MŰEGYETEM 1782

Budapest University of Technology and Economics
Department of Measurement and Information Systems
Fault Tolerant Systems Research Group

Critical Architectures Laboratory
Spring Semester 2017/2018

Dependability modeling

Syllabus
v0.10

Authors: Attila Klenik
(klenik@mit.bme.hu)
András Vörös
(vori@mit.bme.hu)

February 17, 2018

Contents

1 Introduction **2**

2 Modeling formalisms **2**

3 Fault tree models **2**

3.1 Infrastructure 2

3.2 Fault tree construction 3

3.3 Dependability measures 4

4 Stochastic activity network models **5**

4.1 Atomic model composition 7

4.2 Performance variable definition 8

4.3 Study definition 9

4.4 Transformer Selection 9

4.5 Solver selection 11

5 List of Questions **11**

1 Introduction

Dependability modeling plays a more and more important role during the design of today's IT infrastructures. As the services perform more critical tasks nowadays, their unavailability causes more loss of profit for companies (even up to millions of dollars per hour). In order to plan for and predict these costs, we need to employ dependability modeling and use its solid mathematical foundation.

The goal of this syllabus is to prepare students for the dependability laboratory by providing a basic background for the needed modelling formalisms and tools, with the help of example exercises.

Before the laboratory, it is recommended to review the following related materials of the Design for Dependability and Formal Methods courses.

2 Modeling formalisms

In order to calculate dependability measures we can employ either:

- simulation,
- or analytic solutions.

The advantage of simulation is that we can use it with arbitrary models because of the lack of constraints (distributions, special behaviors) present in case of analytic solution. However, the results gathered during simulation are limited, and we have to take great care to ensure that we run the simulation for the appropriate amount of cases and time.

On the other hand, analytic solutions provide accurate answers (to a certain degree), but cannot be employed for every model due to its constraints, especially for some dynamic, behavioural models.

In the next sections we discuss the non-state space-based fault tree and state space-based stochastic activity network modeling (SAN) formalisms and demonstrate their usage through some basic examples.

3 Fault tree models

In this section we introduce fault tree-based modeling and its usage through the *TopEvent FTA Express 2017* tool.

3.1 Infrastructure

Figure 1 shows a simple infrastructure for a web-based service. We will construct a fault tree for this system.

The infrastructure consists of a web server cluster, a database cluster and a common disk subsystem. Both the web server and database layers have built-in redundancy to increase their availability, thus the system can tolerate a failed web server and database server.

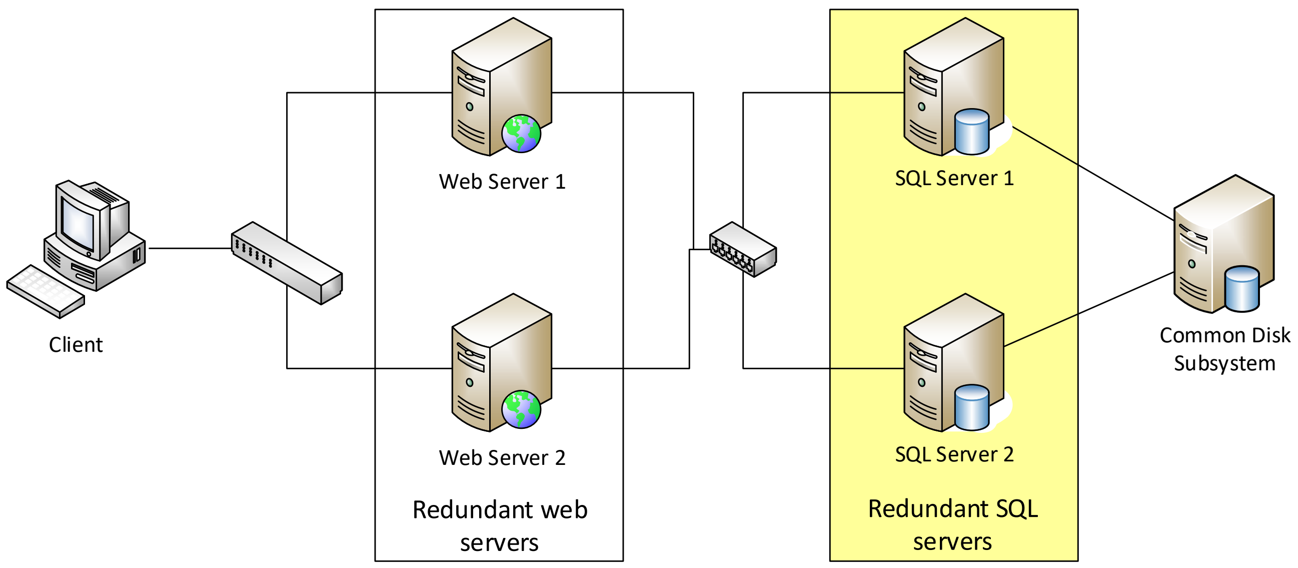


Figure 1: Service infrastructure

3.2 Fault tree construction

First we start with the top-level failure event, this is denoted as *Service Failure* in Figure 2. The second event level denotes subsystem failures and its nodes are connected to the top-level with an *OR* gate (meaning that *either* of those failures can cause service failure). On the last event level we see the atomic events, and their respective wirings to their subsystem-level failures (using *AND* gates to denote redundancy). We can see, that the common disk subsystem is a single point of failure (SPOF) in the system, since its failure on its own is enough for a service level failure.

The same fault tree in TopEvent FTA is shown in Figure 3. The tool is intuitive to use. Selecting a node in the tree, we can add child nodes to it by using the items in the *Add Input to Selected Item* group on the ribbon. We can also easily reorder the elements by using their context menus.

When a node is selected, a *Preferences* icon is displayed at its top-right corner. We can use this special context menu to set (or change) the name, description and child gate type of an intermediate event. Reconnecting sub-trees to other parents is not supported, so it is inconvenient to delete middle nodes from the tree as it also deletes the sub-tree.

In case of basic events, there is an additional tab in the preferences window denoting the failure probability model of the event. As shown in Figure 4, we can assign a named probability model for an event (this allows reusability of models between events) and set its parameters depending on the model type.

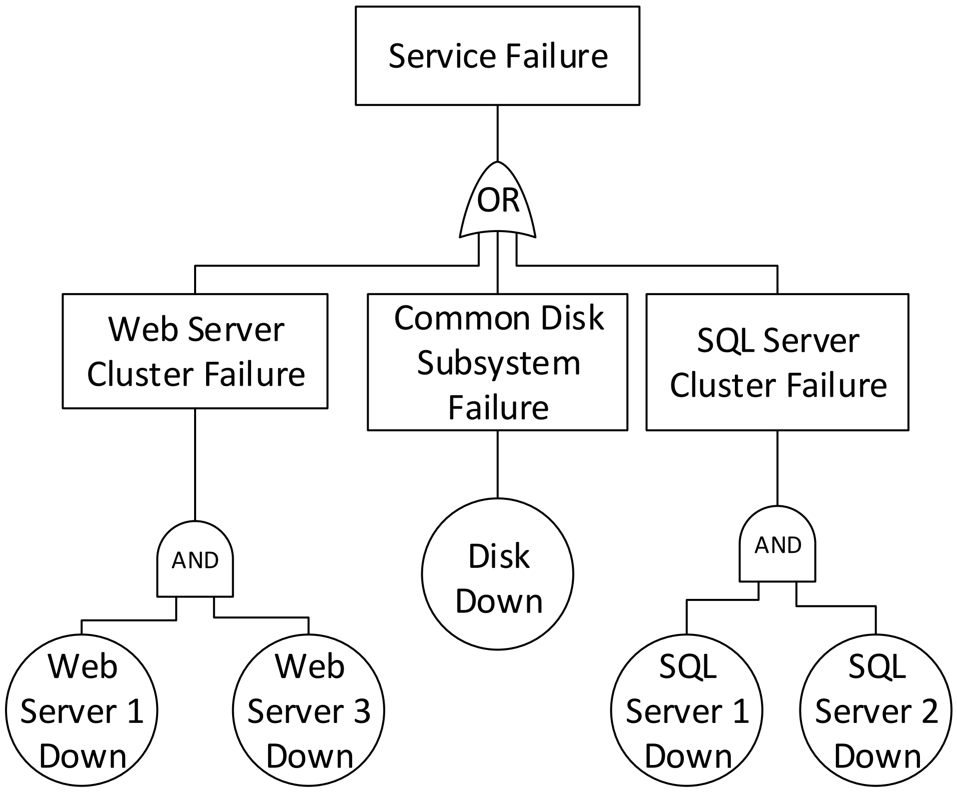


Figure 2: Service fault tree

3.3 Dependability measures

We characterize the reliability of a component with a failure rate taken from an exponential distribution with a parameter of λ. In this case the *expected* time of failure will be $1/\lambda$.

Let the failure rates be the following:

Component	Failure rate (λ) in days
Web server	0.05
SQL server	0.01
Common disk	0.2

Setting the probability models of each event to *Unrepairable* with the given failure rates (leaving the initial probability at 0) enables us to derive *reliability* measures of the system (since we cannot repair components). It is recommended to create named probability models for web server, database server and disk events and reuse them (as shown in Figure 5).

By clicking on the *Evaluate Fault Tree* item on the ribbon, a new evaluation item is added to the Project Explorer. Selecting that item will enable us perform the evaluation for a given mission time (with the default settings) by clicking again on (the different) Evaluate Fault Tree item on the ribbon.

As a result, we gain unavailability diagrams and *minimal cut sets*, i.e. combination of events that can cause system-level failure, for every complex event in the tree. We can export the results in MS Excel format to visualize more than one evaluation in the same graph.

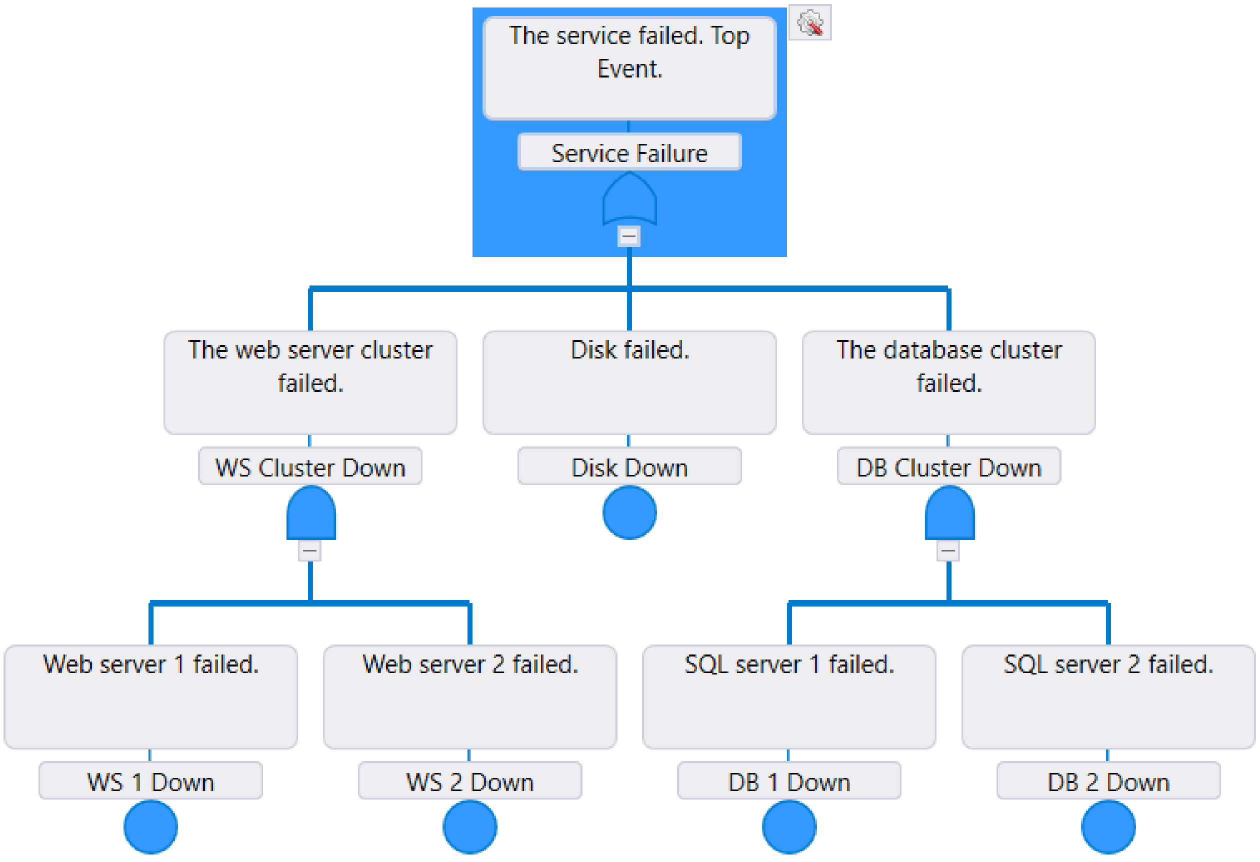


Figure 3: Fault tree in TopEvent FTA

4 Stochastic activity network models

In this section we introduce Möbius, an industrial tool for quantitative analysis of multiple modeling formalisms (and their combinations). Möbius uses a simple linear workflow with well-separated stages in order to provide flexible evaluation. These steps will be demonstrated with the help of a simple running example. The stages are the following:

Atomic model composition for defining basic models (using multiple formalisms) as building blocks for composite models.

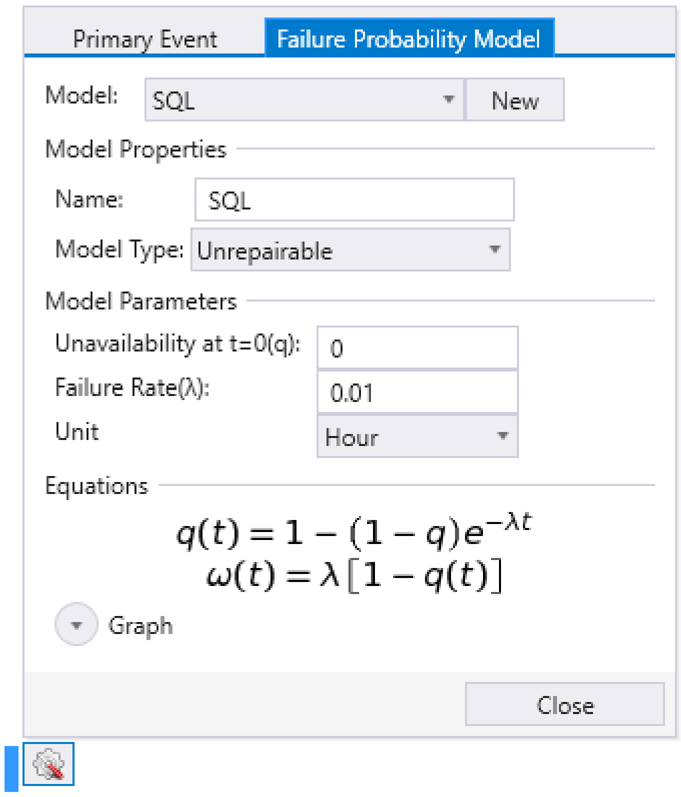
Composed model definition for building complex system out of atomic models. This step is optional, and not in the scope of this laboratory.

Performance variable definition in order to formulate different reward measures based on the current state and actions of the system.

Study definition for evaluation the defined rewards for multiple values of model parameters.

Transformer selection for generating the state space of the model with its current parameter values (defined by the current study).

Solver selection for actually calculating the rewards based on the probabilistic behavior (state space) of the system.



Model: SQL

Model Properties

Name: SQL

Model Type: Unrepairable

Model Parameters

Unavailability at t=0(q): 0

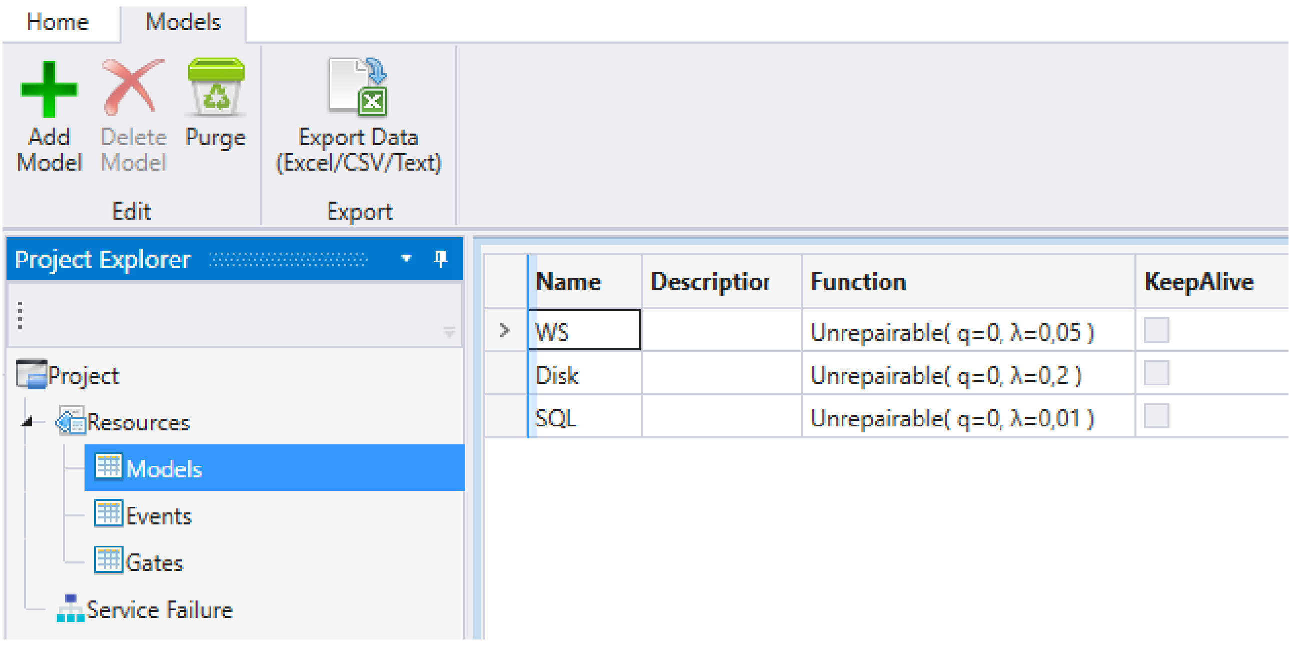
Failure Rate(λ): 0.01

Unit: Hour

Equations

$$q(t) = 1 - (1-q)e^{-\lambda t}$$
$$\omega(t) = \lambda\left[1 - q(t)\right]$$

Figure 4: Event probability settings



Name	Descriptior	Function	KeepAlive
WS		Unrepairable(q=0, λ=0,05)	☐
Disk		Unrepairable(q=0, λ=0,2)	☐
SQL		Unrepairable(q=0, λ=0,01)	☐

Figure 5: Different probability models

Be careful not to use spaces (or any whitespaces) and special characters throughout the project, since most of the names in the model will be translated to native code variables! This can cause some shady errors during analysis.

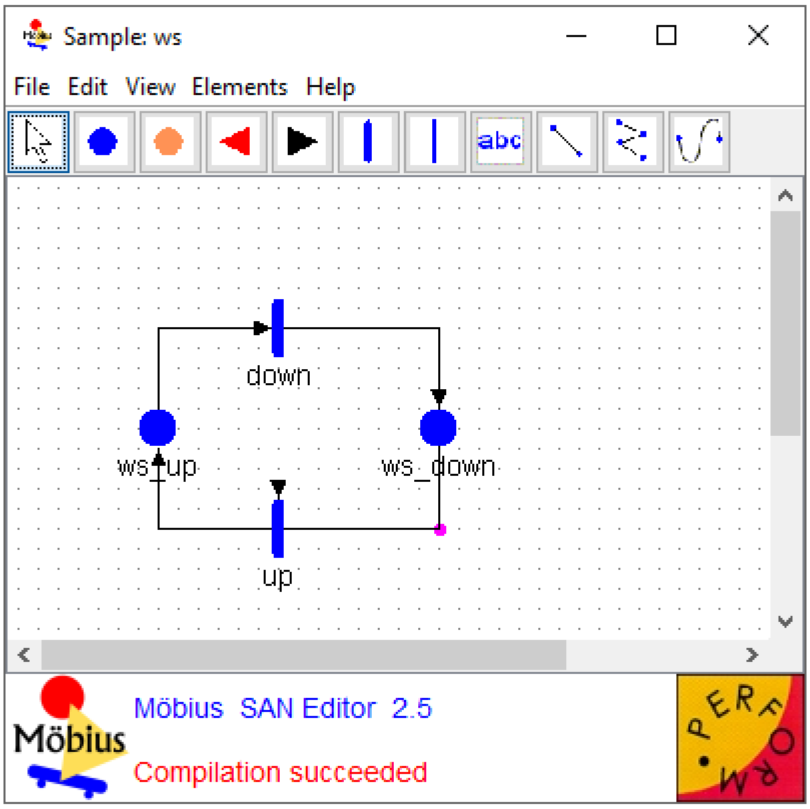
Detailed tutorials regarding Möbius can be found on its wiki page (much of which is out of scope for this laboratory, but good for reference). It is highly recommended to review the

SAN section.

We will use only a subset of the SAN functionality Möbius provides, which will make them equivalent to stochastic Petri nets.

4.1 Atomic model composition

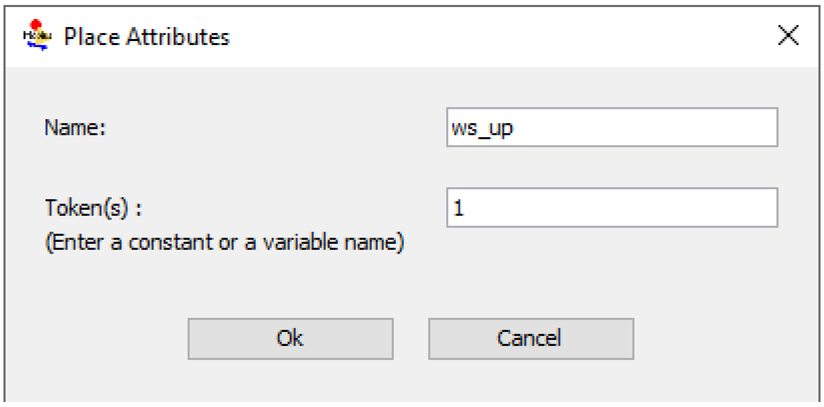
First, let's add a new atomic SAN model, and construct a model of a web server that can be either in up (good) state, or down state, as depicted in Figure 6.



Compilation succeeded

Figure 6: Example SAN for the web server

When adding places, we also have to give them a name, and an initial marking, as seen in Figure 7. Place *ws_up* will have a marking of 1, the other place has 0.

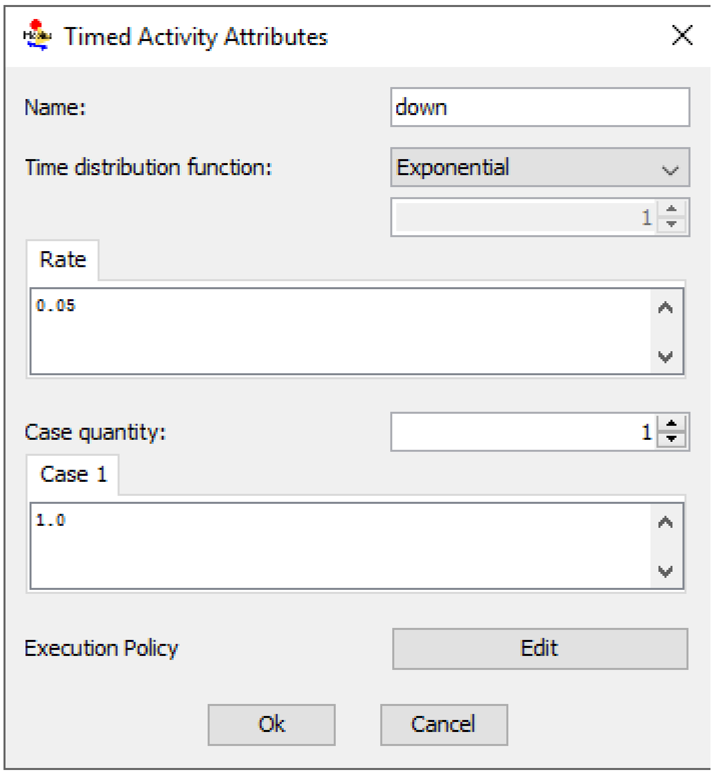


Name: ws_up

Token(s) : 1
(Enter a constant or a variable name)

Figure 7: Place settings

Adding transitions requires a little more configuration beside naming it. We also have to provide a firing delay with its distribution and parameter (rate), as demonstrated in Figure 8. The rate can be either a C++ expression or a complex statement containing explicit return statements. In this case it is a simple expression of the form 0.05. For the description of case quantity, please refer to the Möbius SAN wiki.


Timed Activity Attributes

Name: down

Time distribution function: Exponential

1

Rate

```
0.05
```

Case quantity: 1

Case 1

```
1.0
```

Execution Policy

Figure 8: Transition settings

4.2 Performance variable definition

Once we successfully saved (and compiled) our SAN model, lets add a new *Reward* definitions to the project. The definition will build on the previous SAN model as *component child*.

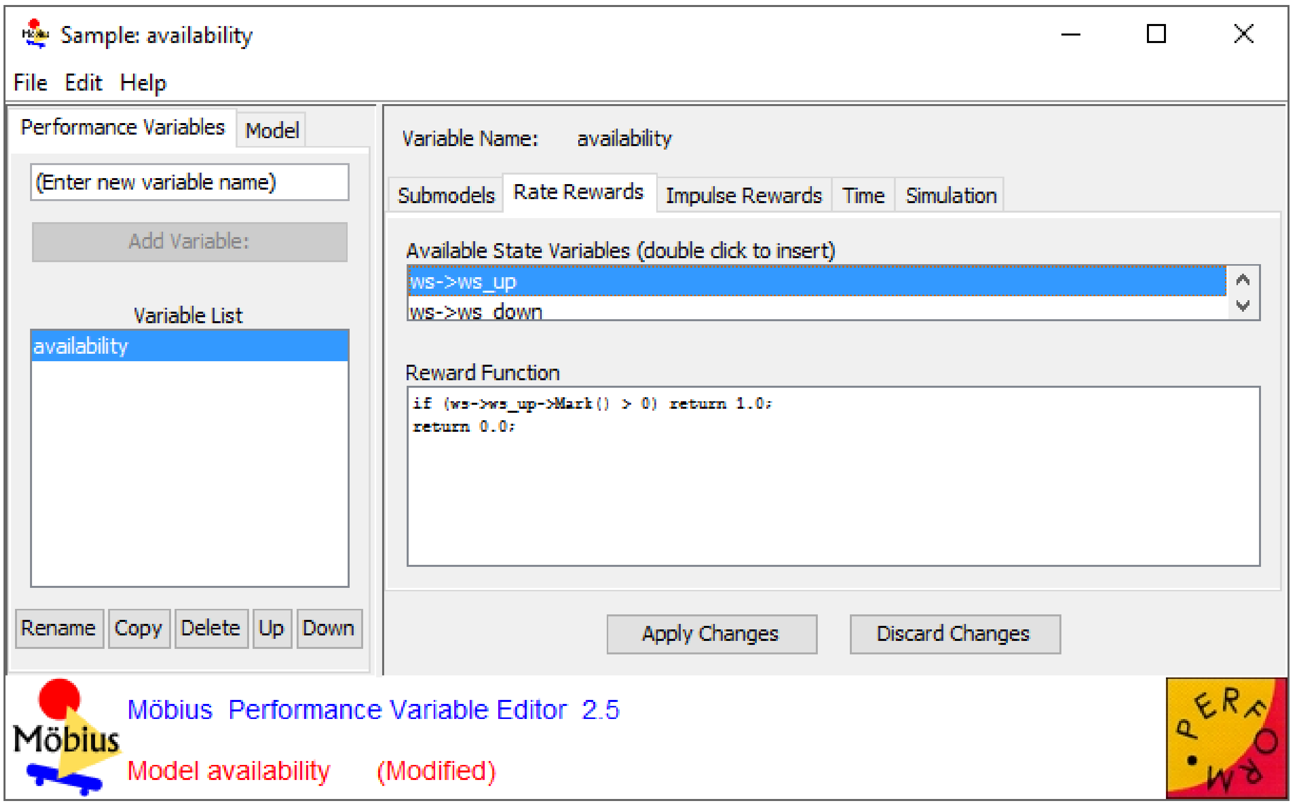
In the new window, we can define multiple reward variables, so let's create one with the name *availability*. On the right side we can configure the currently selected reward variable.

The availability can be calculated based on the states of the model, so this variable will be a *Rate Reward*, with a reward function that will be evaluated for every state and weighted with the probability of being in that state in a given time.

In order to reference the current marking of a place, we can use the *model->place->*Mark() notation. Since we want to "count" the probability of being in a functional state, our reward function will be:

```
if (ws->ws_up->Mark() > 0) return 1.0;
return 0.0;
```

where *ws* is the name of our SAN model, and *ws_up* is the name of the place denoting an operational state (see Figure 9).



Variable Name: availability

```
if (ws->ws_up->Mark() > 0) return 1.0;
return 0.0;
```

Figure 9: Reward function definition

Beside the actual definition, we also need to state the logical time at which we want to evaluate the reward variable. This can be done on the *Time* tab of the same window. We want to evaluate the variable multiple time points, so we set the time points accordingly, as seen in Figure 10.

Make sure not to set 0.0 as an actual time point, as it may cause the solver to fail without generating an error message.

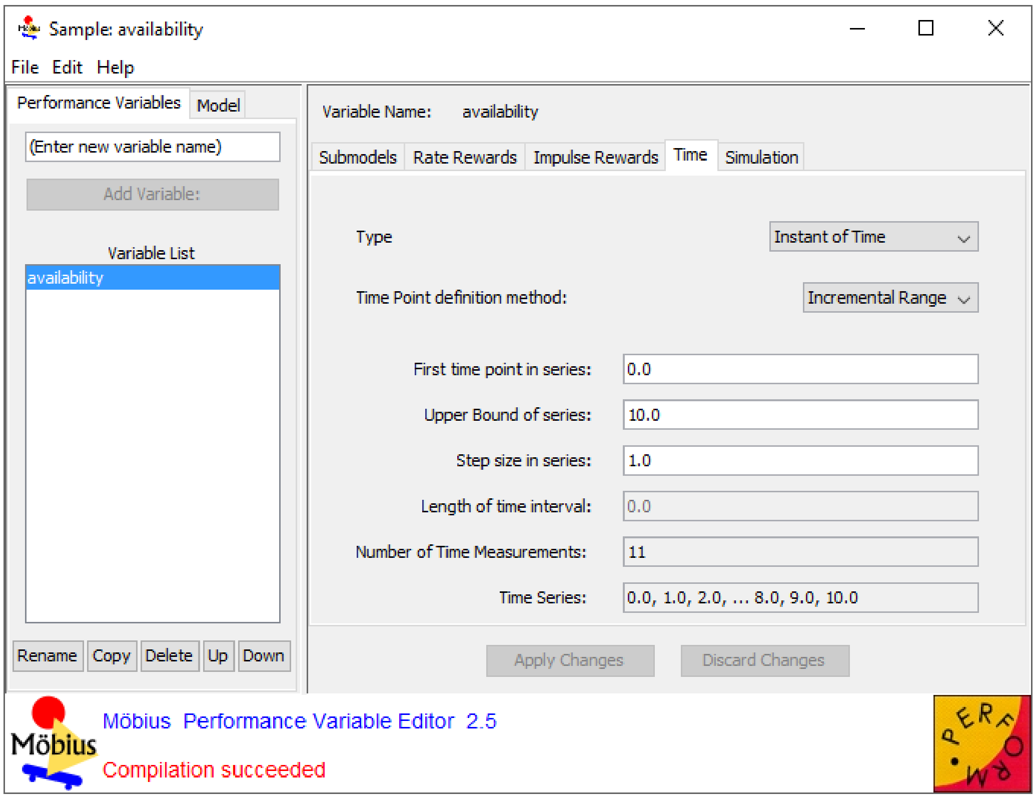
4.3 Study definition

This step allows us to evaluate previously defined rewards for multiple value combination of model parameters. Since we do not have parameters, we simply create a new study and leave it as it is (unfortunately we need this *"dummy"* study to continue).

4.4 Transformer Selection

The next step consists of defining the state space traversal method for a given study. The symbolic method is recommended for bigger state spaces. During this laboratory, the selected method will not make a difference. Although perhaps the explicit method is better suited for debugging.

If we set the *Build Type* to *Normal*, we can enable tracing during the algorithm execution. We can start the state space generation by clicking *Start State Space Generation*. This is not required, as the solver in the next step will do this for us, but it is useful for validation purposes.



Variable List: availability

Variable Name: availability

Type: Instant of Time

Time Point definition method: Incremental Range

First time point in series: 0.0

Upper Bound of series: 10.0

Step size in series: 1.0

Length of time interval: 0.0

Number of Time Measurements: 11

Time Series: 0.0, 1.0, 2.0, ... 8.0, 9.0, 10.0

Compilation succeeded

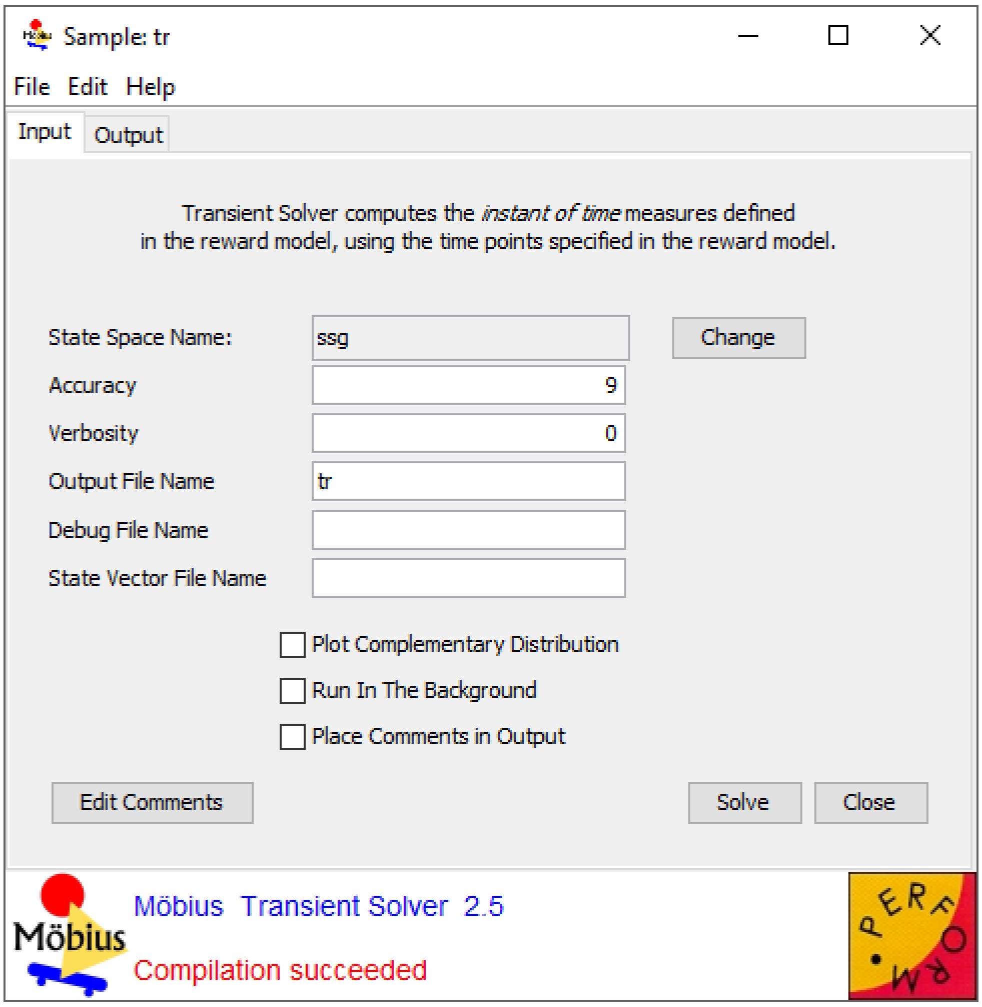
Figure 10: Time definition

Upon successful generation, we should see an output something like the following:

```
........................................................
Updating modules... Done
Building State Space Generator for win32 architecture
Building for win32 systems on KLENIK-PC
Compiling win32 ...
ws
availability
simple
Done

*******************************************************

State Generation initiated on Experiment_1 of ssg started at Tue
   Mar 28 15:50:19 CEST 2017 ...
path=[c:/MobiusProject/Sample/Transformer/ssg/]
License File=C:\Users\aklen\AppData\Local\Temp\
   mobius1974288483002388957license
There are no global variables.
SSG: SetTraceLevel: 0

Generated: 2 states
Computation Time (user + system): 8.000000e-003 seconds
State Generation of Experiment_1 on model ssg finished at Tue Mar
    28 15:50:20 CEST 2017.
```

4.5 Solver selection

As the last step we select the numerical solver we would like to use to evaluate the reward variables. Since we want to evaluate it at specific time points, we add a Transient solver to the project (Figure 11). For the description of available solvers, refer to the corresponding wiki page. We can provide the accuracy for the solver as the exponent of the $1E - [exponent]$ expression. After clicking *Solve*, the results will be available in CSV format in the specified file, under the *$ProjectPath\Solver\SolverName* directory.



Transient Solver computes the *instant of time* measures defined
in the reward model, using the time points specified in the reward model.

State Space Name:	ssg
Accuracy	9
Verbosity	0
Output File Name	tr
Debug File Name	
State Vector File Name	

☐ Plot Complementary Distribution
☐ Run In The Background
☐ Place Comments in Output

Möbius Transient Solver 2.5
Compilation succeeded

Figure 11: Configuring the transient solver

5 List of Questions

This list of questions aims to help to prepare for the entry test of the laboratory:

1. Create a Petri net model for an infrastructure containing two redundant SQL servers and a single web server!
2. Create a fault tree model for an infrastructure containing two redundant SQL servers and a single web server, that depicts the ”Service failed” top-level event!

3. What does *minimal cut sets* mean in fault tree modeling?
4. What approximation/simplification can we use to evaluate an AND gate probability based on its child event probabilities during fault tree analysis?
5. What approximation/simplification can we use to evaluate an OR gate probability based on its child event probabilities during fault tree analysis?
6. What are the main steps Möbius uses to construct and evaluate models?