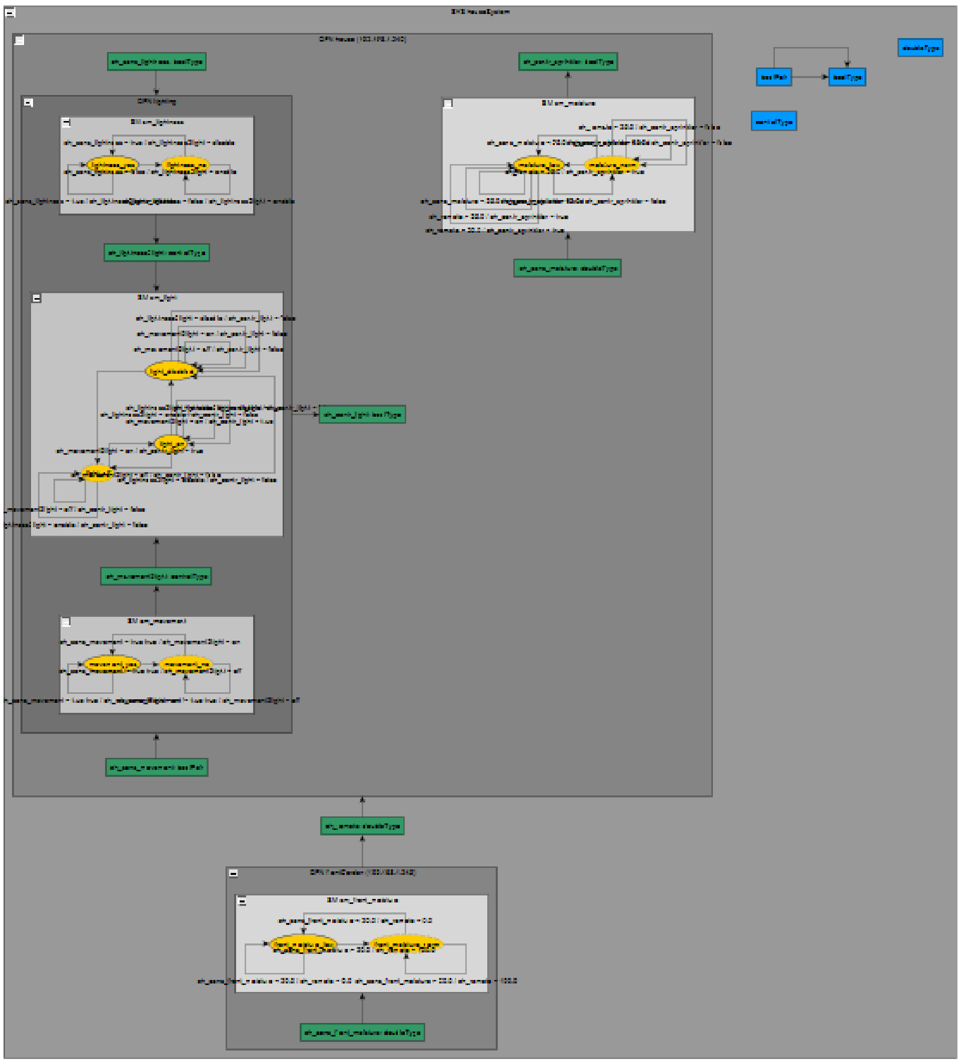
Node Height

Ignore Height ☑

Vertical Space 1 20 5

Minimum Height 5 100 30

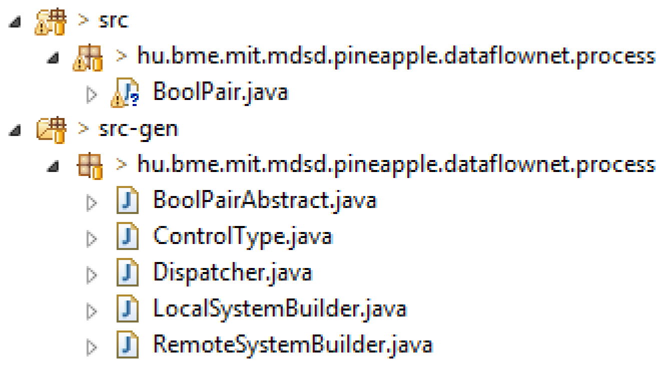
Use some layout algorithm, for example Layout → Orthogonal → Classic. The result can be seen below.



Generate Java code

Right click on the house_v9.dataflownet file and select “Generate Java Simulation Code” then select “Generate Java UI Code”.

The following files are generated in the bundle project:



```
> src
  > hu.bme.mit.mdsd.pineapple.dataflownet.process
    BoolPair.java
> src-gen
  > hu.bme.mit.mdsd.pineapple.dataflownet.process
    BoolPairAbstract.java
    ControlType.java
    Dispatcher.java
    LocalSystemBuilder.java
    RemoteSystemBuilder.java
```

Implement the unimplemented methods (compareTo and toString) of the custom complex types (BoolPair) under the src folder. For example use the following implementation:

```
@Override
public int compareTo(BoolPair boolPair) {
      if(value1 == boolPair.value1 && value2 == boolPair.value2) return 0;
      return 1;
}

@Override
public String toString(){
      return value1 + "," + value2;
}
```

A new project is generated for the UI, namely hu.bme.mit.mdsd.pineapple.dataflownet.process.ui.

Try the generated Java code

Close the runtime Eclipse application and load the following projects into the host Eclipse:

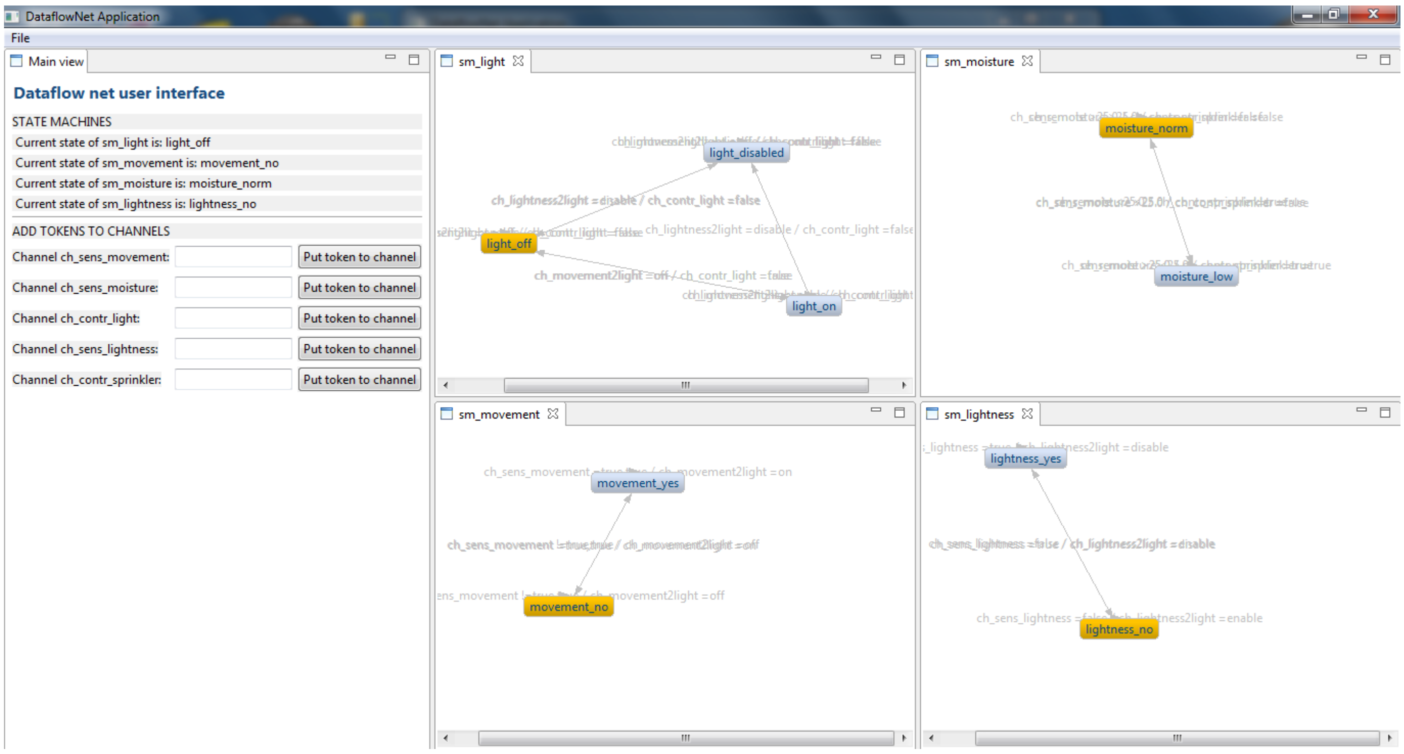
- hu.bme.mit.mdsd.pineapple.dataflownet.process.ui
- hu.bme.mit.mdsd.pineapple.dataflownet.bundle
- hu.bme.mit.mdsd.pineapple.dataflownet.bundle.dependencies
- hu.bme.mit.mdsd.pineapple.dataflownet.bundle.dependency.paho
- hu.bme.mit.mdsd.pineapple.dataflownet.ui.product

Export the RCP using the DataflownNetApplication.product in the project hu.bme.mit.mdsd.pineapple.dataflownet.ui.product. You may have to add the required plug-ins on the Dependencies tab of DataflownNetApplication.product.

Run the exported application DataflowNetUI.exe in two instances with the following command line arguments:

- The first one with “local –clearPersistedState”
- The second one with “remote –clearPersistedState”

The first parameter (“local” and “remote” in the example) is the name of the process defined in the metamodel. After arranging the tabs, you get the following screens:



DataflowNet Application

File

Main view

Dataflow net user interface

STATE MACHINES

Current state of sm_light is: light_off

Current state of sm_movement is: movement_no

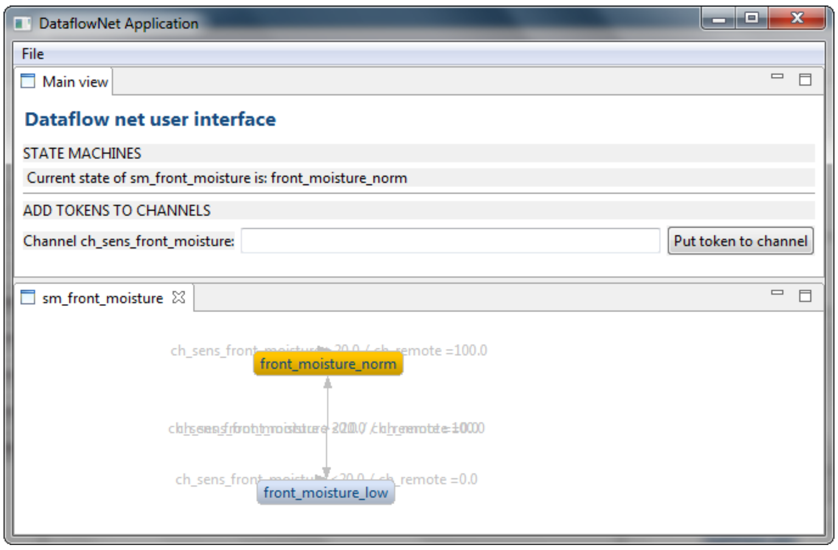
Current state of sm_moisture is: moisture_norm

Current state of sm_lightness is: lightness_no

ADD TOKENS TO CHANNELS

Channel ch_sens_movement:		Put token to channel
Channel ch_sens_moisture:		Put token to channel
Channel ch_contr_light:		Put token to channel
Channel ch_sens_lightness:		Put token to channel
Channel ch_contr_sprinkler:		Put token to channel

sm_light
light_disabled
ch_lightness2light = disable / ch_contr_light = false
light_off
ch_lightness2light = disable / ch_contr_light = false
ch_movement2light = off / ch_contr_light = false
light_on
sm_moisture
moisture_norm
moisture_low
sm_movement
movement_yes
ch_sens_movement = true true / ch_movement2light = on
movement_no
ch_sens_movement != true true / ch_movement2light = off
sm_lightness
lightness_yes
ch_sens_lightness = false / ch_lightness2light = disable
ch_sens_lightness = true / ch_lightness2light = enable
lightness_no



DataflowNet Application

File

Main view

Dataflow net user interface

STATE MACHINES

Current state of sm_front_moisture is: front_moisture_norm

ADD TOKENS TO CHANNELS

Channel ch_sens_front_moisture:		Put token to channel

sm_front_moisture
ch_sens_front_moisture >= 20.0 / ch_remote = 100.0
front_moisture_norm
ch_sens_front_moisture < 20.0 / ch_remote = 0.0
front_moisture_low

You can now put tokens on the top-level channels by entering the value of the token as a string in the textboxes. You can try for example putting “10” on ch_sens_front_moisture and putting “true true” on ch_sens_movement.