EITI

FXU

Framework for eXecutable UML

Marian Szczykulski under the supervision of Anna Derezińska

2009-06-24

This document describes a process of creating an executable application using the FXU framework.

1. About FXU

FXU is a framework used for creating applications according to MDE (Model-driven Engineering) methodology. FXU performs transformations from UML class and state machine diagrams into a C# implementation. FXU is composed of *FXU Code Generator* (*FXU Generator*), *Application Wizard* and *Runtime Environment*.

FXU was initially written by Romuald Pilitowski as a part of his master's thesis: "*Generation of C# code from UML 2.0 class and state machine diagrams*" in 2006. It was designed to collaborate with UML 2.0 models. Moreover, it was not equipped with any graphical interface to create an executable application. The *FXU Generator* worked with EMF 2.1 (Eclipse Modeling Framework) and read an uml2 format of Eclipse in which the UML model was serialized.

In 2009 it was redeveloped by Marian Szczykulski to satisfy the UML 2.1 specification. The *FXU Generator* was modified in order to work with EMF 2.4.1 (Eclipse Modeling Framework). Furthermore, the *FXU Generator* was equipped with a graphical user interface written in *Java-Swing* technology and the *Application Wizard* to improve a process of building an executable application and creating a *Microsoft Visual Studio 2008* project.

FXU:

- The *FXU Code Generator* (*Fxu.jar*) – It is a tool to make a transformation from UML class and state machine diagrams, serialized in the uml format of Eclipse, into the C# implementation.

- The *Application Wizard* – It is a tool integrated with the *FXU Code Generator*. It is useful to create the *Microsoft Visual Studio 2008* project and a *main* function, where state machines can be initialized and started.

- The *Runtime Environment* (*FXU.dll*) – It is an implementation of UML State Machines written in the C# language.

2. User documentation

This section describes how to use FXU to efficiently develop applications according to MDE methodology. The whole process can be described in few steps:

2.1 Creating UML model

A preferred tool to create the UML model is *IBM Rational Software Architect 7.5*, but it can be any other tool, which is able to export a model into the UML 2.1 format of Eclipse (*.uml* extension). In this section the process of creation the UML model using *IBM Rational Software Architect 7.5* is described.

First, create the UML model containing a class diagram (Figure 1) and state machine diagrams (Figure 2).

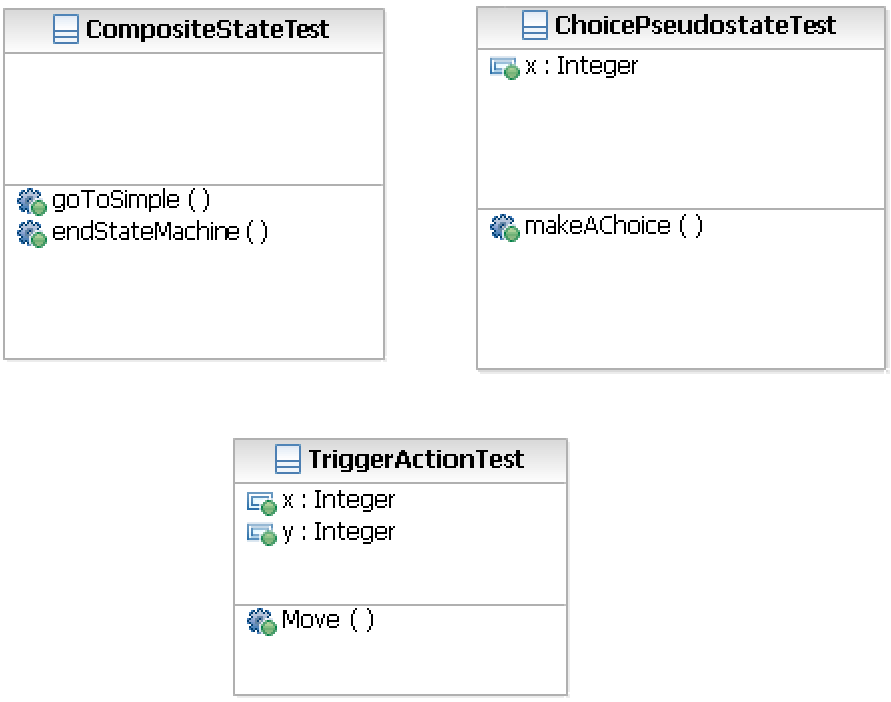


Figure 1. Simple example of class diagram.

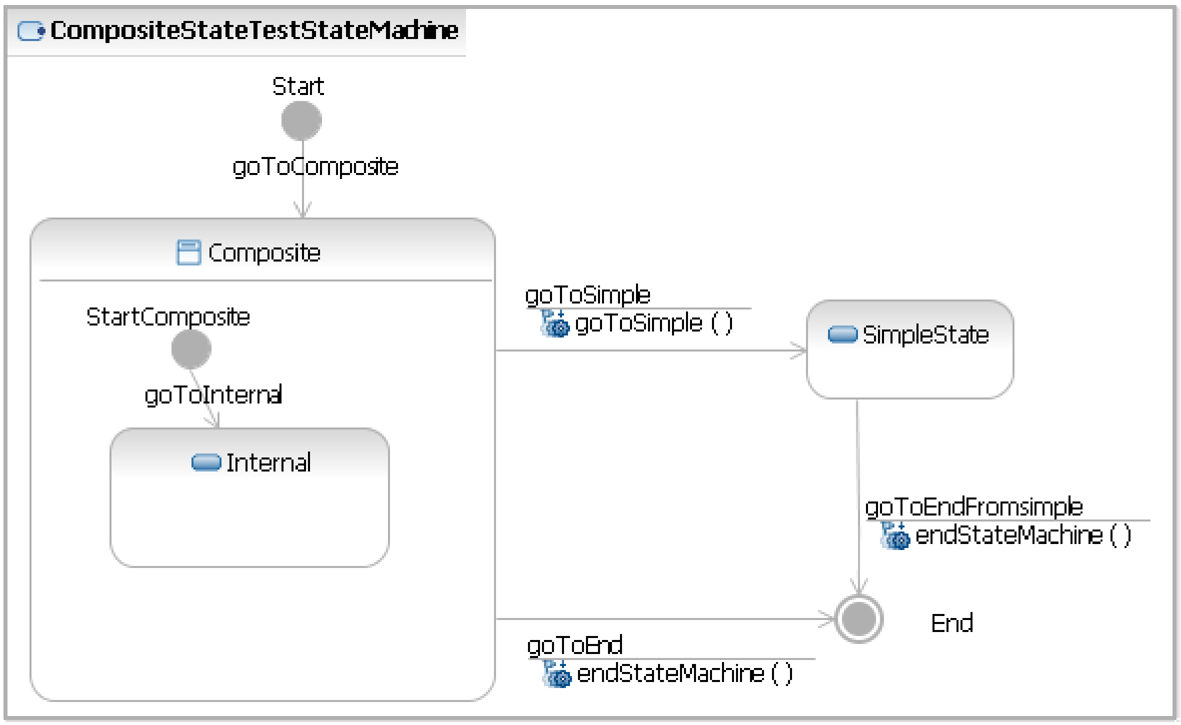
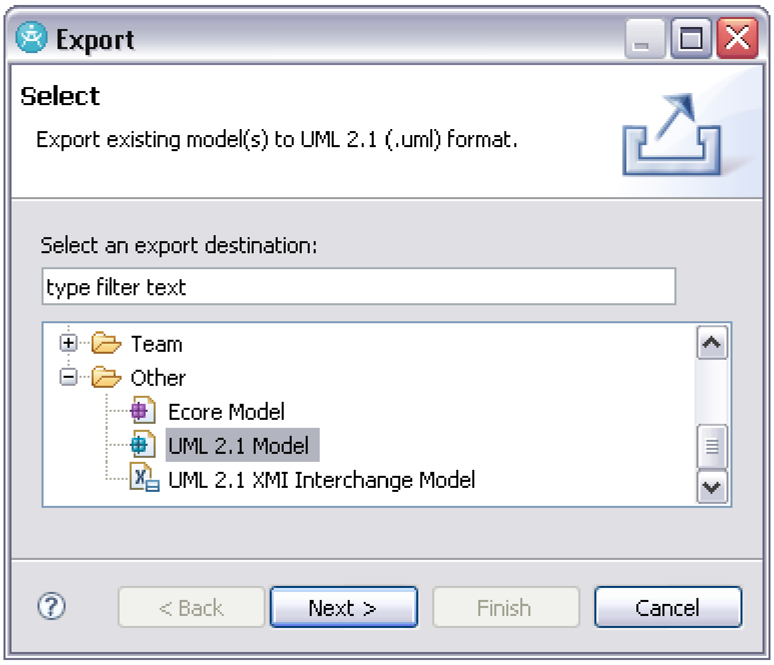


Figure 2. Simple example of state machine diagram.

Next, export the model to the UML 2.1 format. Click “*File*”->”*Export*” and an *Export Window* is opened. Then choose “*Other*”->”*UML 2.1 Model*” and click “*Next*“ button (Figure 3).



Export

Select

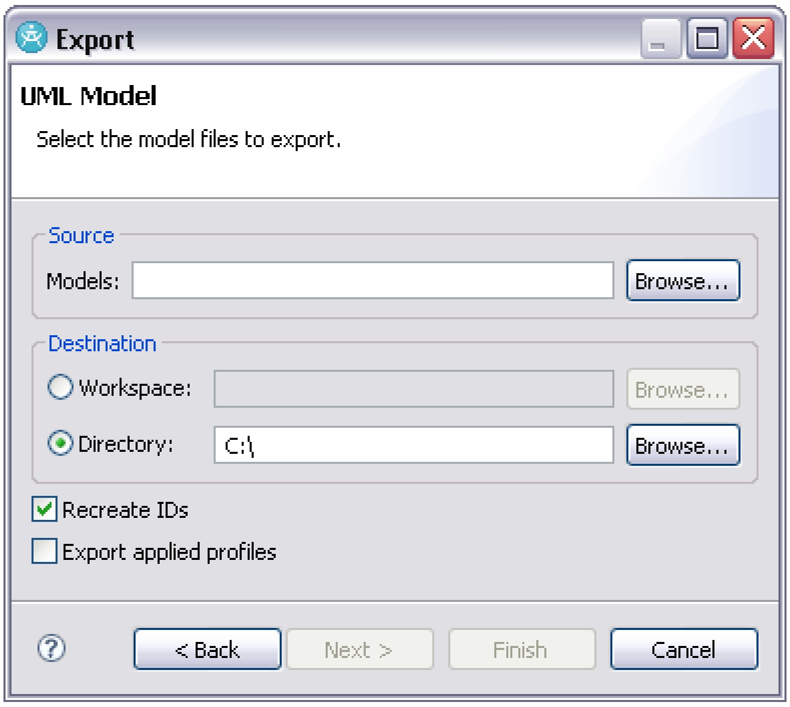
Export existing model(s) to UML 2.1 (.uml) format.

Select an export destination:

type filter text

Figure 3. Export Window in IBM Rational Software Architect 7.5

On the next window specify the UML model to export in *Source* section and a destination path in *Destination* section (Figure 4).


Export

UML Model

Select the model files to export.

Source

Models:

Destination

( ) Workspace:

(●) Directory: C:\

[✓] Recreate IDs

[ ] Export applied profiles

Figure 4. Export window in IBM Rational Software Architect 7.5 - choosing source and destination.

Now the UML2.1 model file is created in the specified path. It will be used in next steps to generate a C# code.

2.2 Launching the FXU Generator

First, start FXU by double-clicking an *Fxu.jar* file. The main window of the *FXU Generator* is displayed (Figure 5). It is composed of tree panels. The left panel is designed to hold a list of loaded UML models. The right panel is designed to hold trees which are visualizations of UML models. The bottom panel holds a text area, where logs about generation and validation results are displayed. There is also a menu bar and a tool strap, which are designed to manage a process of code generation. They are described above on next sections.

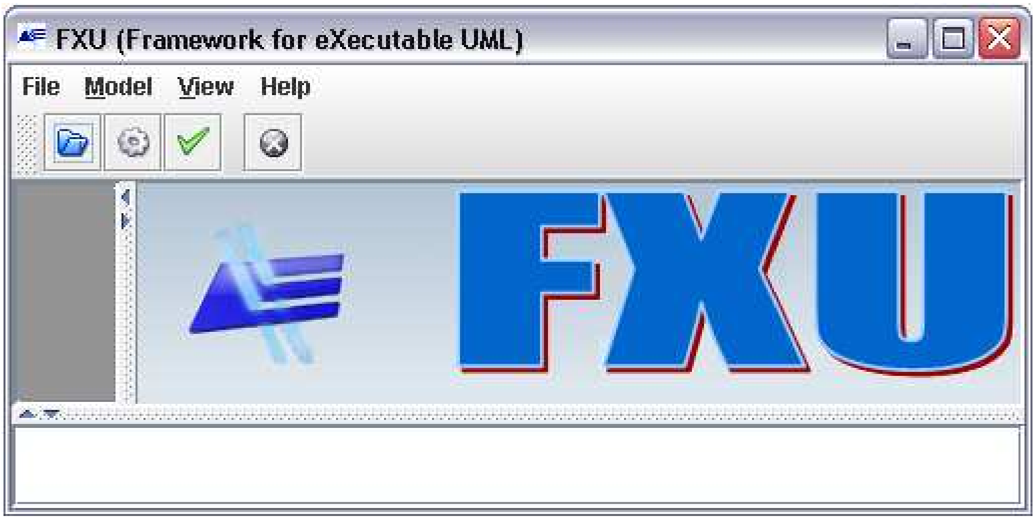
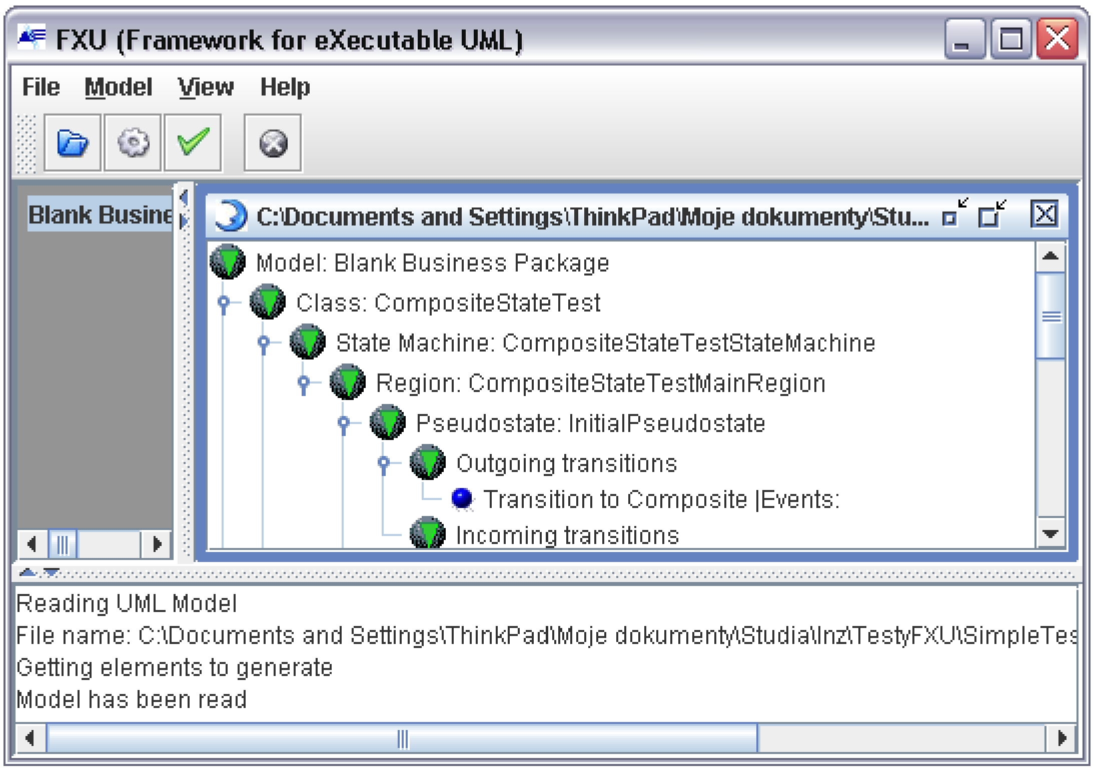


Figure 5. The main window of FXU Generator.

2.3 Loading UML model to FXU

Click "*File*"->"*Open*" and select an UML2.1 model file. It may take few seconds to load the UML model. Model is displayed as tree containing all nodes specified in the UML 2.1 model file (Figure 6).


FXU (Framework for eXecutable UML)
File Model View Help
Blank Busine
C:\Documents and Settings\ThinkPad\Moje dokumenty\Stu...
Model: Blank Business Package
Class: CompositeStateTest
State Machine: CompositeStateTestStateMachine
Region: CompositeStateTestMainRegion
Pseudostate: InitialPseudostate
Outgoing transitions
Transition to Composite |Events:
Incoming transitions

```
Reading UML Model
File name: C:\Documents and Settings\ThinkPad\Moje dokumenty\Studia\Inz\TestyFXU\SimpleTes
Getting elements to generate
Model has been read
```

Figure 6. FXU Generator - the UML model is loaded.

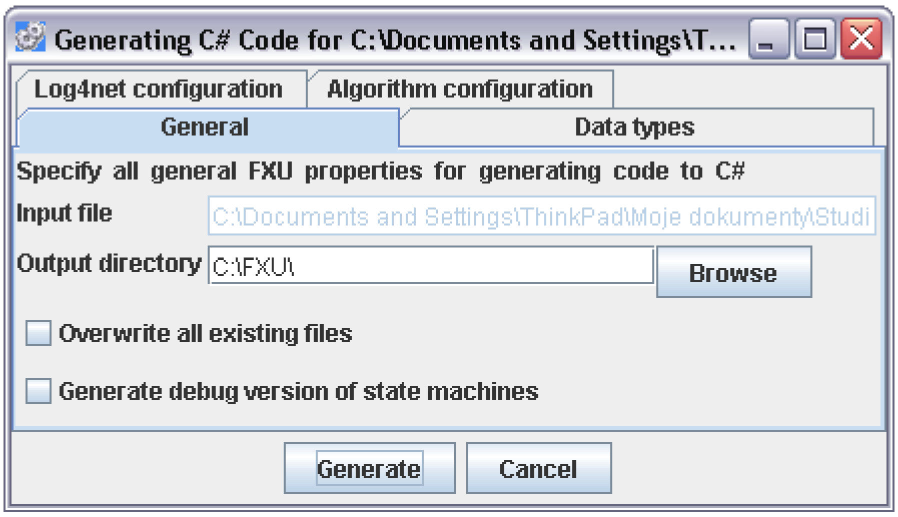
Now, the model has been read and is ready for validation and code generation.

2.4 Validating the UML Model

This is an optional step, because the model is being validated always before code generation. Click “*Model*”->”*Validate Model*”. FXU will display appropriate information message about validation result.

2.4 Generating C# Code

Click “*Model*”->”*Generate C# Code*”. The *Generation Window* will be displayed (Figure 7).


Generating C# Code for C:\Documents and Settings\T...

Log4net configuration | Algorithm configuration | General | Data types

Specify all general FXU properties for generating code to C#

Input file: C:\Documents and Settings\ThinkPad\Moje dokumenty\Studi

Output directory: C:\FXU\

Browse
Overwrite all existing files
Generate debug version of state machines
Generate
Cancel

Figure 7. The Generation Window of FXU Generator.

The *Generation Window* is composed of four tabs. They are designed to configure *FXU Generator* and generated elements. They are briefly described above.

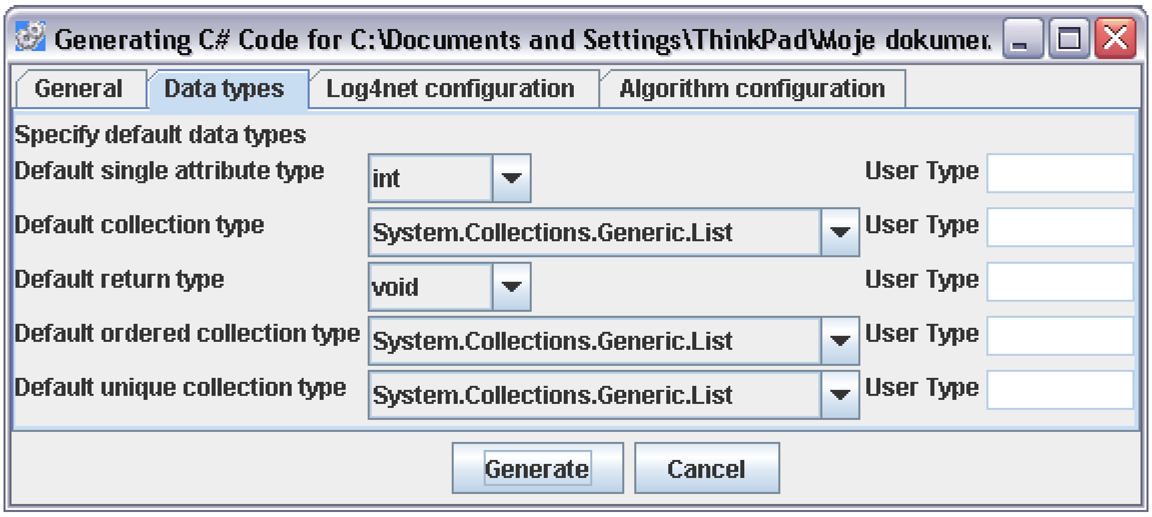
The “*General*” tab (Figure 7) is composed of following elements:

- The “*Input file*” text field – it cannot be changed and it holds the path to the UML model file.

- The "*Output directory*" text field – directory where the generated C# code will be placed.
- The "*Overwrite all existing files*" check box – select to overwrite all existing files in the output directory.
- The "*Generate debug version of FXU Environment*" check box – select to generate application with the *FXU Runtime Environment*, which enable tracing of a state machines execution.

The "*Data types*" tab (Figure 8) contains information about default data types used in the generated C# code:

- Default single attribute type – possible values: *int*, *double*, *object*, *string*, *decimal*, *bool*, *char*, *byte*, *sbyte*, *short*, *long*, *ulong*, *single*, *float* and *User type*, which is specified on the text field.
- Default collection type – possible values: *System.Collections.Generic.List*, *System.Collections.Generic.LinkedList*, *System.Collections.Generic.SortedList*, *System.Collections.Generic.Queue*, *System.Collections.Generic.Stack* and *User type*, which is specified on the text field.
- Default return type – possible values: *void*, *int*, *double*, *object*, *string*, *decimal*, *bool*, *char*, *byte*, *sbyte*, *short*, *long*, *ulong*, *single*, *float* and *User type*, which is specified on the text field.
- Default ordered collection type – possible values: *System.Collections.Generic.List*, *System.Collections.Generic.LinkedList*, *System.Collections.Generic.SortedList*, *System.Collections.Generic.Queue*, *System.Collections.Generic.Stack* and *User type*, which is specified on the text field.
- Default unique collection type – possible values: *System.Collections.Generic.List*, *System.Collections.Generic.LinkedList*, *System.Collections.Generic.SortedList*, *System.Collections.Generic.Queue*, *System.Collections.Generic.Stack* and *User type*, which is specified on the text field.


Generating C# Code for C:\Documents and Settings\ThinkPad\Moje dokumer

General | Data types | Log4net configuration | Algorithm configuration

Specify default data types

Default single attribute type	int	User Type	
Default collection type	System.Collections.Generic.List	User Type	
Default return type	void	User Type	
Default ordered collection type	System.Collections.Generic.List	User Type	
Default unique collection type	System.Collections.Generic.List	User Type	

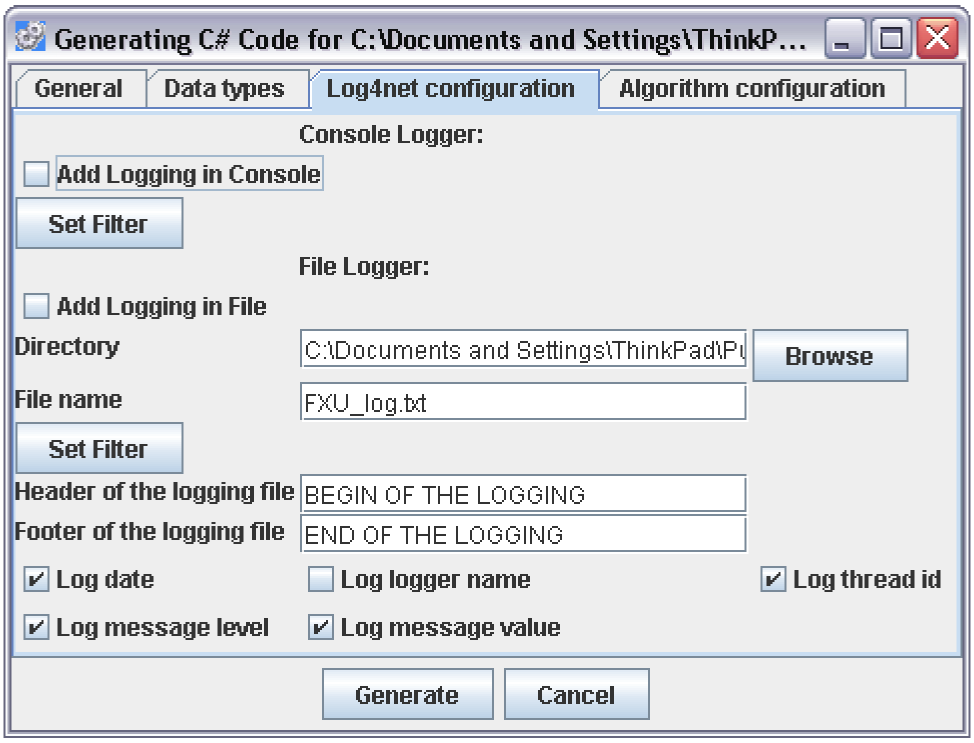
Figure 8. The Generation Window of FXU Generator - Data types tab.

The “*Log4net configuration*” tab (Figure 9) is responsible for a configuration of a log4net library. The log4net library logs all events during the execution of state machines in the generated application. It is composed of following elements:

- The “*Add Logging in Console*” check box – specify if logs should be visible in a console after launching the generated application.
- The “*Set Filter*” button in “*Console Logger*” section – set a filter configuration of the console logger.
- The “*Add logging in file*” check box – specify if logs should be placed in a file.
- The “*Directory*” text field – specify the directory where the log file is created.
- The “*File name*” text field – specify the log file name.
- The “Set Filter” in “*File Logger*” section – set a filter configuration of the file logger.
- The “*Header of logging file*” text field – specify the header of the log file.
- The “*Footer of logging file*” text field – specify the footer of the log file.
- The “*Log date*” check box – specify if date should be logged in the log file.
- The “*Log message level*” check box – specify if log level should be logged in the log file.
- The “*Logger name*” check box – specify if the logger name should be logged in the log file.
- The “*Log message value*” check box – specify if logging messages should be logged in the log file.

- The "*Log thread id*" check box – specify if a thread identifier should be logged in the log file.

Note, that this data types are used only if they are not specified in the UML model created at first step.


Generating C# Code for C:\Documents and Settings\ThinkP...

General | Data types | Log4net configuration | Algorithm configuration

Console Logger:

[ ] Add Logging in Console

Set Filter

File Logger:

[ ] Add Logging in File

Directory: C:\Documents and Settings\ThinkPad\Pu

Browse

File name: FXU_log.txt

Set Filter

Header of the logging file: BEGIN OF THE LOGGING

Footer of the logging file: END OF THE LOGGING

[✓] Log date [ ] Log logger name [✓] Log thread id

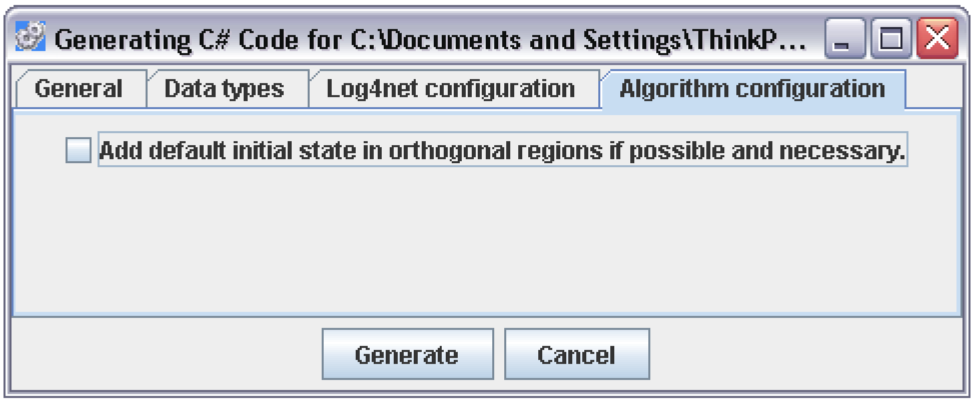
[✓] Log message level [✓] Log message value

Generate | Cancel

Figure 9. The Generation Window of FXU Generator – The Log4net configuration tab.

The "Algorithm configuration" tab (Figure 10) is responsible for a configuration of a generation schema in the *FXU Generator*. It is composed of following elements:

- The "*Add default initial state in orthogonal regions if possible and necessary*" check box – specify if the *FXU Generator* is supposed to correct an orthogonal state if any region of this state does not contain an initial state.



☐ Add default initial state in orthogonal regions if possible and necessary.

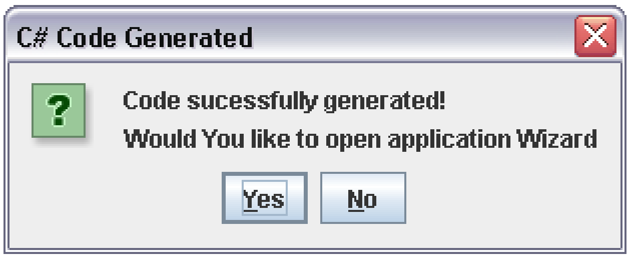
Figure 10. The Generation Window of FXU Generator - The Algorithm Configuration tab.

Click a "*Generate*" button in order to start the generation process. There are created two subdirectories in the output directory:

- *lib* - it contains:
  - *FXU.dll*
  - *FXU.dll.manifest*
  - *log4net.dll*
  - *log4net.dll.xml*
- src - it contains namespace directories, where C# files are located. For the UML model on Figure 1 and Figure 2 the *src* directory contains:
  - *Blank_Business_Package*
    - *ChoicePseudostateTest.cs*
    - *CompositeStateTest.cs*
    - *TriggerActionTest.cs*

2.4 Generating the Microsoft Visual Studio 2008 project

After the C# code generation completion a question about starting the *Application Wizard* is displayed (Figure 11).



C# Code Generated

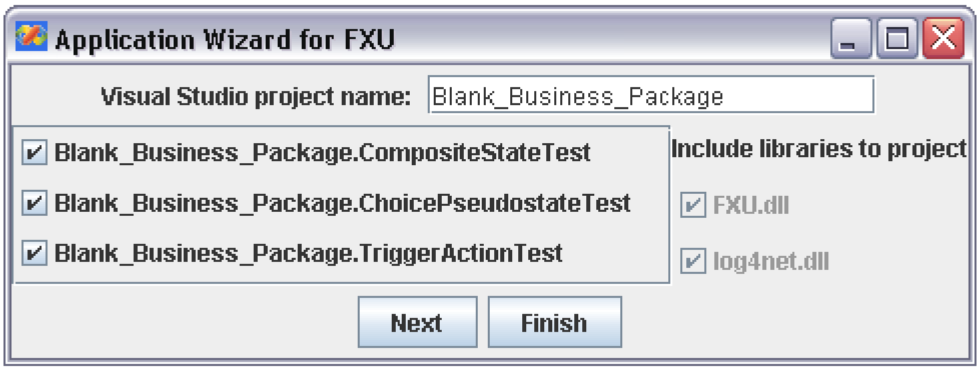
Code sucessfully generated!
Would You like to open application Wizard

Figure 11. The Question Window of the Application Wizard.

The *Application Wizard* creates the *Microsoft Visual Studio 2008* project in the output directory. At first step of the project creation (Figure 12):

- Specify the project name. A default value is retrieved from the UML model.
- Select classes that the project contains. By default all classes from the UML model are included.

Note, that the FXU.dll library and the log4net.dll library are mandatory and they cannot be removed from the project.



Application Wizard for FXU

Visual Studio project name: Blank_Business_Package

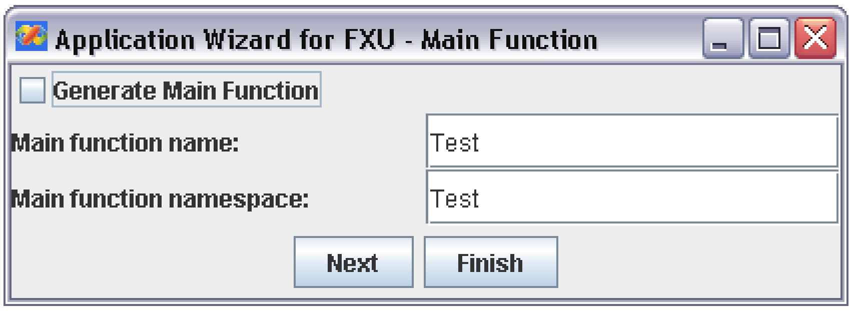
☑ Blank_Business_Package.CompositeStateTest
☑ Blank_Business_Package.ChoicePseudostateTest
☑ Blank_Business_Package.TriggerActionTest

Include libraries to project
☑ FXU.dll
☑ log4net.dll

Figure 12. The Application Wizard - the first window.

At second step select if the *Main* function is to be generated (Figure 13). If yes:

- Specify a name of a class containing the *Main* function – the class is created by the *Application Wizard*.
- Specify a name space of the class containing the *Main* function.



Generate Main Function [ ]

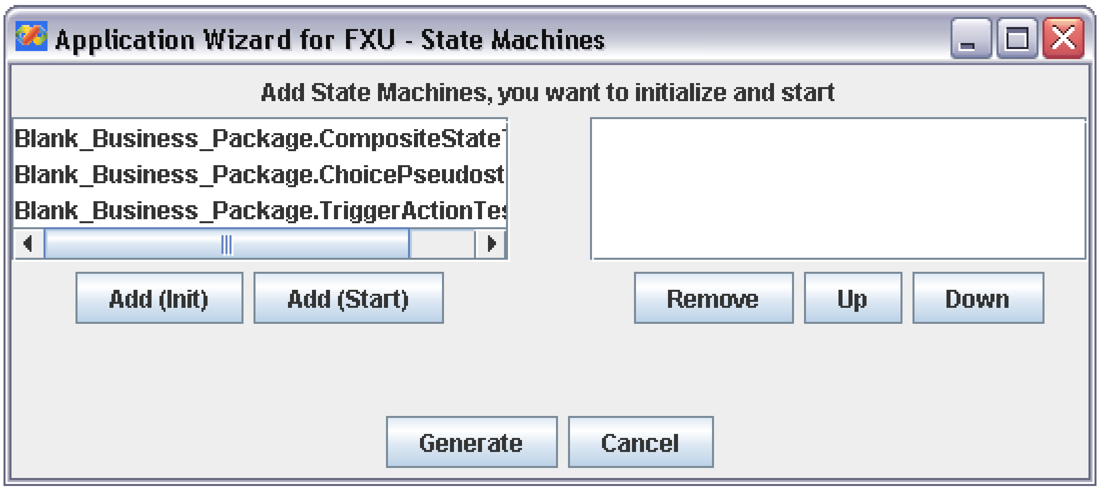
Main function name: Test

Main function namespace: Test

Figure 13. The Application Wizard - the second window.

At third step, specify state machines, which will be initialized and started. Select appropriate class containing a state machine and click “*Add (Init)*” button. Then select appropriate class containing a state machine and click “*Add (Start)*” button (Figure 14).

It is possible to specify an order of the initialization and starting state machines. Note, that starting state machine must be placed after the initialization this state machine.



Add State Machines, you want to initialize and start

Blank_Business_Package.CompositeState
Blank_Business_Package.ChoicePseudost
Blank_Business_Package.TriggerActionTe

Figure 14. The Application Wizard - the third tab.

Click “*Generate*” button. The project is created and ready to open and run in the *Microsoft Visual Studio 2008*. The *Application Wizard* creates:

- Microsoft Visual Studio files:
  - Blank_Business_Package.csproj - project file
  - Blank_Business_Package.sln - solution file
- Optionally, C# files with the *Main* function and namespace directories