EITI

FXU 3.0

Framework for eXecutable UML 3.0

Marian Szczykulski under the supervision of Anna Derezińska

2011-01-29

This document describes a process of creating an executable application using the FXU framework and shows changes introduced in the new version of the tool.

1. Goals of document

This document presents changes in *Framework for eXecutable UML* (FXU), that were performed as a part of release 3.0. In the next section all new features are listed, as well as bugs from previous version which have been detected and fixed.

2. FXU 3.0 – release notes

The main feature of the new version of FXU is the possibility of generating additional source code for stereotypes applied in a UML model. In the previous version of the framework stereotypes were ignored. FXU was able to read them from the serialized UML model. Hence, the whole algorithm of reading the model and the architecture of FXU was changed. In order to interpret stereotypes and generate additional code, the separate components are provided. One single component is responsible for stereotypes from certain UML profile and there is another component for another profile, but the FXU is independent from those components. Therefore, if they are not provided, the FXU is still able to generate C# code, but stereotypes are just ignored and included as comments. Other important features are new generation options which allow to configure *FXU Runtime Environment*'s algorithms in order to optimize the execution of state machines. All changes are listed in the next subsections.

2.1. New features

- ***New generation option:***
  - **Priority event queue** – there is a possibility to choose priority event queue and specify priorities for different event types during code generation.
  - **Events filtration** – there is a possibility to choose a filtration option for UML events during code generation. Thanks to this option, events are not sent to every state machine in the application, but only to those where these events are really used.
  - **"After" event optimization** – this option allows to run every "after" event for a state in one single thread instead of separate threads for every "after" event.
- ***Plug-in architecture for code generation from stereotypes and tag values*** – a dedicated architecture allows to implement and add new components (plug-ins) into

the FXU Generator in order to generate C# code from stereotypes and tag values of a certain UML profile.

- ***Components for MARTE::Time profile*** – a component (plug-in) generates additional C# code for a subset of stereotypes and tag values from *MARTE::Time* profile. Moreover, the *FXU Runtime Environment* was equipped with library that supports the execution of the code generated by *MARTE::Time* plug-in.
- ***Invocation of methods in Main function*** – the *FXU Application Wizard* was equipped with a feature which allows to specify operation (with its parameters) to invoke in them *Main* function.

2.2. Other improvements

Apart from the new features described in the previous section some bugs were detected and fixed. The most important are listed below:

- ***Considering classes generated from UML signals in the FXU Application Wizard –*** the previous version of the *FXU Application Wizard* does not consider classes generated from signals in a UML model. Therefore, in some cases, a generated project does not contain all elements.
- ***Improved model visualization –*** a tree which represents a UML model in the FXU GUI was improved and enriched with new elements in order to visualize the original UML model more precisely.
- ***Improved error handling*** – error situations are described more precisely.
- ***Adjustment FXU to the UML 2.3*** – changes introduced in the UML 2.3 have been taken into account and implemented in the new version of FXU.
- ***Optimization of FXU Runtime Environment's algorithms*** – the *FXU Runtime Environment* was modified in order to create fewer threads and move some time-consuming computations to the code generation phase.

3. User guide

In this section there is presented a simple example, which shows how to use the FXU Generator in order to generate a C# code from an UML model enriched with *MARTE::Time* stereotypes and tag values.

3.1. Create an UML model with stereotypes

In the first step, an UML model has to be created. In the example *IBM Rational Software Architect 7.5.1* has been used, but it is possible to use any CASE tool which supports exporting the UML model into the UML format of Eclipse (uml 2.1 format).

At the beginning, a class diagram has to be created (Figure 1). In the example there are two classes: *MarteTimedEventTest*, which has neither operations nor attributes, and *SimpleClock* class in a package *clocks*. The second class has a *ClockType* stereotype from the *MARTE::Time* profile, which tag values are shown in the frame in the picture below. Moreover the package *clocks* has a *TimedDomain* stereotype.

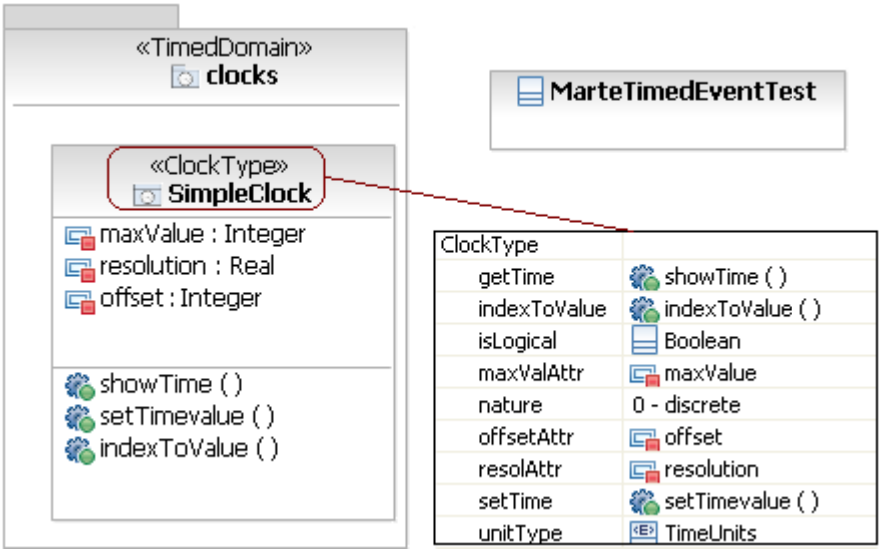

Figure 1. Exemplary class diagram.

The *SimpleClock* class defines a new type of clock. The stereotype *ClockType* and its tag values specify properties of this kind of clock. But in order to use this type of clock it is necessary to create an instance of the class. Therefore, in the next step, an object diagram for the *clocks* package is created (Figure 2). Only one object is created and it will be used by the exemplary state machine.

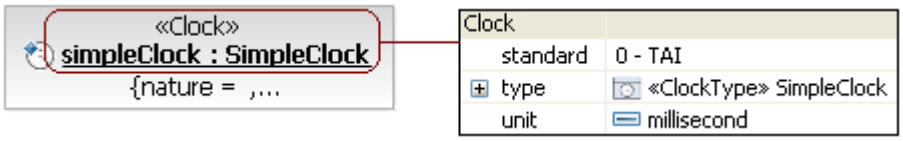

Figure 2. Exemplary object diagram.

The third part of the UML model is a state machine diagram for the *MarteTimedEventTest* class (Figure 3). It consists of four simple states. The transition between those states are triggered by a time event which has a *TimedEvent* stereotype from the *MARTE::Time* profile. The first event is generated after 2000 units of time on a clock specified by "*on*" tag value of the stereotype *TimedEvent*. Next events will be generated after every 5000 units of time, as it is specified by "*every*" tag value.

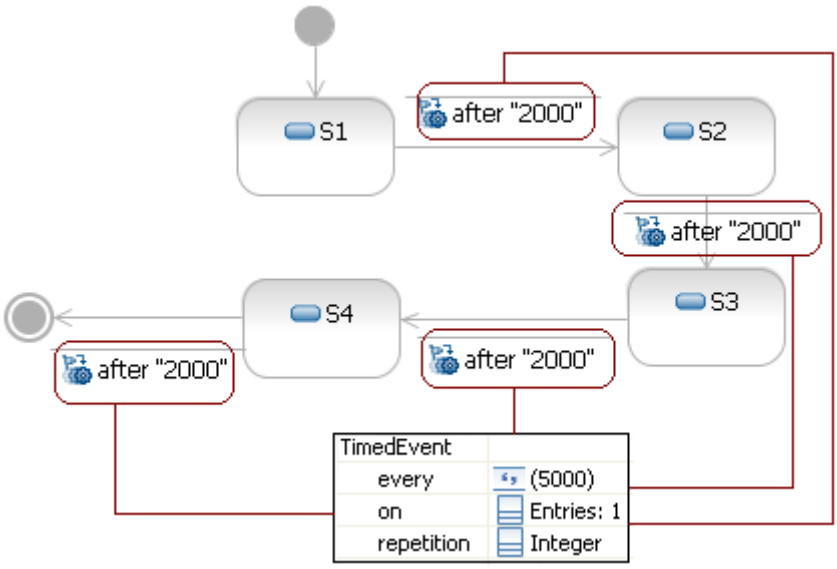

Figure 3. Exemplary state machine diagram.

The created model has to be exported to the UML format of Eclipse. It can be done by clicking "*File->Export*", selecting "*UML 2.1 Model*" on the "*Other*" section and following wizard's steps. It is important to select an "*Export applied profiles*" option in the second frame. Thanks to this, the *FXU Generator* will be able to read all applied profiles in the model and generate appropriate C# code.

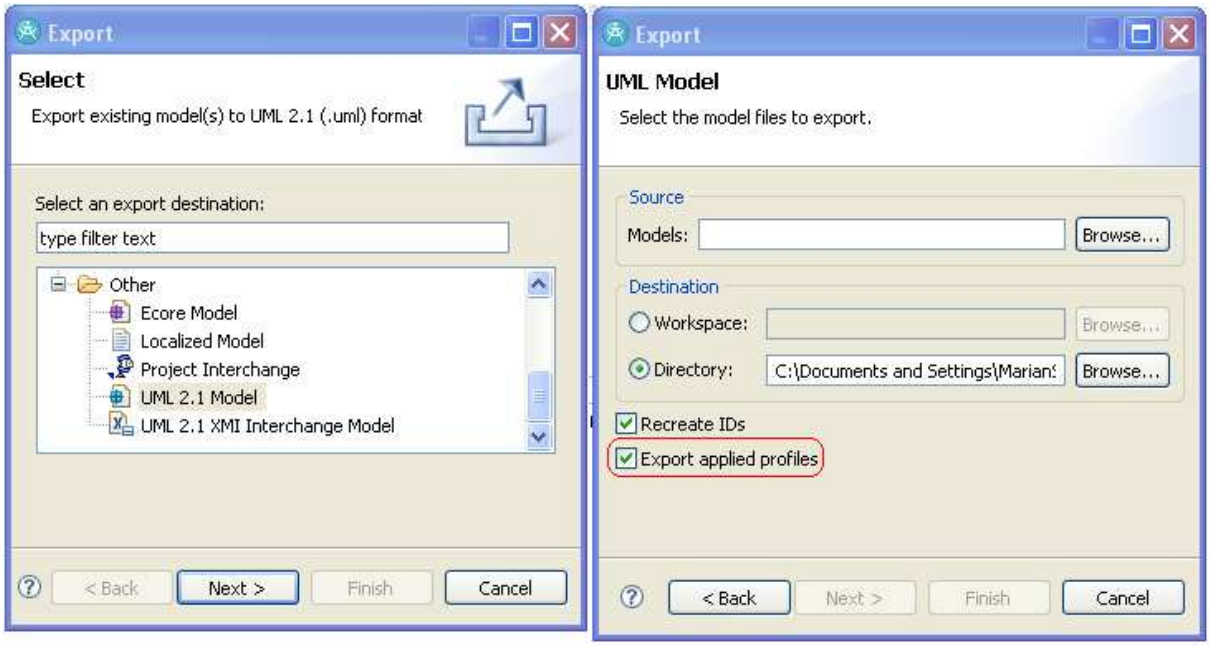

Figure 4. RSA 7.5.1 Export Wizard.

3.2. Launching the FXU Generator and loading the UML model

The generator can be launched by double clicking the "*FXUGenerator3.0.jar*" file. Before launching the generator, ensure yourself that the *MARTE::Time* plug-in is provided by checking the "*FXUGenerator3.0_lib*" folder. It should contain the "*MarteTime.jar*" file.

In order to load the model click "*File->Open*" and select a file with the exported model in your file system.

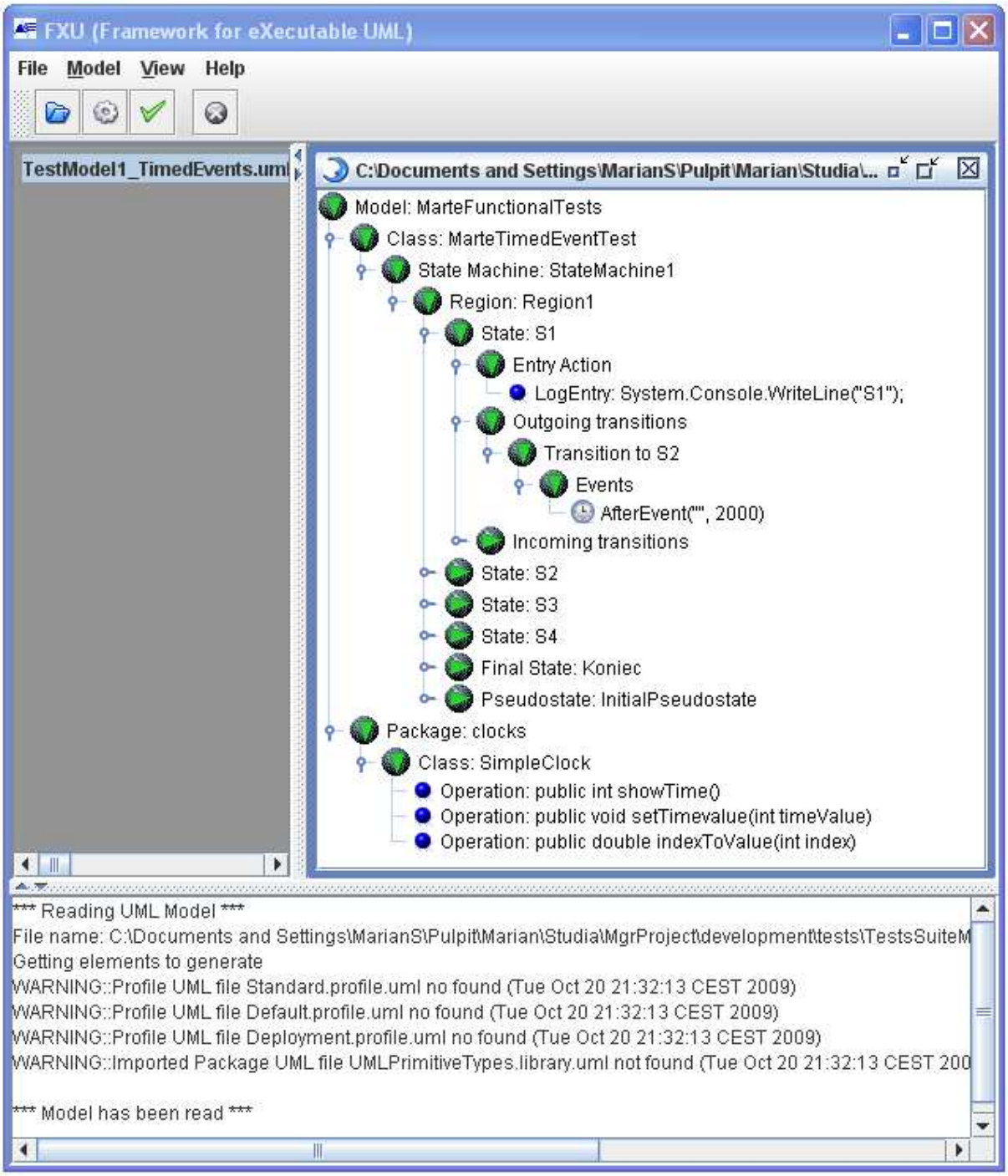

Figure 5. An UML model loaded to the FXU Generator.

Now, the model is loaded and visualized in a tree-like form which represents the real hierarchy in the original UML model (Figure 5). It is important to check logs in the result frame at the bottom of the main window. There are a few warnings. They indicate that the FXU loader has not been able to find files with UML profiles (*Standard.profile.uml*, *Default.profile.uml*, *Deployment.profile.uml*), which are applied in the model. In this case, stereotypes from these profiles were not used, so warnings can be ignored.

There is a possibility to choose how detailed the view of the UML model should be by clicking "*View*" on menu bar. There are three options possible:

- *Simple view* – the tree, that represents the UML model, contains only information about names of classes, state machines and main regions in these state machines.
- *Standard view* – the tree, that represents the UML model, contains information about names of classes and theirs operations. Moreover there are detailed information about every state machine in the model (states, transitions, events, guards, pseudostates).
- *Advanced view* – apart from information available in standard view, there are also information about generalization of classes, attributes in classes and stereotypes in packages, classes and state machines.

3.3. Validating the UML model

In order to perform model validation click "*Model->Validate Model*". The dialog window with a message "*Model is valid*" should appear. In the result frame, at the bottom of the main window, additional messages should appear. Apart from messages from the FXU model loader, described earlier, there is one warning:

WARNING::Parameter
<MarteFunctionalTests::clocks::SimpleClock::indexToValue::value> - Name of the element is a C# contextual keyword.

It indicates that there is a parameter named "*value*", which is a C# keyword. But in this case, this parameter is a return parameter of an operation *SimpleClock::indexToValue* and it will not appear in the generated code. Therefore these warnings can be ignored..

3.4. Generating C# code

In order to generate C# code from the loaded model click "*Model->Generate C# Code*". In the generation window there is a possibility to configure some features of the generated code such as algorithms for the *FXU Runtime Environment*, a destination path, default data types or an application logger configuration. Next four pictures show every tab in the generation window.

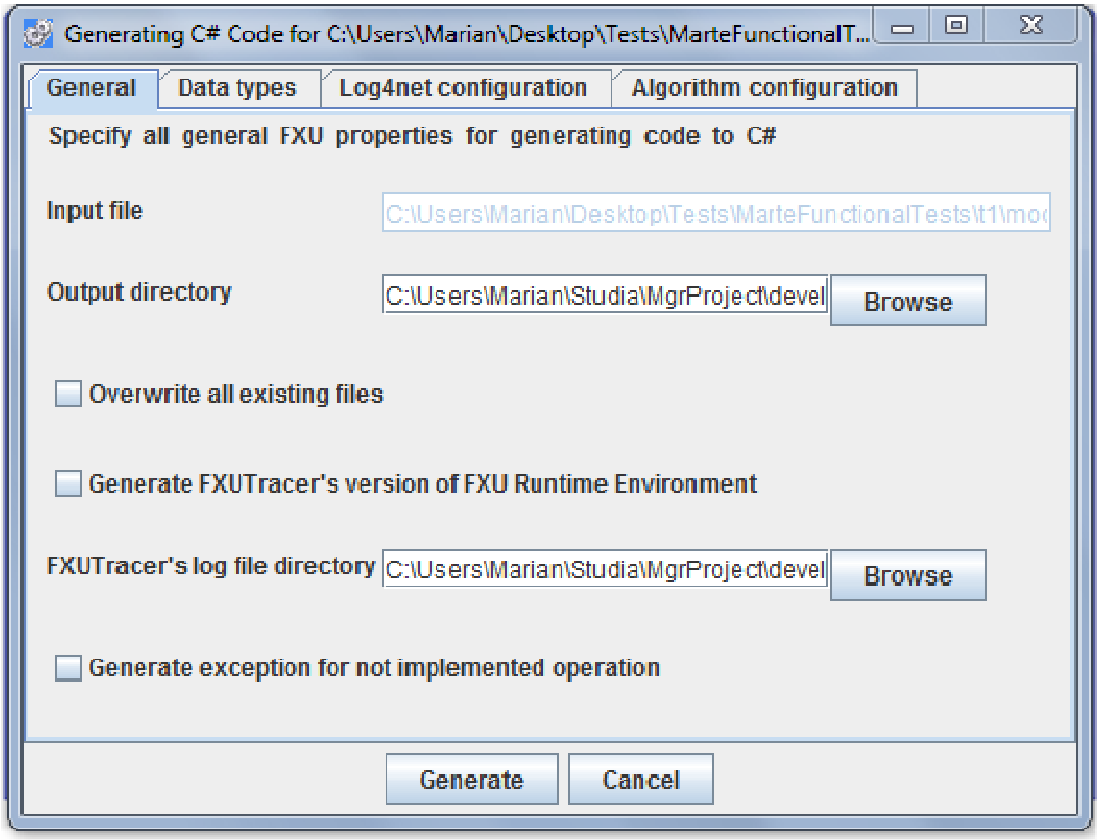

Figure 6. FXU Generator - general properties.

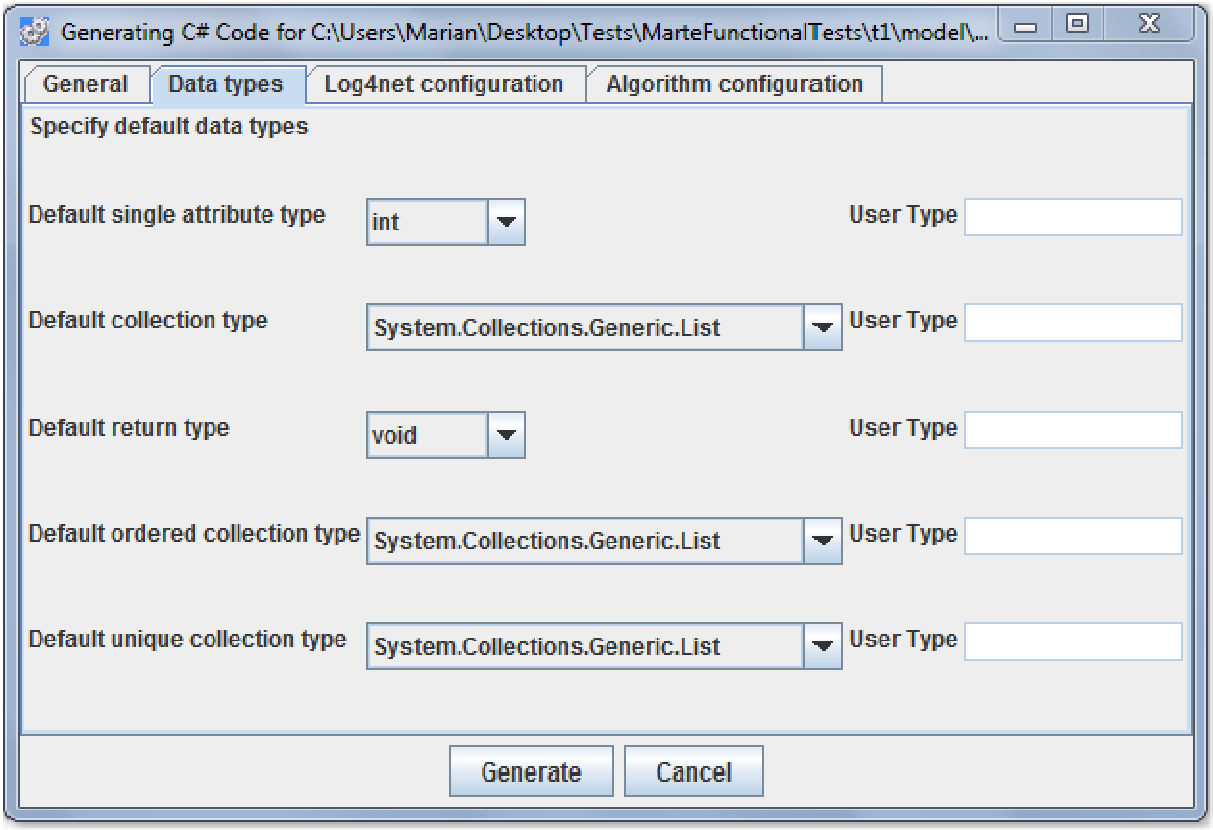

Figure 7. FXU Generator - default data types.

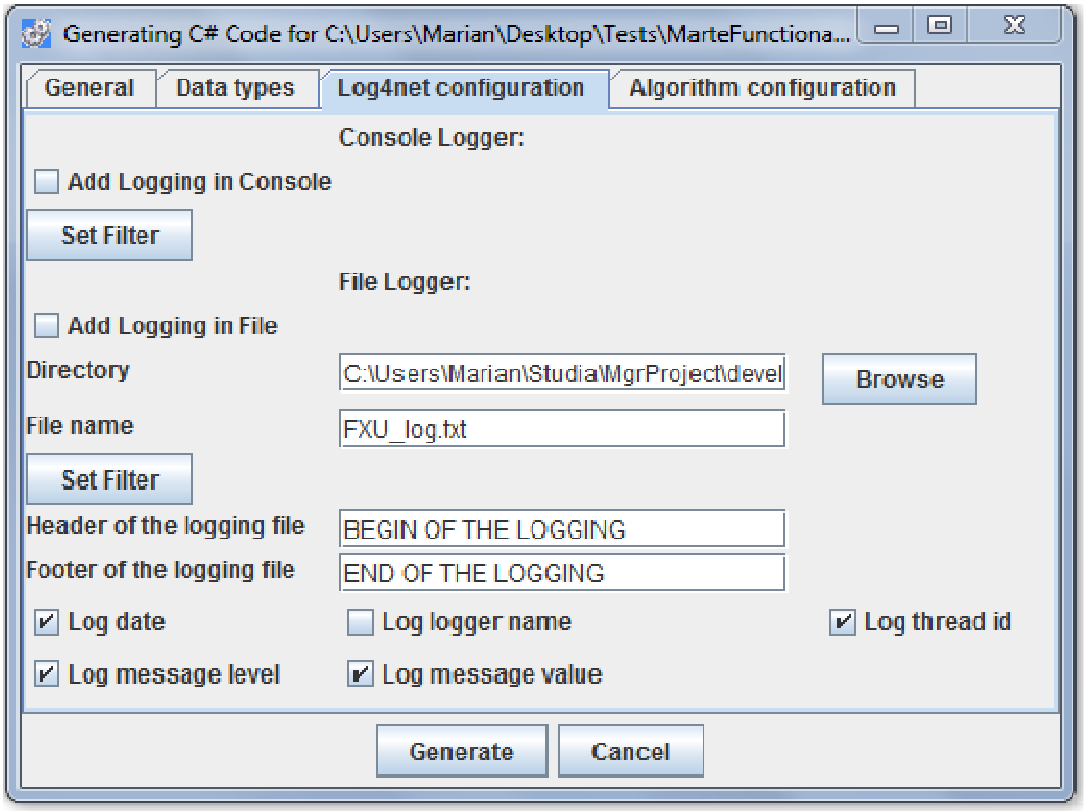

Figure 8. FXU Generator - log4net properties.

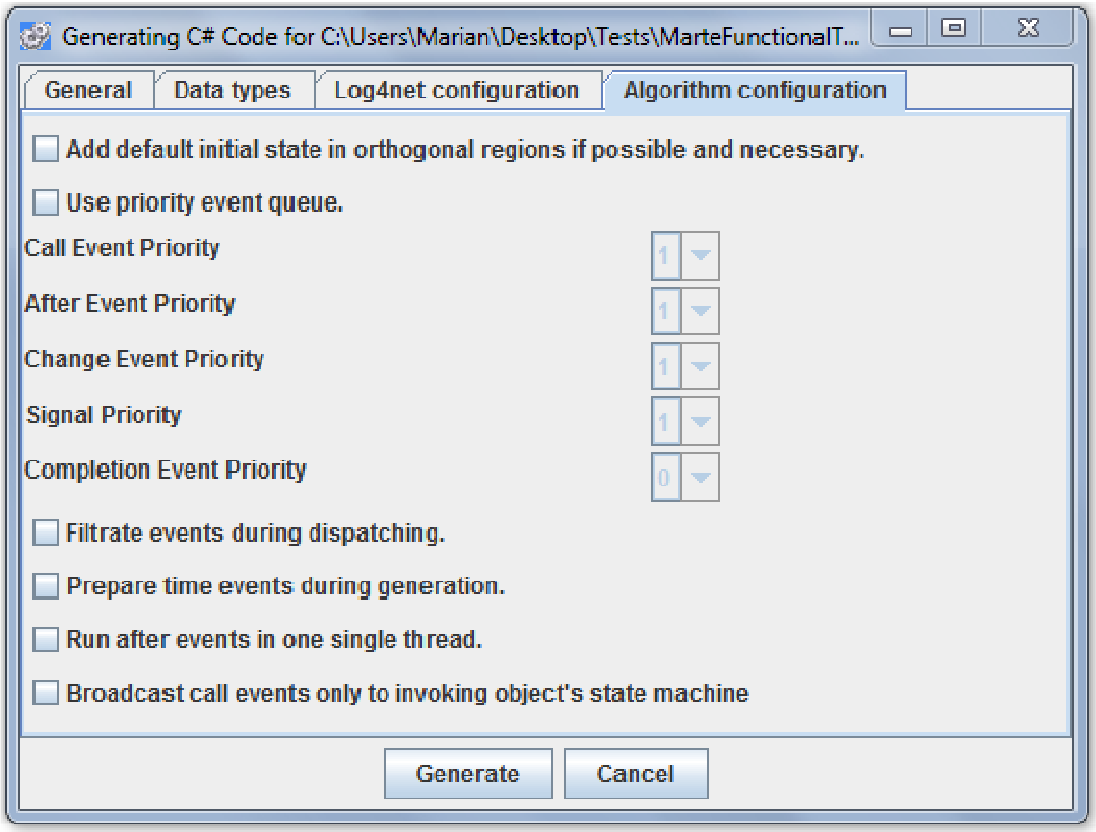

Figure 9. FXU Generator - algorithm properties.

Default values can be used and by clicking the "*Generate*" button C# code can be generated. However, sometimes it would be helpful to choose other possibilities. They are listed and described shortly below.

1) **First Tab (*General*):**
   - *Output directory* – indicates a place in the file system where generated C# code will be stored.
   - *Overwrite all existing files* – if selected, all existing files of the same name in the output directory will be replaced by new ones.
   - *Generate FXU Tracer's version of FXU Runtime Environment* – if selected, there will be application generated with the *FXU Runtime Environment*, which enables tracing of a state machines execution**.**
   - *Generate exception for not implemented operation* – if selected, there will be exceptions generated for unimplemented methods. If not selected, dummy return values for unimplemented methods will be generated **(new option in the FXU 3.0).**
2) **Second Tab (*Data Types*):**
   - *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.

3) Third Tab (*Log4net configuration*):

- 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.

4) Fourth Tab (*Algorithm configuration*):

- *Add default initial state in orthogonal regions if possible and necessary* – if selected, the generator will correct the error situation when an orthogonal region has no initial state.
- *Use priority event queue* – if selected, a priority event queue will be used instead of a FIFO queue in the *FXU Runtime Environment*. There is also a possibility to

configure priorities for different event types (4 is the highest) **(new option in the FXU 3.0).**

- *Filtrate events during dispatching* – if selected, events will be dispatched only to those state machines in which they are really used **(new option in the FXU 3.0).**
- *Prepare time events during generation* – if selected, time events for a state will be calculated during the code generation phase, but not during an application execution **(new option in the FXU 3.0).**
- *Run "after" event in one single thread* – if selected, there will be one single thread for all "after" events that might occur in a state instead of separate thread for every "after" event **(new option in the FXU 3.0).**
- *Broadcast call events only to invoking object's state machine* – if selected, a call event for a non-static operation in a class will be dispatched only to the state machine of the object that has invoked the operation **(new option in the FXU 3.0).**

In order to start a C# code generation process, click the "*Generate*" button. If generation process is completed an appropriate message dialog window will appear (Figure 10).

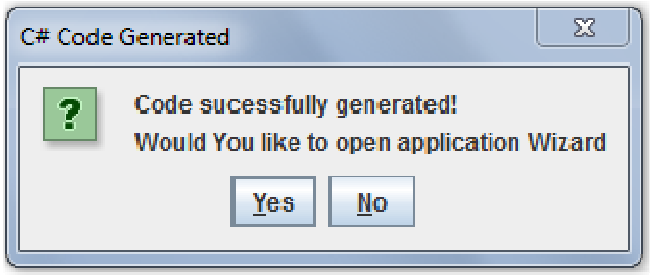

Figure 10. The message window after successful code generation.

3.5. Creating the Microsoft Visual Studio 2008 project

The last step is a *Microsoft Visual Studio 2008* project generation. It is an optional step and can be omitted. Figure 11 shows first three windows of the *FXU Application Wizard*. There is a possibility to specify a project name, to generate the *Main* function and specify its containing class namespace and name. In the third window of the wizard, selected state machines can be initialized and started.

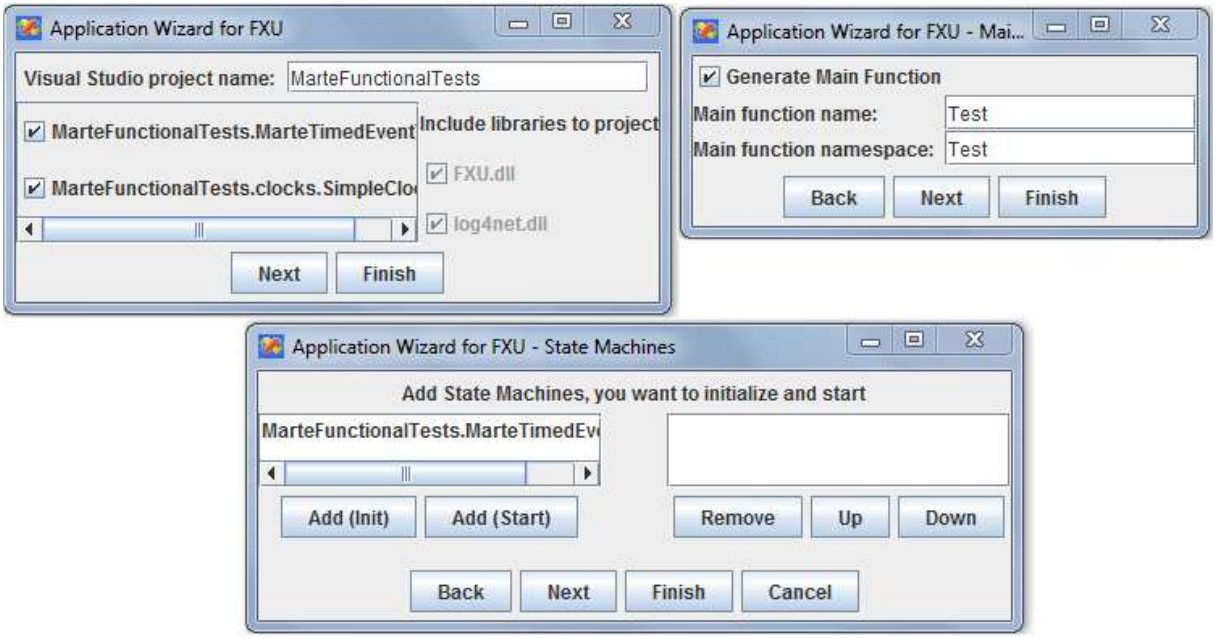

Figure 11. First three windows of Application Wizard.

In the last window of the wizard there is a possibility to specify which operations have to be invoked in the *Main* function. There is also a possibility to configure attributes for invoked operations. The window is shown in the figure 12.

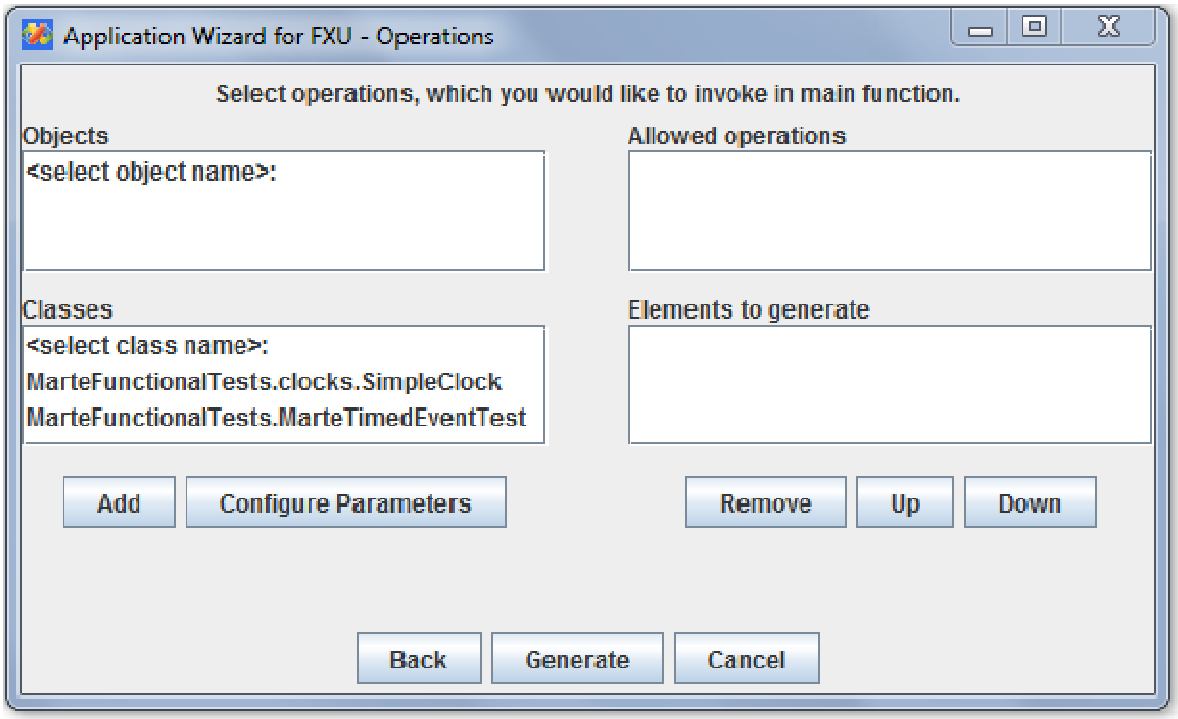

Figure 12. Application Wizard - configuration of operations in the *Main* function.

In order to generate the *Microsoft Visual Studio 2008* project click the "*Generate*" button. If the generation process is successful, an appropriate message window will appear (Figure 13).

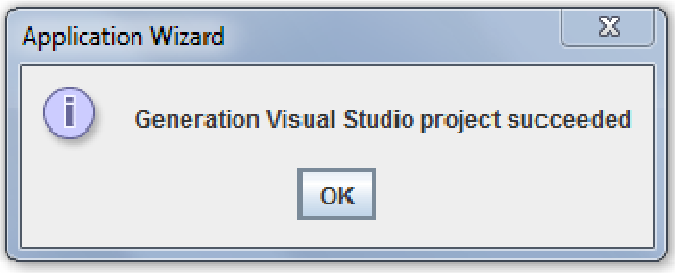

Figure 13. Message window - after successful *Microsoft Visual Studio 2008* project generation.

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

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