M Ű E G Y E T E M 1 7 8 2

Budapest University of Technology and Economics (BME)
Faculty of Electrical Engineering and Informatics (VIK)
Department of Measurement and Information Systems

Tamás Borbás
Péter Lunk
Dávid Zilahi

DFN MODELER USER GUIDE

BUDAPEST, 2014

Contents

1 Environment

Java Development Kit: 1.7.0

Java JDK 1.7

Eclipse: Kepler (4.3) Modeling Tools

Eclipse 4.3 Modeling Tools

Plugins for Eclipse:

- XText 2.5.3, XTend 2.5.3
  - Update Site:
  - http://download.eclipse.org/modeling/tmf/xtext/updates/composite/releases/
- EMF-IncQuery SDK: 0.8.0
  - Update Site: http://download.eclipse.org/incquery/updates/integration
- E(fx)clipse: 0.9.0 (For Example Greenhouse tester App)
  - Update Site:
  - http://download.eclipse.org/efxclipse/updates-released/0.9.0/site
- m2e: 1.4.0 (Maven Eclipse plug-in)
  - Update Site: http://download.eclipse.org/technology/m2e/releases/

MQTT broker: http://mosquitto.org/

Paho library: http://www.eclipse.org/paho/

ActiveMQ: http://activemq.apache.org/

2 How To

2.1 Run configuration

Main page

- Run a product: org.eclipse.platform.ide

Arguments page

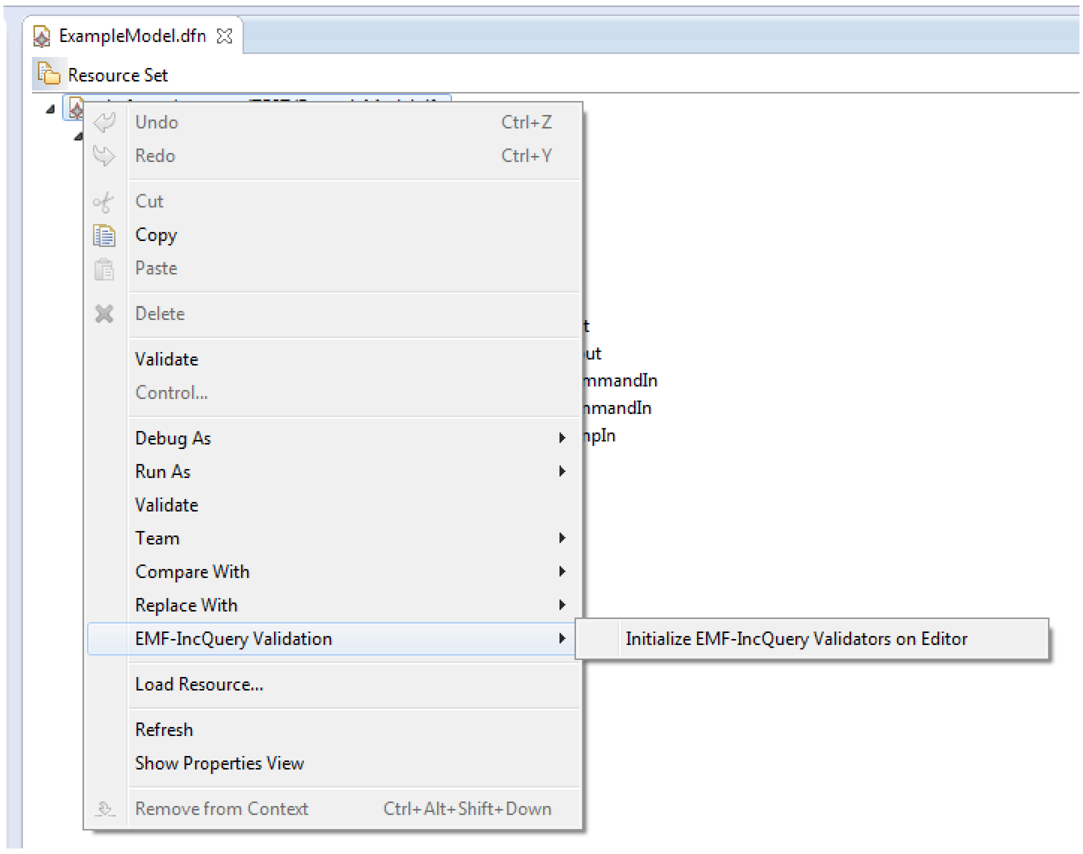
- Program arguments: -os ${target.os} -ws ${target.ws} -arch ${target.arch} -nl ${target.nl} –consoleLog
- VM arguments: -Dosgi.requiredJavaVersion=1.6 -XX:MaxPermSize=512m –Xms128m -Xmx512m

2.2 Plugin usage

Import projects to Eclipse and start the previously described run configuration.

2.2.1 Modeller

First of all you need a simple project. In this project you can create new DFN model. If you open the dfn with the DFN Model Editor you get a tree editor. The root element should be a Data Flow Network but in this you have a lot of options what you want to create. Before you start working we suggest initializing our model validator system (1.).



1. Figure – Initializing the model validator

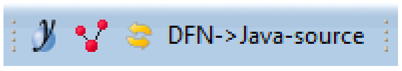
You can separate your models generated code in different projects. You only have to create an Application object in the model and assign the desired nodes to it. It is also possible to specify the communication channel between the Applications (JMS, MQTT or local), via a property of a Data Flow Network object. Data Flow Networks and Nodes communicate over Channels and these channels are represented by node and DFN ports. Each channel can be assigned to a source Port and to a target Port. You also have to specify the Token which to send over the Channel. Apart from containing Nodes, it is also possible to incorporate other Data Flow Networks in a Data Flow Network object.

Nodes contain States and Transactions between these given State objects. In addition to this, you can define conditions based on which the Transactions will fire. The firing rules are represented by Equations or Inequalities. The states and transactions specify how the system will work.

There is an example project with a .dfn file in our repository's example folder. You can import this project to an Eclipse where the DFN modeler plug-in structure has been installed (or a run-time Eclipse with the specified settings).

2.2.2 Generators

The toolbar, which is responsible for the main functions of the tool, consists of four buttons (2.) These are responsible for code generation and model visualization. The first and second button generates graphml files in the project's graphs folder (if the folder does not exist the plug-in will create it). The difference between generated files is that the first button generates a yEd specific graph with labels, colors and content based size. The second button however, generates a standard graphml file, which can be used with any other tool as well. The third, refresh button refreshes the Zest-based run-time DFN visualizer view. To access this feature, please open the DFN Visualizer/DFN Graph Visualizer Eclipse view first.

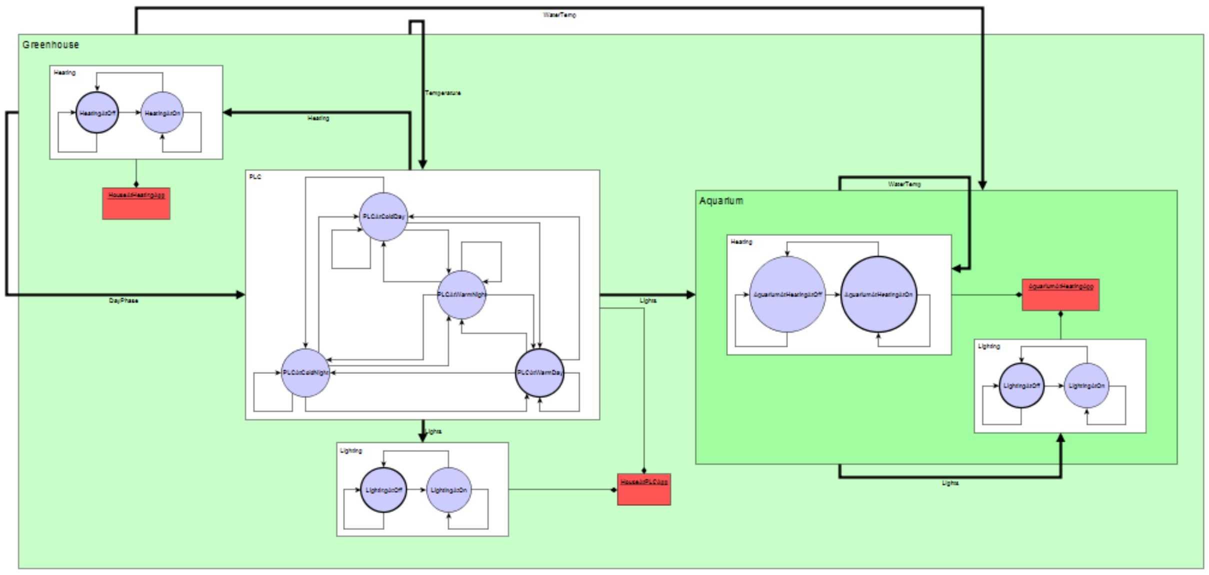


2. Figure - Toolbar

The last button will generate Java source code, which can be used to create distributed DFN systems. It also generates a java library that enables easier input access. It generates multiple projects. The one that has the name of the root DFN contains the DFN logic. The one with the "controller" postfix is the access library, and all the other projects are the distributed components of the DFN called "App".

2.3 Graph Generator usage

Both graphs which are generated by the plug-in (yEd specific and standard) can be opened with yEd. A layout is needed if you want to see the graphs because the generated XML file contains no information on layouting. We suggest the Classic Orthogonal layout if you use yEd because the others can hide some edges. After using the generator on the example project the following output is created.



3. figure – Layouted graph

2.4 Generated code usage

2.4.1 Project setup

After generation, you can see that some of the projects have errors. This happens because of broken dependencies. We use maven, and the generated maven pom.xml files are contained by each project. Before executing the examples refresh maven (alt+F5), so that it downloads the dependencies.

Also make sure that the required Message queue provider is running. ActiveMQ 5.9.1 is recommended as it supports both MQTT and JMS as well.

2.4.2 Running the Apps

Every project with “App” postfix has a main function, and is executable. You have specified the message broker in the project, and if it is available, you can run the apps.

At this point, you have a running distributed multilevel DFN, but you can’t give it input, so you can’t test it. The testing can be done via the unified controller library.

2.4.3 Using the Controller library

You can use this library as a simple remote control to your DFN. In this java library, there is a class for every DFN in your multilevel DFN graph. You can set the inputs with the functions start with “setInputOn”. You can also query their states with simple getter functions.

```
GreenhouseDFNController gdfn = new GreenhouseDFNController();
gdfn.start();

gdfn.setInputOnHouseAtTempSensorInPort(new TemperatureToken(21));
System.out.println(gdfn.getHeatingNodestate().toString());

gdfn.close();
```

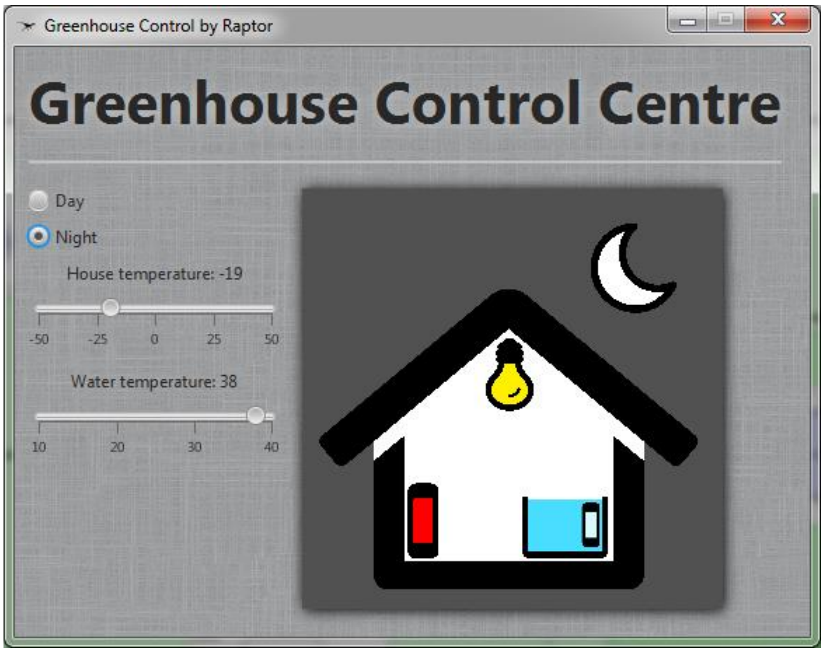
This is a small usage example. You should only create the root DFN, the other needed DFN-s will be managed by their parents. Then you should start it. At this point, the controller library is in connection with the whole DFN, and can control it.

You have the tools to access its states, and modify its inputs. You can also reach deeper DFN-s and modify their inputs through the rood DFN. Finally, you should close the connection.

2.4.4 Controlling the Example DFN network via UI

The example Greenhouse network comes with a JavaFX-based testing UI that uses the interface provided by the controller library. Based on this it is quite simple to create a UI for your own DFN.

After the generation and setup is completed and all of the APP projects are running, run the GreenHouseApp project as well. It will provide you with the following UI:



4. Figure – GreenHouseApp User Interface

Here the image on the right shows the actual state of the house itself. This information is gathered via using the Controller API if the network. The buttons and slides on the left side can be use3d to provide the system with input.

Based on this example UI project it is quite simple to create a similar UI to any other DFN created by the DFN modeler.