CoSMoS Help

CoSMoS 2.0.0
Help Document / Guide

Hessam S. Sarjoughian
Vignesh Elamvazhuthi

Arizona Center for Integrative Modeling & Simulation

2002-2009

CoSMoS Introduction

CoSMoS (Component-based System Modeling and Simulation**) is a new integrated modeling and simulation environment. Its modeling engine supports logical, visual, and persistent model specification with support for automated simulation code generation. Its simulation engine supports visual experimentation configuration and run-time data collection and observation. The CoSMoS tool enables simulation-based system design process with support for model verification and simulation validation. The integrated model specification, simulation code generation, and controlled experimentation capabilities of the CoSMoS tool are demonstrated with a model of an Anti-Virus Network software system.**

CoSMoS supports simulation-based design and analysis of complex systems using well-defined precise structural and behavioral abstractions. The modeling and simulation processes are shown in the figure below. A synopsis of the steps in creating simulation models is described below.

Select Database**: This process defines the user selecting the database that serves as a repository for the models. The relational database supports functionalities like creation, modification, storage, and reuse of the stored models. Structured Query Language (SQL) is used as a medium of communication as it is a standard language for databases and helps in application portability. The user is required to locate the database and create an appropriate data source for it using Microsoft Access (*.mdb) as the driver.**

Select the existing template model or create new**: CoSMoS allows reuse of the models since models are stored in the database. The user can also**

create new, unique template models to represent a new family of models. The template model defines the primitive or composite model with input or output ports and values. The atomic model contains state variables, the ports, and the name of the model. The coupled model specifies the couplings between its components and the name of the ports. The name assigned to the primitive or the composite model must be unique, i.e., it must be identifiable within its hierarchical decomposition.

Transform Instance Models**: The template models created are instantiated to a well defined model when they are transformed into Instance Models. If the model has specialized models, the user can select the specialization for these models during the transformation. The modeler can specify different models depending upon his choice during the instantiation of template models. This gives the modeler the independence to create alternative models depending on alternative resolution and aspects.**

Partial DEVS models created**: The translator in CoSMoS can export the logical models into simulation code that conforms to the syntax of the DEVS-Suite simulation engine. The behaviors of the primitive models are defined in terms of dynamic characteristics of the model, such as input variables, output variables, state variables, and state transition functions.**

Manually Add behavior to the simulation models**: The primitive models are completed using the IDE in the CoSMoS environment. The models are completed by adding the behavior and completing the transition functions.**

Select and load simulation models**: The visual model in CoSMo is selected to determine the model to be simulated. The models are mapped to their files that are simulation code written in JAVA. These models are complete and are compiled before the model class files are ready for simulation. It is an iterative process between** Completed and compiled Java implementation files **and** Select and load simulation models.

Visually Select components and ports of models**: The ports of the primitive and composite models can be selected visually. These selections by the user are stored in the memory (**JVM**) and are used by the** Tracking Control **in DEVS-Suite for simulating the models.**

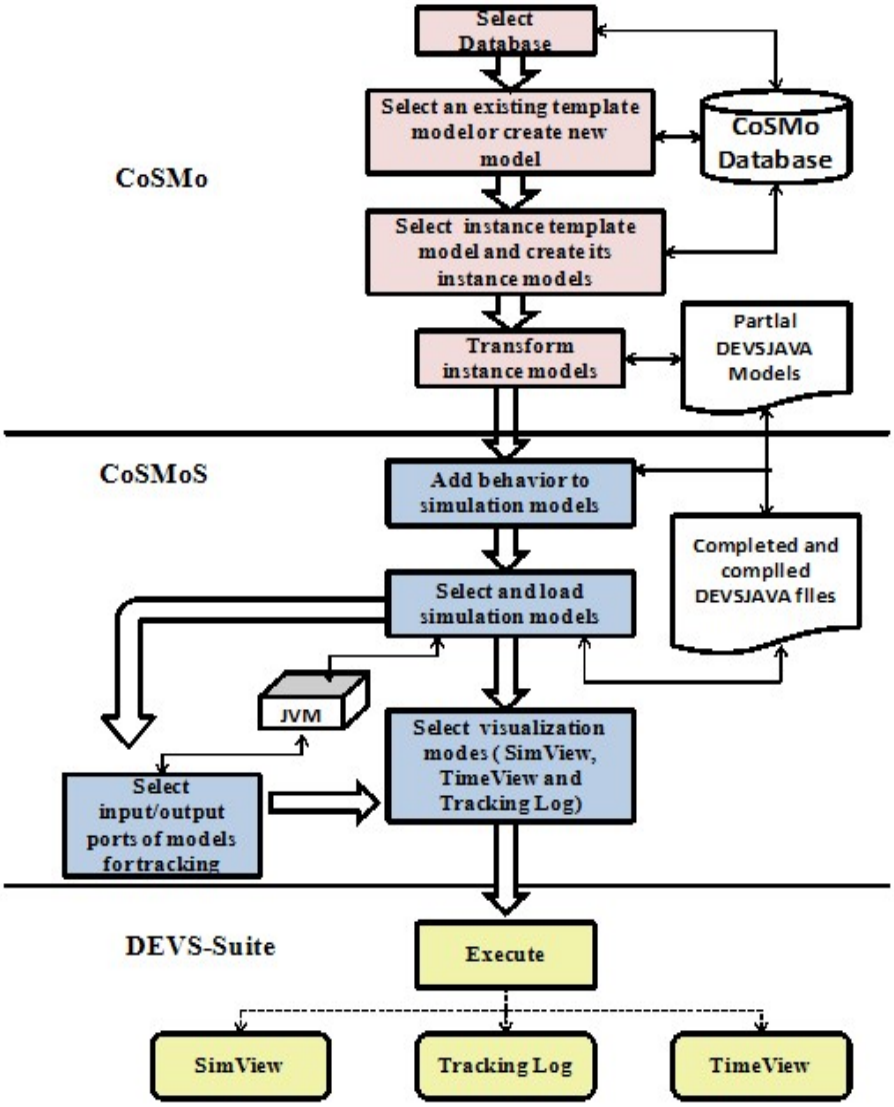


Figure 1 CoSMoS Process

A snapshot of the CoSMoS tool is shown in the following figure. The NetVirus example model is created visually and from it partial simulation code is automatically generated.

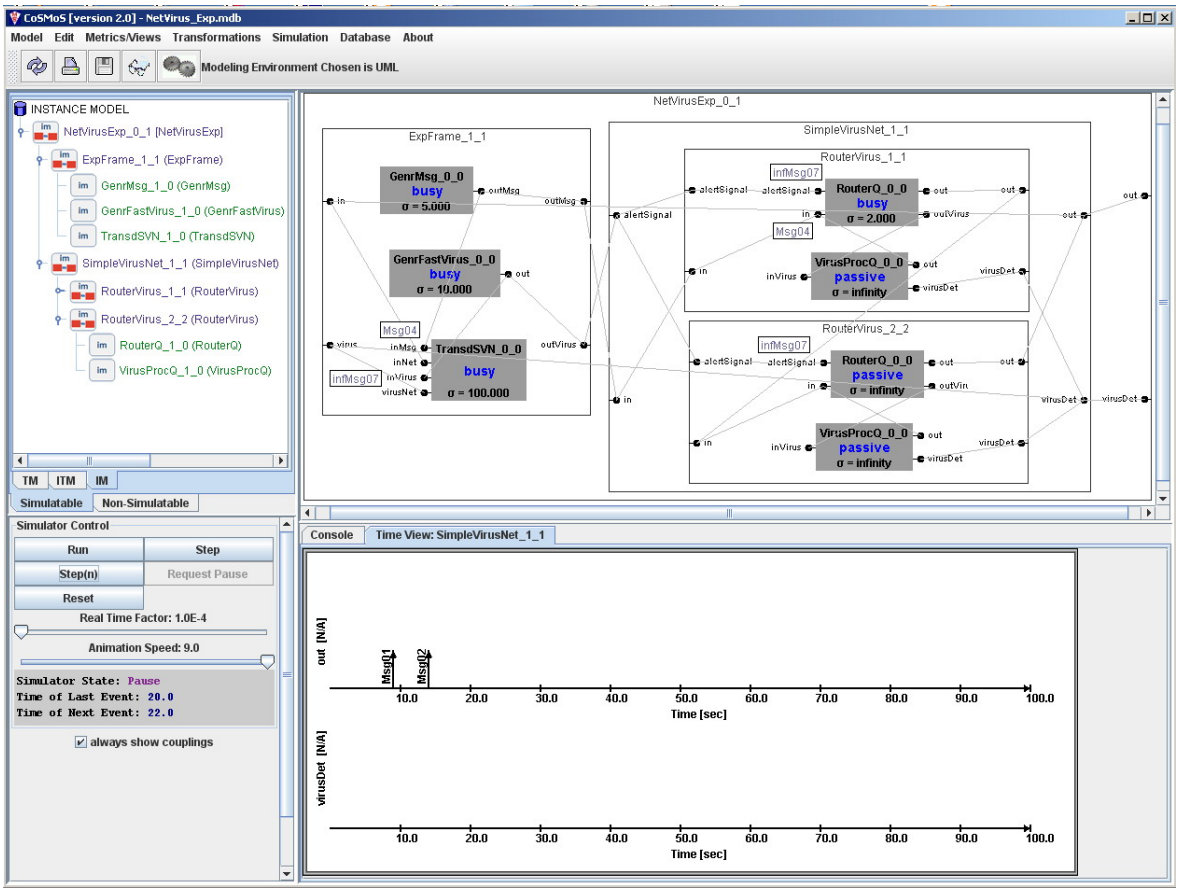


Figure 2 ODBC DEVS-Suite Environment

The partial simulation code is completed which is necessary to be executed using the DEVS-Suite simulator as shown below.

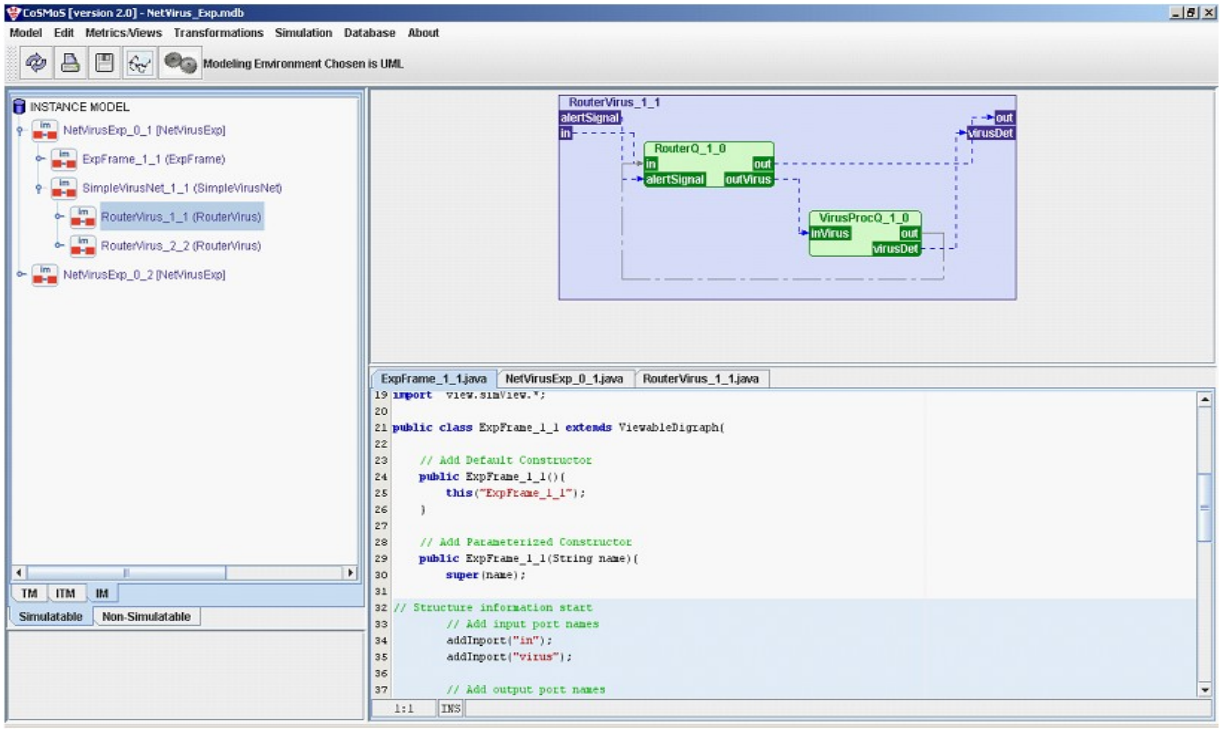


Figure 3 Source Generated Source Code for DEVS-Suite

Arizona Center for Integrative Modeling and Simulation
This help file has been generated by the freeware version of HelpNDoc

DATABASE CONNECTIVITY

DATABASE CONNECTIVITY

CoSMoS stores the models that it creates in MS Access databases. The databases and the structure of the models are created automatically which will be discussed later. Once the databases has been created, the database has to be added as a data source before it can be used by the modeler for storing models

Arizona Center for Integrative Modeling and Simulation

Creating Data Sources

CREATING DATABASE MODELS

The data sources can be created using the ODBC Data Source Administrator. The steps for creating the data source are as under

1. **Open the Control Panel from the Settings menu in the Start Menu**

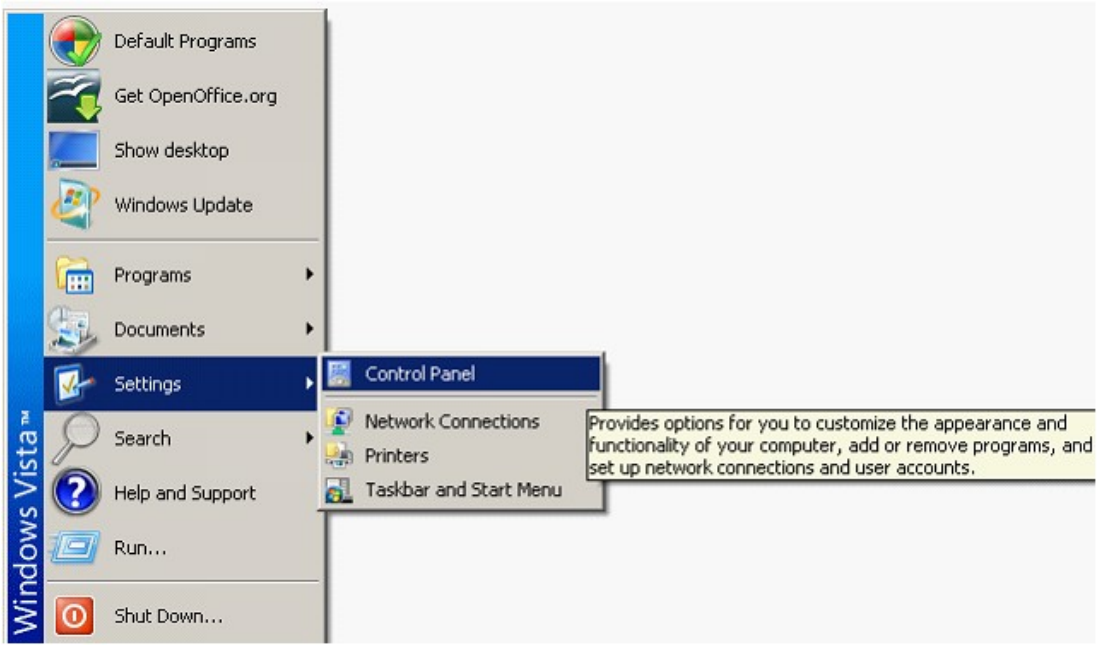


Figure 1 Control Panel

2. **Select Administrative Tools -> Data Sources (ODBC)**
3. **Now a window should popup with title “ODBC Data Source Administrator” as shown in Figure 2.**

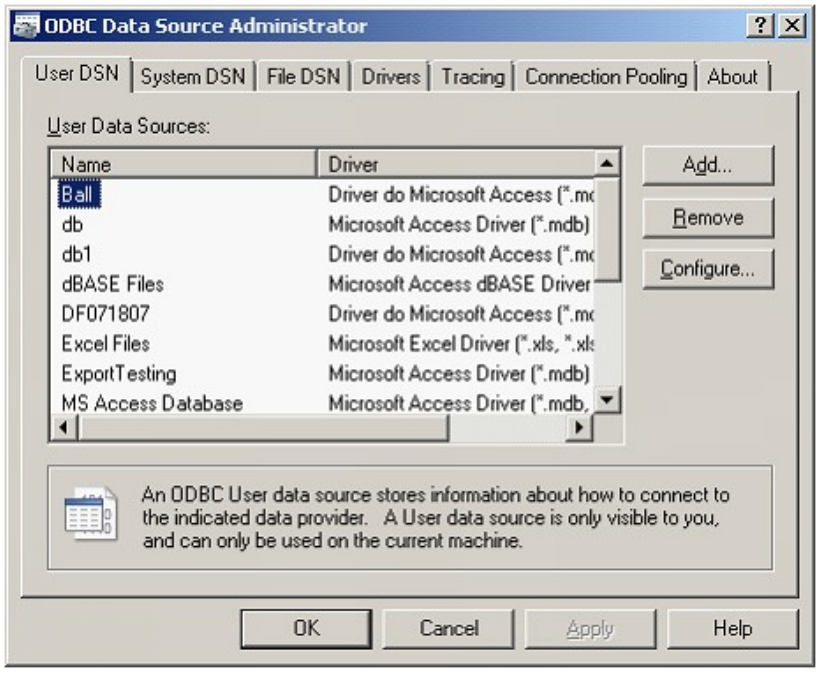


Figure 2 ODBC Data Source Administrator

4. **Click the “Add” button on the right end of the window to open up a panel giving you options to select the driver for the database. Select the option Microsoft Access Driver (*.mdb) and select “Finish” to proceed.**

5. **Now a pane (as shown in Figure 3) shows up with options to select the database, assign a name and some description. Click “Select” button on the center of the screen to popup a dialog box to select the database.**

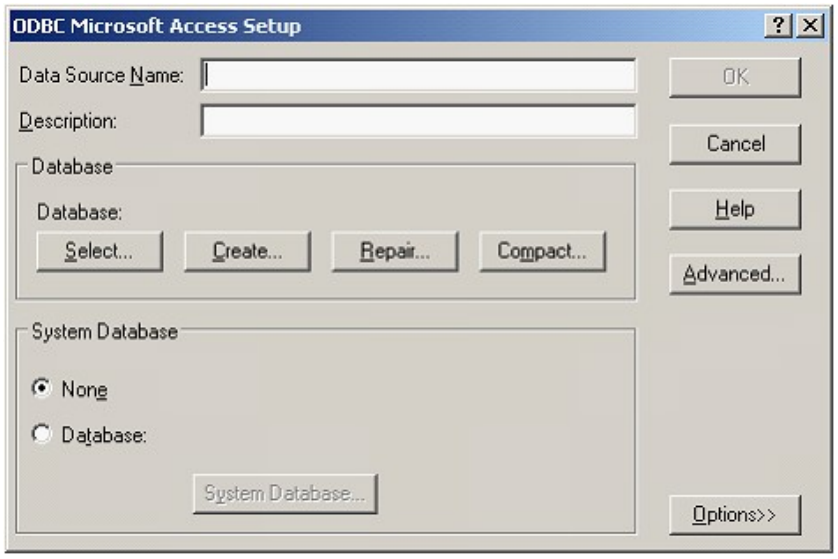


Figure 3 ODBC Microsoft Access Setup

6. **Select the newly created database by giving the path to it usually present at “root/MB_Models/Name of the database/Database/Name of the database.mdb”. Select OK to accept the database.**

7. **Now in the Figure 3, the field Data Source Name should be given the same name as the name of the database selected, excluding the extension. Description can be given anything to describe the database.**

8. **Select OK on both the panels to finish the process.**

CoSMoS

Arizona Center for Integrative Modeling and Simulation

CoSMoS DATABASE

CoSMoS DATABASE

Arizona Center for Integrative Modeling and Simulation

This help file has been generated by the freeware version of HelpNDoc

CREATE A DATABASE

CREATE A DATABASE

On running CoSMoS the user is shown a form to specify the username and password select the mode of operation (i.e. UML or XML) and then select the database for the operation. If this is the first time for the user to run CoSMoS on the system and the user doesn't have a database created for him. The user can type the name of the database and the system creates a new database with the required structure. The user needs to create the data source for it by following the steps mentioned in Create Data Source Section.

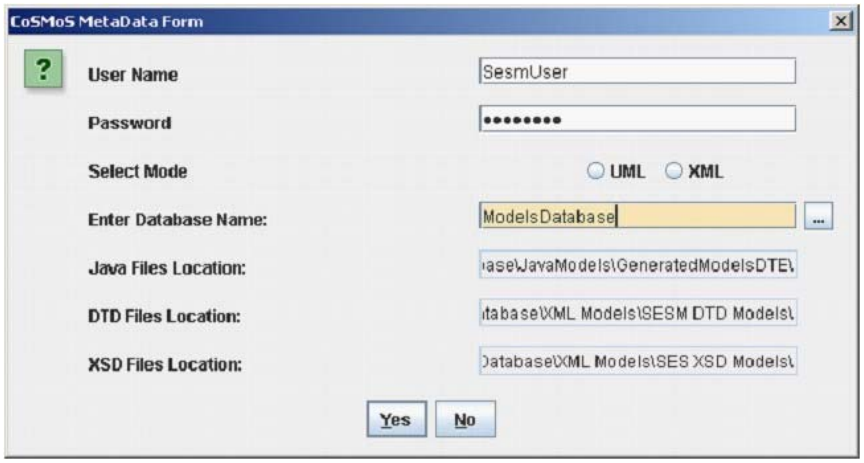


Figure 1 CoSMoS Meta Data Form

As the user mentions the name of the database, a set of folders gets created in the location "root\ MB_Models\ NewDatabase" if the name of the database is "NewDatabase". Select "Yes" to finish the process.

Arizona Center for Integrative Modeling and Simulation

This help file has been generated by the freeware version of HelpNDoc

Select an exisiting database

SELECT AN EXISTING DATABASE

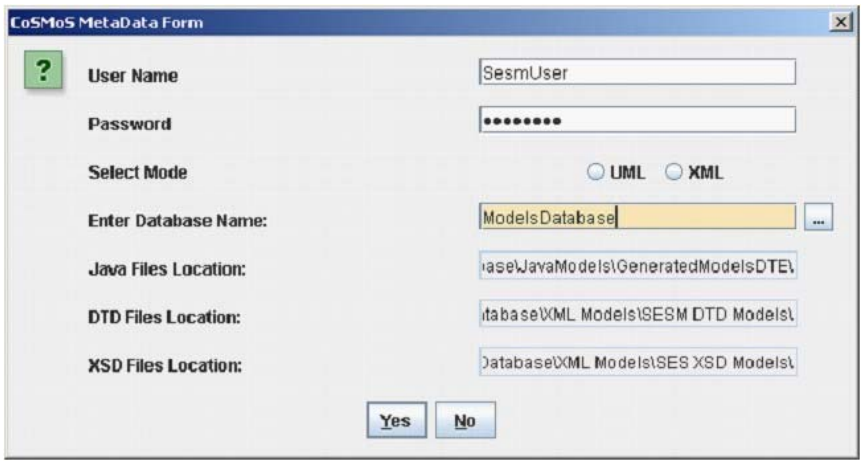


Figure 1 CoSMoS Meta Data Form

SELECT AN EXISTING DATABASE

1. **If the user has a database already defined for CoSMoS and needs to work on it, the ellipses button next to the name field of the database has to be clicked.**
2. **A Small window with the options of selecting the available databases shows up.**
3. **Select the database and proceed further.**

SELECT NON-SIMULTABLE MODEL TYPE

The modeler can choose either UML or XML.

Arizona Center for Integrative Modeling and Simulation
This help file has been generated by the freeware version of HelpNDoc

Template Model

Template Models

Arizona Center for Integrative Modeling and Simulation

This help file has been generated by the freeware version of HelpNDoc

Create Template Models

PRIMITIVE

The model can be created in two ways

Using the menu

1. **Select the Model Menu.**
2. **Select** Create Model Template **as shown in Figure 1. This command can also be executed by a Keyboard shortcut ALT+T.**
3. **Enter the name of the model in the popped up dialog box. (Figure 2).**

4. **Select the model subtype for the model to be added.**
5. **A window is popped up showing if the model was created successful or not.**

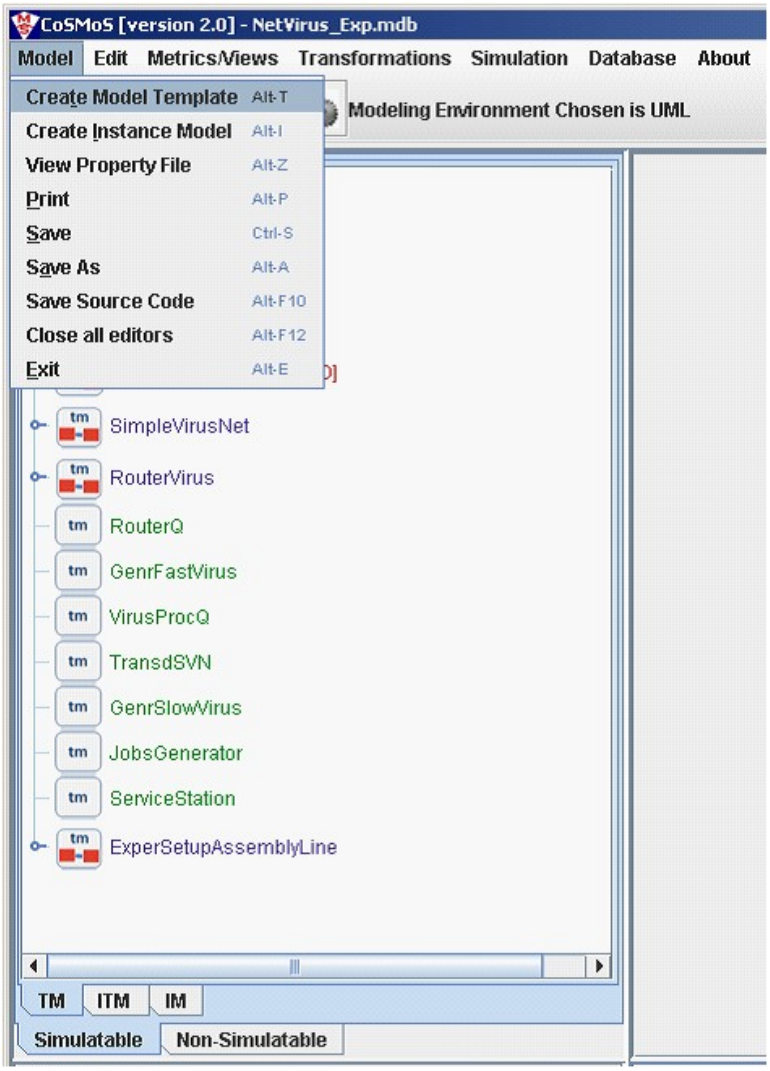
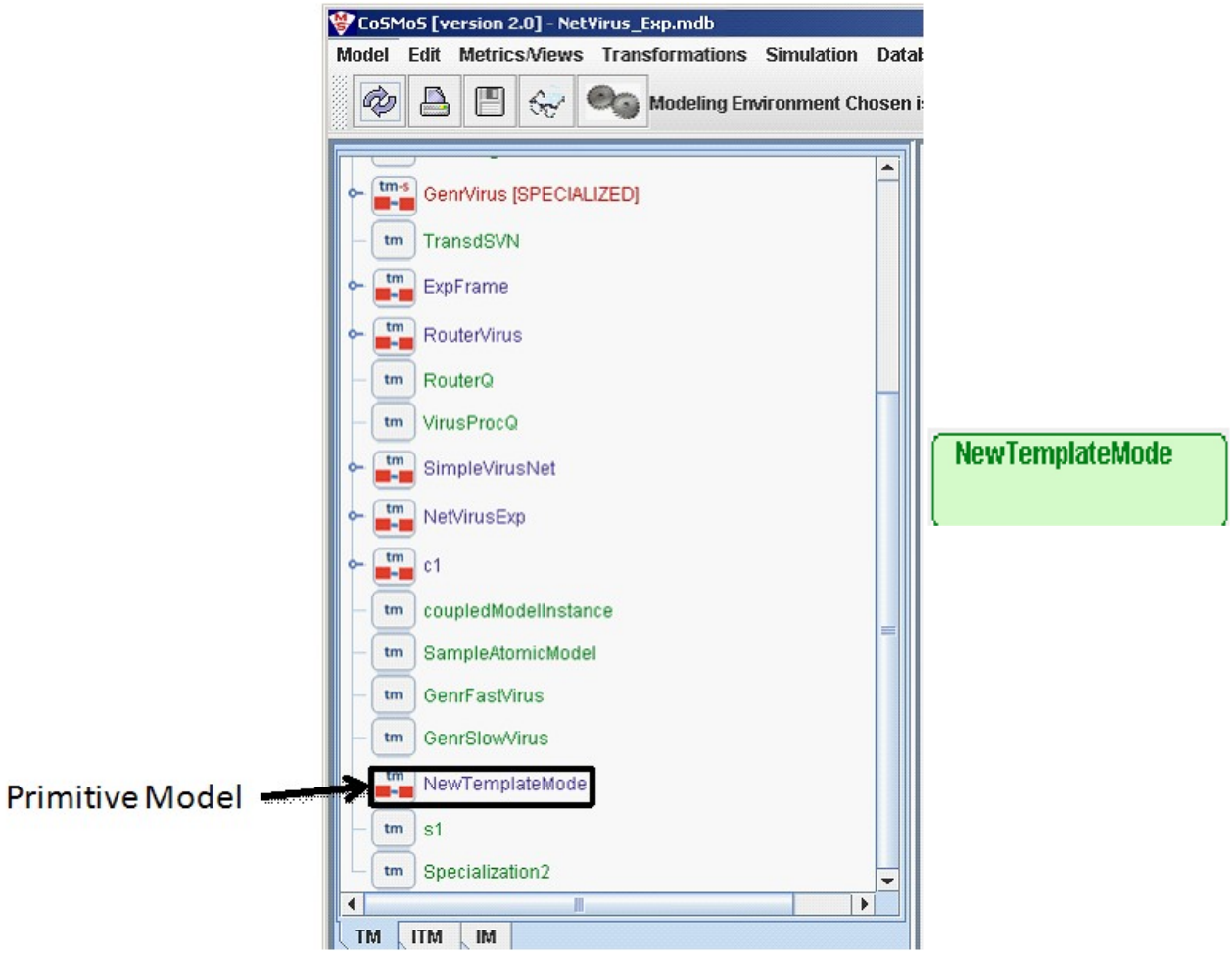


Figure 1 Menu Selection

Figure 2 Enter Name of the Model

Once the primitive model has been created, a new node is added to the tree on the left hand side. On clicking the node a rectangle shape of the model is shown in the right hand side in the model-view pane.



(a) Tree View (b) Block Model View

Figure 3 Template Model Tree and Block Views

Composite Model

The models are always created at first a primitive model. They can be converted into composite models by adding other primitive and composite components into it. There are three ways to add components into an existing model.

Using the Menu Bar

1. **Select the model on the tree view.**
2. **Select the menu Edit -> Add Component.**
3. **A dialog box (Figure 4) shows up, it allows the user to select the component to be added.**

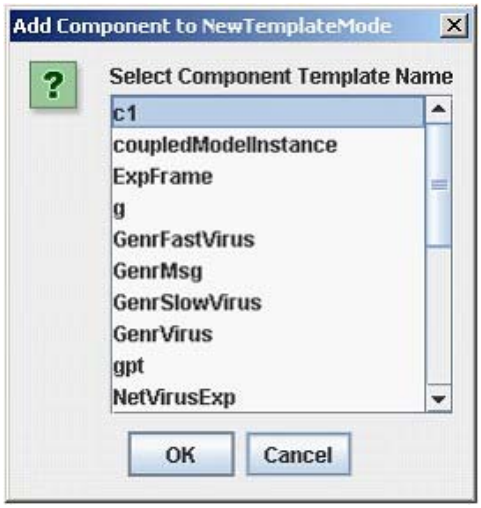


Figure 4 Add Component to Template Model

4. **After the component is selected, the user is asked to enter the multiplicity of the components in the model being added.**

Now the model is a composite model. This changes the block representation and the icon of the model in the tree and as shown in Figures 5 and 6.

Figure 5 Composite Model (Block Model View)

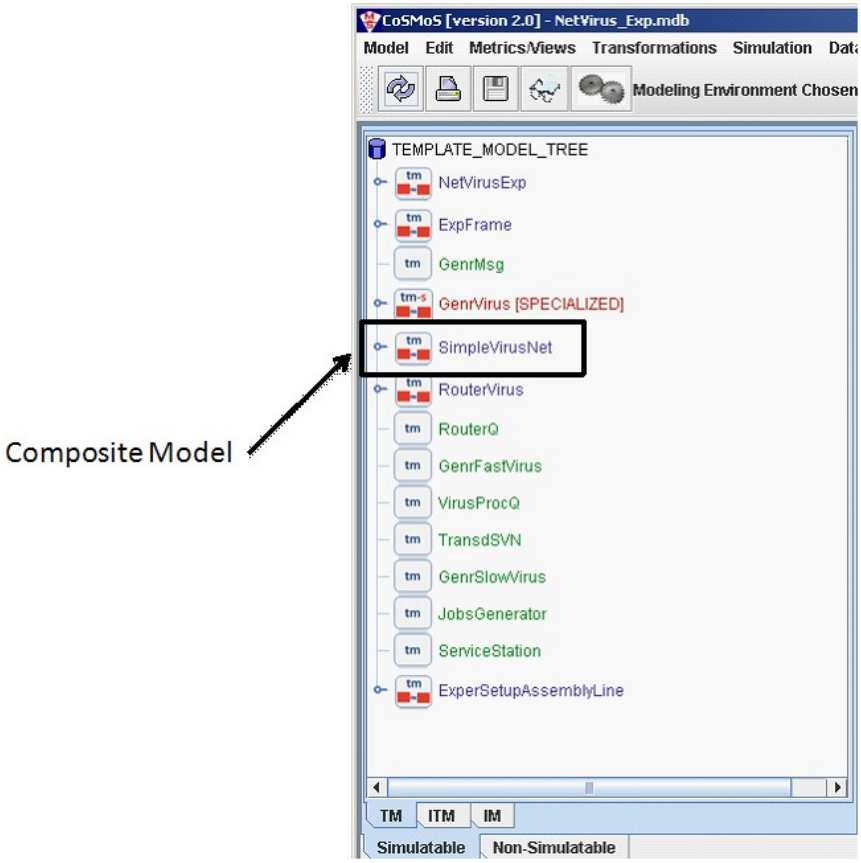


Figure 6 Composite Model (Tree View)

Using Right Click popup menu

Right clicking on the model in tree or right clicking in the model's graphical area brings up a popup menu. The option Model -> Add-> Component allows the user to add components.

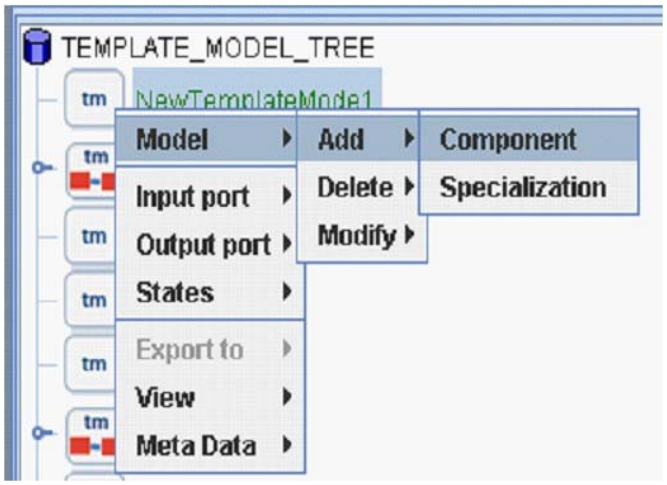


Figure 7 Right Click Menu to add components

Specialized Models

Specializations can be added to models using the popup menu that is shown in the Figure 7. The steps involved in the adding specialization for a model are as under.

1. **Select the option of adding specialization.**
2. **Give a name for the added specialization.**
3. **The icon of the model should change in the tree view and the block model layout should also have a new color.**

Figure 8 Specialized Model (Block Model View)

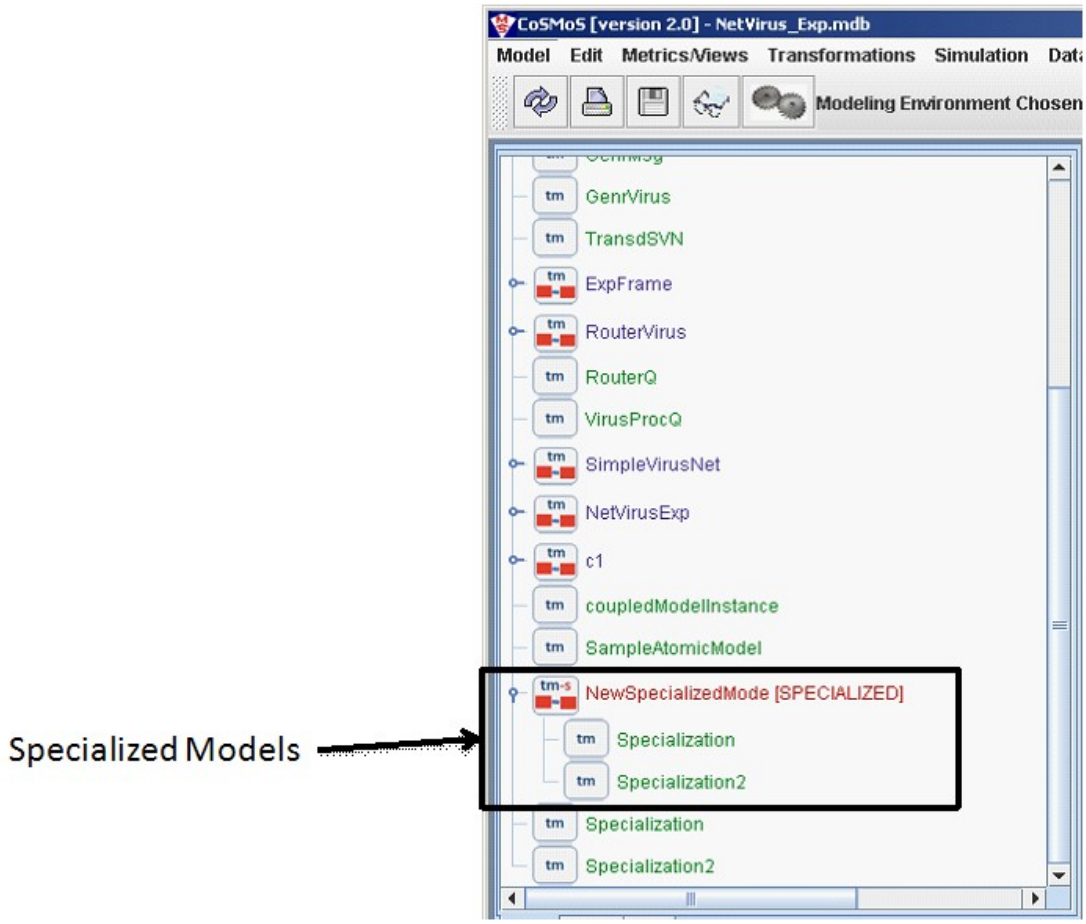


Figure 9 Specialized Model (Tree View)

Delete Models

DELETE MODELS

There are two forms of delete associated with the template models.

Deleting Models

1. **Right click on the model in tree or block model view.**
2. **Select Model -> Delete -> Delete Model.**

The delete model operation can be executed using the keyboard short cut ALT+DELETE.

Deleting Components

1. **Right click on the model in tree or block model view.**
2. **Select Model -> Delete -> Delete Component.**
3. **A window with the information of all the components in the model pops up. Select the component and proceed.**

The delete component operation can be executed using the keyboard short CTRL+DELETE.

Modify Models

MODIFY MODELS

CoSMoS allows the modeler to modify the following items on the model.

Rename the model

1. **Right click on the model in tree or block model view.**
2. **Select Model -> Modify -> Rename Model.**
3. **Enter the new name and proceed.**

The rename of model be executed using the keyboard shortcut ALT+N

Rename the components

1. **Right click on the model in tree or block model view.**
2. **Select Model -> Modify -> Component.**
3. **In the dialog box shown select the component and enter new name in the text field.**

The components in model can be renamed using the keyboard shortcut CTRL+N

Replace sub-type

1. **Right click on the model in tree or block model view.**
2. **Select Model -> Modify -> Replace Subtype.**
3. **In the dialog box shown select the select the subtype from the drop-down list.**

Input/Output Ports

Arizona Center for Integrative Modeling and Simulation

Create Input/Output ports

CREATE INPUT/OUTPUT PORTS

The input/output ports can be added to the models using two methods

Adding ports using menu bar

1. **Select the model on the tree or in the block view of the model.**
2. **Select items from menu as Edit -> Add Input Port for input ports and Edit -> Add Output Port for output ports.**
3. **Give a name for the port.**

The keyboard shortcut for the adding the ports are ALT+F1 for input ports and ALT+F3 for the output ports.

Adding ports by right click on models

1. **Select the model on the tree or in the block view of the model.**
2. **Right click and select input/output port -> Add -> Port.**
3. **Give a name for the port.**

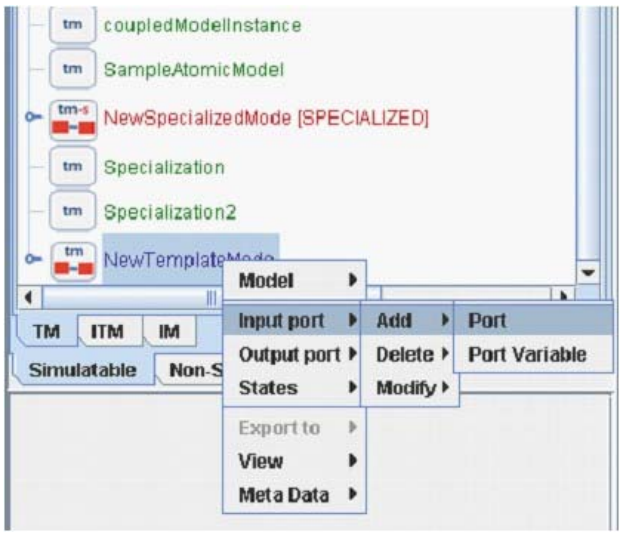


Figure 1 Add input ports

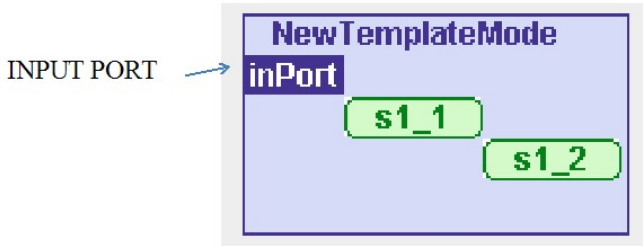


Figure 2 Input port added to a composite model

Delete Input/Output ports

DELETING INPUT/OUTPUT PORTS

The input/output ports can be deleted from the models using three methods

1. **Select the model on the tree or in the block view of the model.**
2. **Select items from menu as Edit -> Delete Input Port for input ports and Edit -> Delete Output Port for output ports.**

The keyboard shortcut for the adding the ports are ALT+F2 for input ports and ALT+F4 for the output ports.

Delete ports by right click on models

1. **Select the model on the tree or in the block view of the model.**
2. **Right click and select input/output port -> Delete -> Port.**

Delete ports by right click on ports in the block model

1. **Right click within the area of the port.**

2. **A small menu with three options Delete, Rename, Coupling opens up.**

3. **Select Delete.**

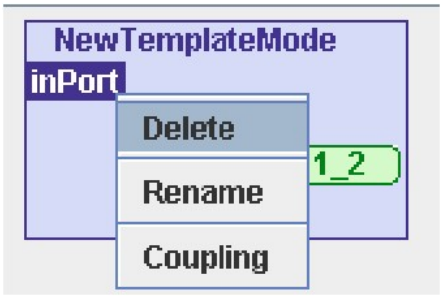


Figure 1 Right Click Menu on Port

Couplings

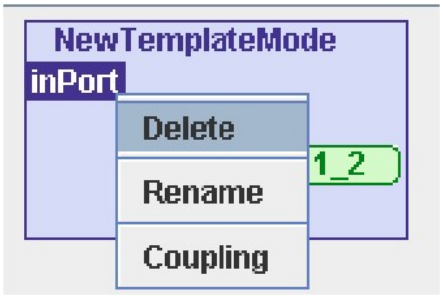


Figure 1 Right Click Menu Port

COUPLINGS

The couplings between two ports can be formed using the right click menu that was introduced in the Figure 1

The steps for adding the couplings are defined below:

1. **The port to be chosen as the source is right clicked on at first to popup the menu shown in Figure 1 and Coupling is chosen.**

2. **The port to be chosen as the destination port is right clicked and the same option is selected.**

3. **If the source and destination was specified conforming to the axioms of DEVS, the couplings would be formed.**

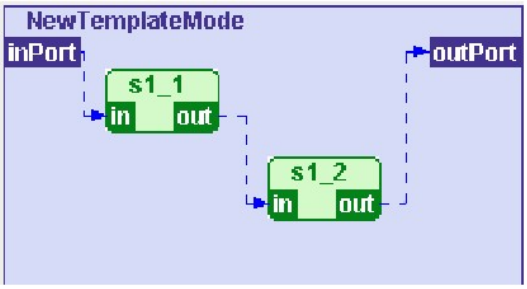


Figure 2 Model with couplings

Arizona Center for Integrative Modeling and Simulation

This help file has been generated by the freeware version of HelpNDoc

State Variables

Arizona Center for Integrative Modeling and Simulation

This help file has been generated by the freeware version of HelpNDoc

Create State Variables

CREATE STATE VARIABLES

The State variables can be added to the models using the following method

Adding ports by right click on models

1. **Select the model on the tree or in the block view of the model.**
2. **Right click and select States -> Add -> State Variable.**
3. **Give a name for the variable, mention the type of the variable and also mention the value for the variable.**
4. **The list of state variables for a model can be seen as a tool tip when the mouse hovers over the model's block view.**

Arizona Center for Integrative Modeling and Simulation

This help file has been generated by the freeware version of HelpNDoc

Delete State Variables

DELETE STATE VARIABLES

The state variable can be deleted from the model using the following method

Delete state variables by right click on models

1. **Select the model on the tree or in the block view of the model.**
2. **Right click and select States -> Delete -> State Variable.**

Arizona Center for Integrative Modeling and Simulation
This help file has been generated by the freeware version of HelpNDoc

Modify State Variables

MODIFY STATE VARIABLES

The state variable can be deleted from the model using the following method

Modify state variables by right click on models

1. **Select the model on the tree or in the block view of the model.**
2. **Right click and select States -> Modify -> State Variable.**
3. **Select the model from the drop down list.**
4. **Change parameters as needed.**

Arizona Center for Integrative Modeling and Simulation
This help file has been generated by the freeware version of HelpNDoc

Metrics/Views

METRICS/VIEWS

Arizona Center for Integrative Modeling and Simulation

This help file has been generated by the freeware version of HelpNDoc

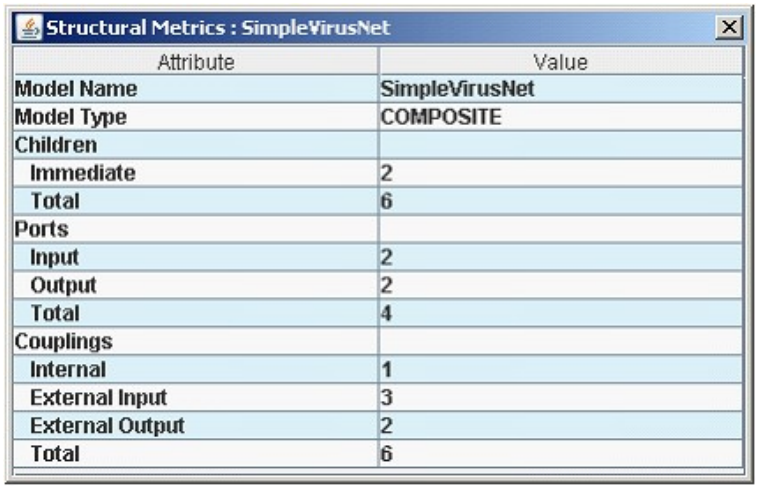
Structural/Behavioral Metrics

STRUCTURE/BEHAVIORAL METRICS

The user can look at the structural metrics using the following methods

View Structural/Behavioral metrics using menu bar

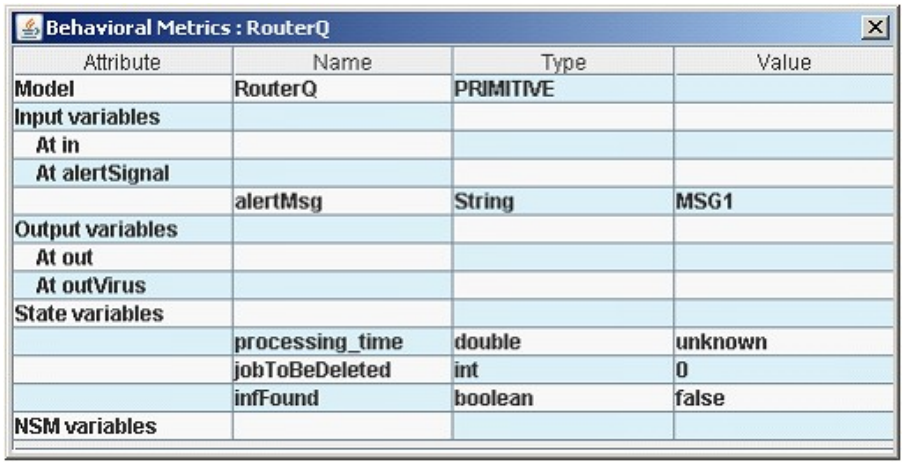
1. **Select the model on the tree or in the block view of the model.**
2. **Select the menu Metrics/Views -> Structure or Behavior.**



Structural Metrics : SimpleVirusNet

Attribute	Value
Model Name	SimpleVirusNet
Model Type	COMPOSITE
Children	
Immediate	2
Total	6
Ports	
Input	2
Output	2
Total	4
Couplings	
Internal	1
External Input	3
External Output	2
Total	6

Figure 1 Structural Metrics of a Model



Behavioral Metrics : RouterQ

Attribute	Name	Type	Value
Model	RouterQ	PRIMITIVE	
Input variables			
At in			
At alertSignal			
	alertMsg	String	MSG1
Output variables			
At out			
At outVirus			
State variables			
	processing_time	double	unknown
	jobToBeDeleted	int	0
	infFound	boolean	false
NSM variables			

Figure 2 Behavioral Metrics of a Model

View Structural/Behavioral metrics using right click on models

1. **Select the model on the tree or in the block view of the model.**
2. **Right click and select View -> Metrics -> Structure or Behavioral**

Arizona Center for Integrative Modeling and Simulation
This help file has been generated by the freeware version of HelpNDoc

Instance Models

Once the template models have been defined the models can be instantiated. Following are the steps to be administered by the user to ensure complete process.

Arizona Center for Integrative Modeling and Simulation

Create Instance Models

CREATE INSTANCE MODELS

1. **From CoSMoS's menu select Model -> Create Instance Models.**
2. **If the template model has specialized models in the hierarchy the user needs to select the specializations for the model from a dialog box. As shown in Figure 21**

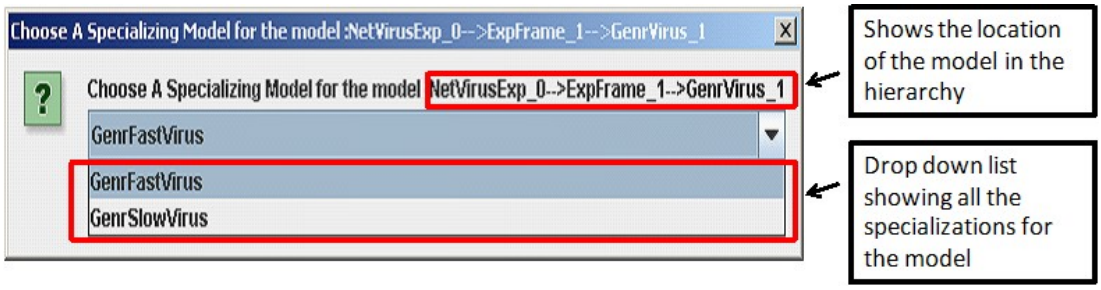


Figure 1 Selecting Specialization

After creating the Instance Model, a unique naming scheme produces the following

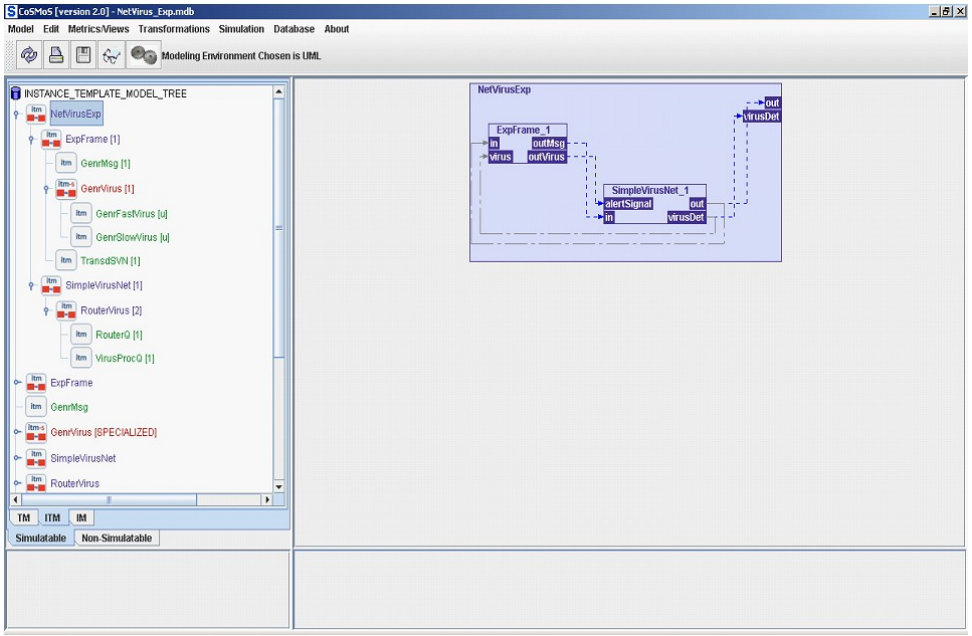


Figure 2 Instance Template Model View

Export Models

EXPORT MODELS

Instantiated models can be exported into their Java implementation.

1. **Go to the IM view by selecting the IM tab in the "Simulatable" section.**
2. **Select the model on the tree or in the block view of the model.**
3. **From the popped menu select the option Export -> DEVSJAVA Model.**
4. **Before the file is stores a dialog box opens up showing the location of the Java files that were exported. The user can look for the location of the files by viewing the properties.**

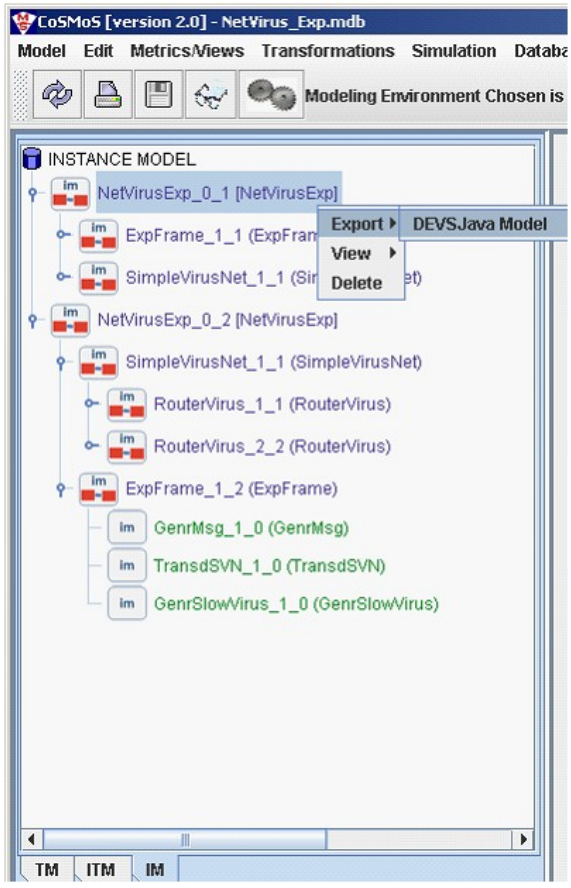


Figure 1 Export Model Menu

Add Behavior

ADD BEHAVIOR

The instance models created are not complete in terms of the atomic models. The transition functions have to be added into the exported java files that were created in the previous section. The editor in CoSMoS can be used to edit the models.

1. **Select the model on the tree or in the block view of the model.**
2. **Select the option from the menu View -> DEVSJAVA Source Code.**

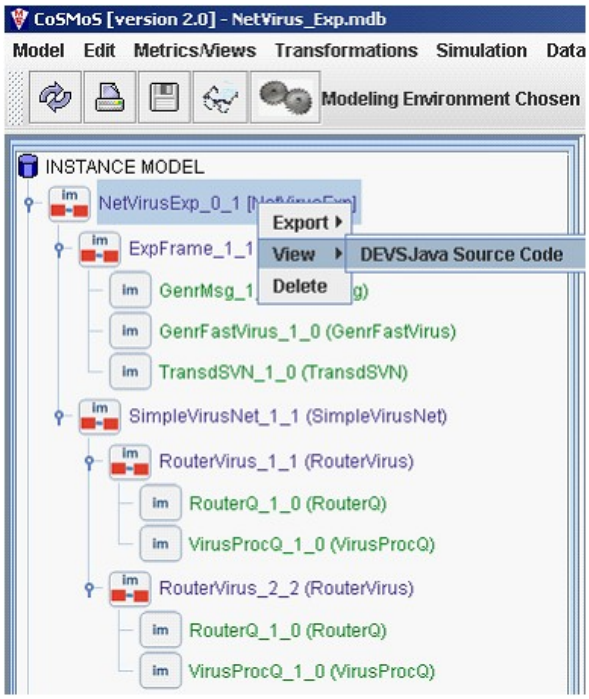


Figure 1 View DEVSJAVA Source Code

The editor opens up with the exported java file. The editor is located in the right hand lower section of the window.

The structure of the model is locked for users to edit and it is shown using the shading in the model.

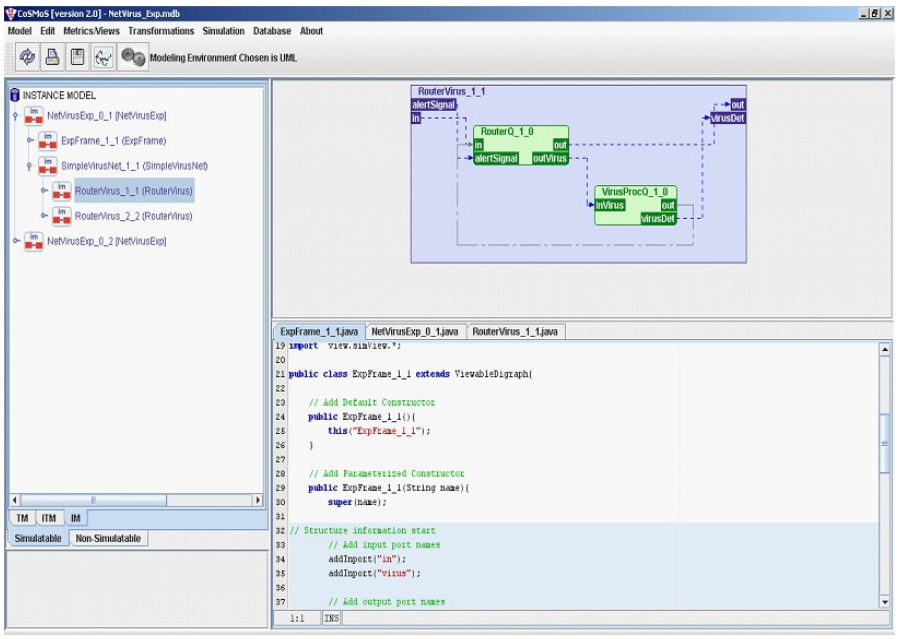


Figure 2 CoSMoS with Editor

After all the models have been edited, save it using the option available in the menu Model -> Save Source Code.

Arizona Center for Integrative Modeling and Simulation
This help file has been generated by the freeware version of HelpNDoc
Simulate Models

SIMULATE MODELS

Arizona Center for Integrative Modeling and Simulation

This help file has been generated by the freeware version of HelpNDoc
Tracking Models

TRACKING MODELS

Before the actual simulation the ports can be selected by the modeler.

1. **Select the option of 'Track' from the 'Simulation Menu'.**
2. **When selecting the option of 'Track', the model that was selected last from the tree menu forms the root model.**
3. **Now left click on any of the ports. If selected the text gets the back ground color of the port and the background gets the white color. The port can be clicked on again to get unselected and the color of the port reverses.**

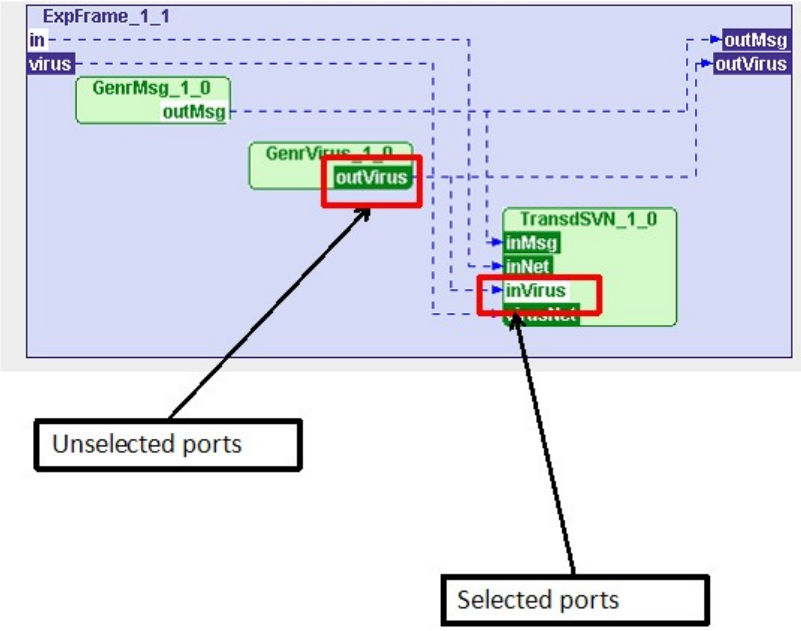


Figure 1 Selecting Ports

Simulating Models

SIMULATING MODELS

1. **The system remembers the root model from the tracking.**
2. **Select the option of 'Start Simulation' from the 'Simulation Menu'.**
3. **A dialog box showing the options of viewing the Output Trajectory is shown.(Figure 1)**

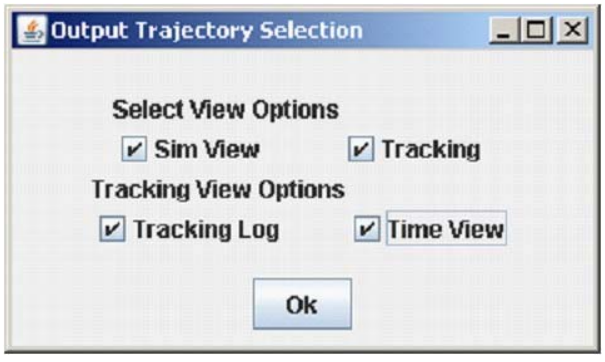


Figure 1 Select Output Trajectory View

4. **The system compiles the models that were updated by adding behavior, if there is error the models cannot be loaded for simulation.**

If models are successfully compiled and loaded, the controls for the simulation appear. (Figure 2)

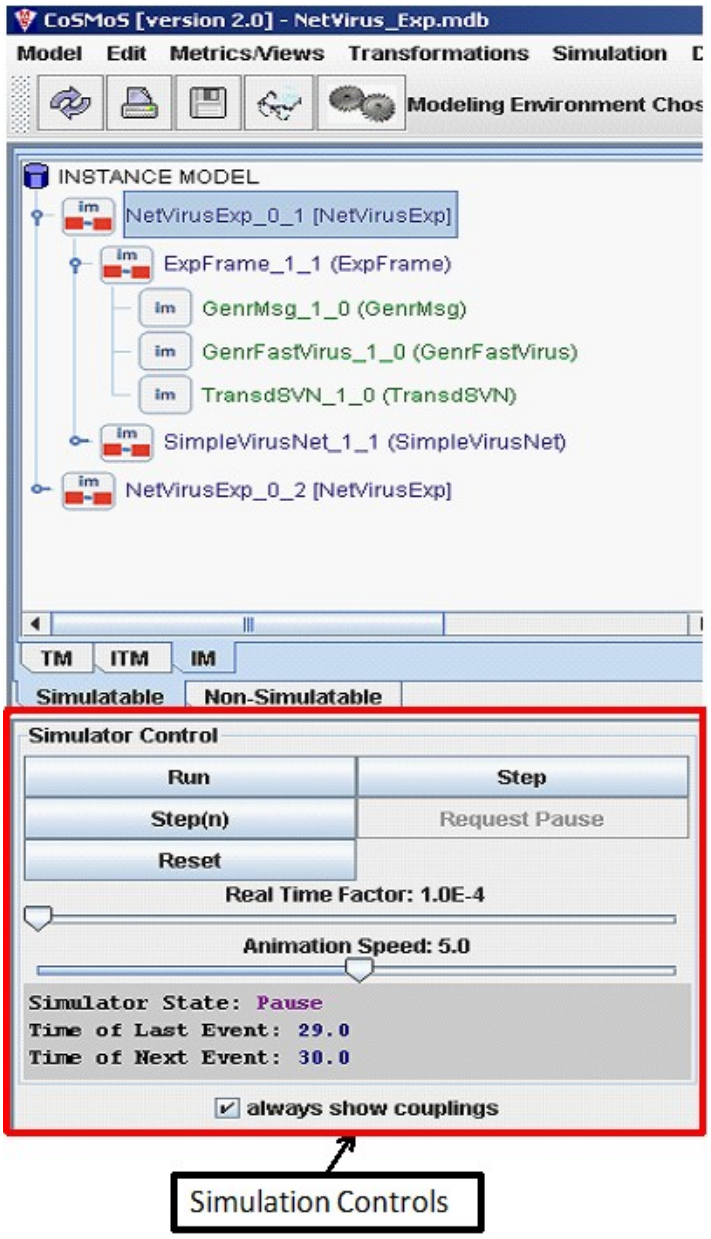


Figure 2 Simulation Controls

OUTPUT TRAJECTORY VIEWERS

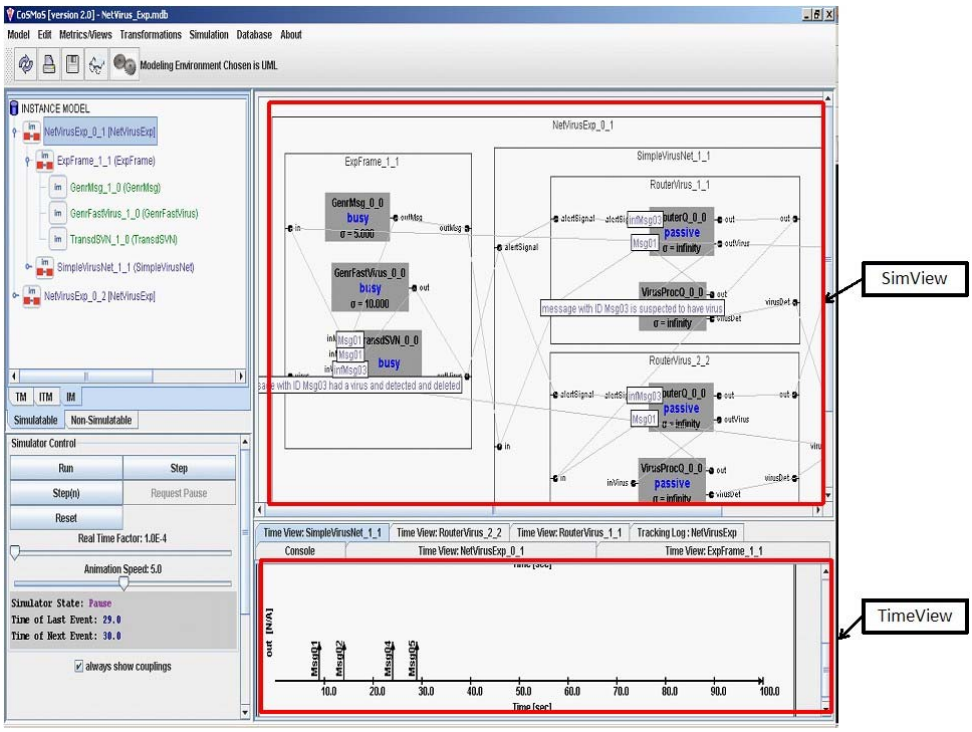


Figure 1 SimView and TimeView

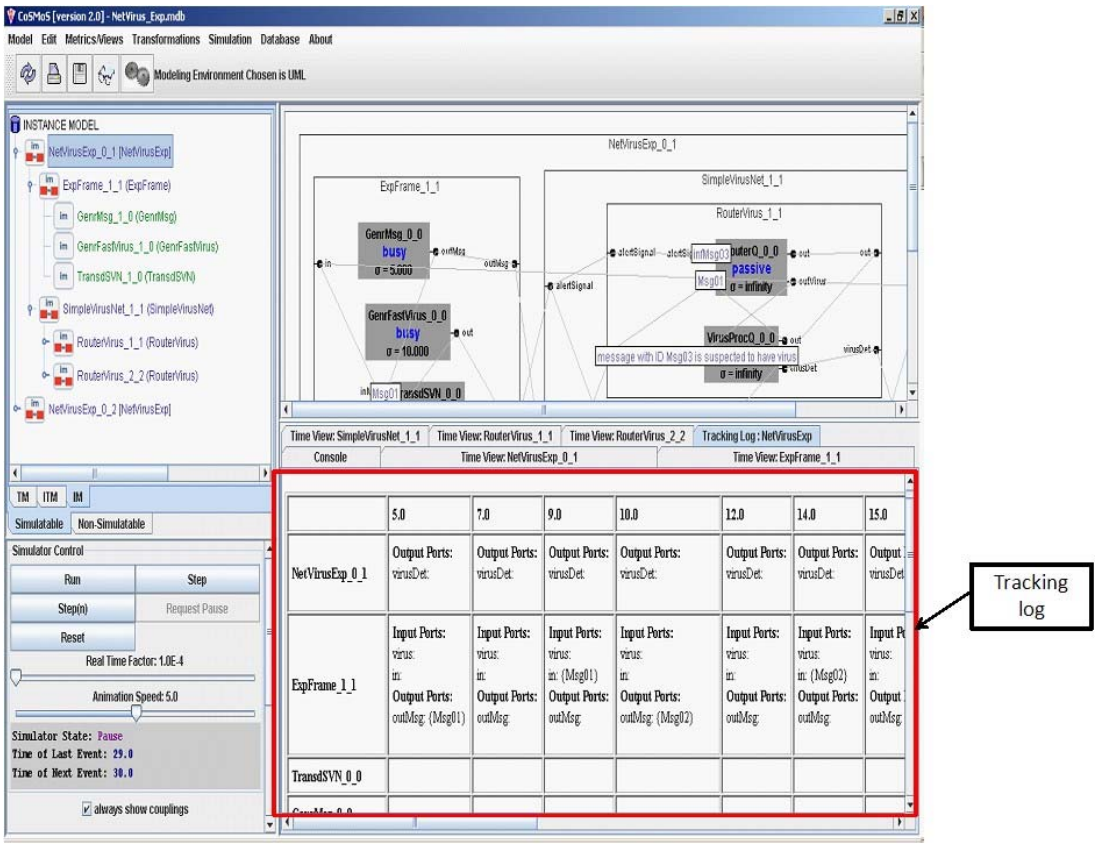


Figure 2 Tracking Log

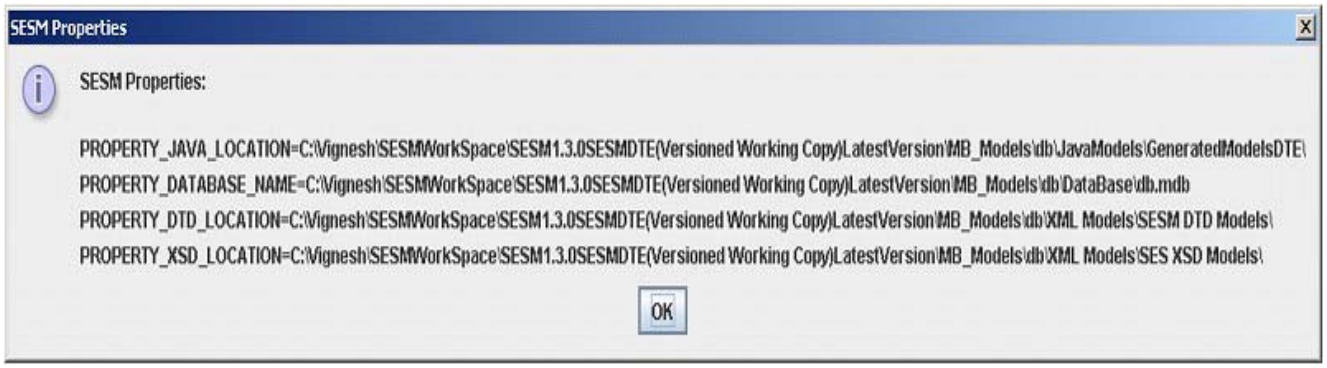
Miscellaneous

Arizona Center for Integrative Modeling and Simulation

View Properties

VIEW PROPERTIES

1. **From the menu bar select Model**
2. **Select Model -> View Property File**



Initialize Database

INITIALIZE DATABASE

1. **From menu bar select DATABASE**
2. **Select Database -> Initialize.**

Caution: initialization removes all models that are contained in the database. This operation cannot be undone.

Arizona Center for Integrative Modeling and Simulation
This help file has been generated by the freeware version of HelpNDoc

Guidelines

Arizona Center for Integrative Modeling and Simulation

This help file has been generated by the freeware version of HelpNDoc

Coupled Models

Arizona Center for Integrative Modeling and Simulation

This help file has been generated by the freeware version of HelpNDoc

Change Constructor Outside CoSMos Environment

CHANGE CONSTRUCTOR OUTSIDE OF CoSMoS ENVIRONMENT

The default declaration of the constituent models can be changed outside the CoSMoS environment using different editors like notepad, eclipse and text pad. This is to match the new constructors that were added for the atomic models. The new additions are shown by color shaded areas.

```
/*
*                Copyright Author
* (USE & RESTRICTIONS - Please read COPYRIGHT file)

* Version          : XX.XX
* Date             : 8/20/08 4:04 PM
*/

// Default Package
package MB_Models.myDB.JavaModels.GeneratedModelsDEVS_Suite;

import java.awt.*;
import java.io.*;
import java.util.*;
import DevsSuite.GenCol.*;
import DevsSuite.model.modeling.*;
import DevsSuite.view.modeling.ViewableDigraph;
import DevsSuite.view.modeling.ViewableAtomic;
import DevsSuite.model.simulation.*;
import DevsSuite.view.modeling.*;
import DevsSuite.view.simView.*;

public class efp_0_1 extends ViewableDigraph{

// Add Default Constructor
public efp_0_1(){
this("efp_0_1");
}

// Add Parameterized Constructor
public efp_0_1(String name){
super(name);

// Structure information start
// Add input port names
```

```
// Add output port names

//add test input ports:

// Initialize sub-components
ViewableDigraph expf_1_1 =  new expf_1_1("expf_1_1",10,100);
ViewableAtomic p_1_0 =  new p_0_0("p_1_0",25);

// Add sub-components
add(expf_1_1);
add(p_1_0);


// Add Couplings
addCoupling(expf_1_1, "out", p_1_0, "in");
addCoupling(p_1_0, "out", expf_1_1, "in");

// Structure information end
initialize();
}


/**
* Automatically generated by the SimView program.
* Do not edit this manually, as such changes will get overwritten.
*/
public void layoutForSimView()
{
preferredSize = new Dimension(591, 332);
(ViewableComponent)withName("p_1_0")).setPreferredLocation(new Point(21, 23));

((ViewableComponent)withName("expf_1_1")).setPreferredLocation(new Point(132, 85));
}
}
```

Arizona Center for Integrative Modeling and Simulation

Adding New Constructors

ADDING NEW CONSTRUCTORS

The modeler can add new constructor in addition to the code generated by CoSMoS. The color shaded area shows the new constructor added to the model. The newly added constructors should have the complete structure as in the default parameterized constructor generated by CoSMoS.

```
/*
*                    Copyright Author
* (USE & RESTRICTIONS - Please read COPYRIGHT file)
```

```
* Version              : XX.XX
* Date                 : 8/20/08 4:04 PM
*/

// Default Package
package MB_Models.myDB.JavaModels.GeneratedModelsDEVS_Suite;

import java.awt.*;
import java.io.*;
import java.util.*;
import DevsSuite.GenCol.*;
import DevsSuite.model.modeling.*;
import DevsSuite.view.modeling.ViewableDigraph;
import DevsSuite.view.modeling.ViewableAtomic;
import DevsSuite.model.simulation.*;
import DevsSuite.view.modeling.*;
import DevsSuite.view.simView.*;

public class expf_1_1 extends ViewableDigraph{

  // Add Default Constructor
  public expf_1_1(){
    this("expf_1_1",10,500);
  }

  // Add Parameterized Constructor

  public expf_1_1(String name){
    super(name);

// Structure information start
    // Add input port names
    addInport("in");

    // Add output port names
    addOutport("out");
    addOutport("result");

    //add test input ports:

    // Initialize sub-components
    ViewableAtomic g_1_0 = new g_0_0("g_1_0");
    ViewableAtomic t_1_0 = new t_0_0("t_1_0");

      // Add sub-components
    add(g_1_0);
    add(t_1_0);

    // Add Couplings
    addCoupling(this, "in", t_1_0, "solved");
```

```
        addCoupling(this, "out", g_1_0, "out");
        addCoupling(this, "result", t_1_0, "out");
        addCoupling(g_1_0, "out", t_1_0, "ariv");
        addCoupling(t_1_0, "out", g_1_0, "stop");
// Structure information end
        initialize();
        }

public expf_1_1(String nm,double int_arr_t,double observe_t){

        super(nm);
                addInport("in");
                addOutport("out");
                addOutport("result");

                ViewableAtomic g_1_0 = new g_0_0("g_1_0",int_arr_t);
                ViewableAtomic t_1_0 = new t_0_0("t_1_0",observe_t);

                add(g_1_0);
                add(t_1_0);

                addTestInput("start",new entity());
                addTestInput("stop",new entity());
                addTestInput("in",new entity("job0"));
                addTestInput("in",new entity("job1"));

                initialize();


                addCoupling(g_1_0,"out",t_1_0,"ariv");
                addCoupling(this,"in",t_1_0,"solved");
                addCoupling(t_1_0,"out",g_1_0,"stop");
                addCoupling(this,"start",g_1_0,"start");
                addCoupling(this,"stop",g_1_0,"stop");
                addCoupling(g_1_0,"out",this,"out");
                addCoupling(t_1_0,"out",this,"result");

        }


  /**
   * Automatically generated by the SimView program.
   * Do not edit this manually, as such changes will get overwritten.
   */
  public void layoutForSimView()
  {
    preferredSize = new Dimension(591, 332);
    ((ViewableComponent)withName("t_1_0")).setPreferredLocation(new Point(42, 140));
    ((ViewableComponent)withName("g_1_0")).setPreferredLocation(new Point(57, 65));
  }
  }
```

Arizona Center for Integrative Modeling and Simulation

Atomic Models

Arizona Center for Integrative Modeling and Simulation

This help file has been generated by the freeware version of HelpNDoc

Adding New Constructor

ADDING NEW CONSTRUCTORS

The area shaded by color in the code shows the newly added constructor for the model.
The constructor in turn calls the default parameterized constructor and then performs the needed operation with the newly added parameters.
This is a good approach since the structural integrity is maintained.

An alternative is to create a new constructor and copy all the structural information from the parameterized constructor that was automatically generated. The modeler should be careful while doing the cut and paste of structural information.

```
/*
*                    Copyright Author
* (USE & RESTRICTIONS - Please read COPYRIGHT file)

* Version                 : XX.XX
* Date                  : 8/20/08 4:04 PM
*/

// Default Package
package MB_Models.myDB.JavaModels.GeneratedModelsDEVS_Suite;

import  java.awt.*;
import  java.io.*;
import  java.util.*;
import  DevsSuite.GenCol.*;
import  DevsSuite.model.modeling.*;
import  DevsSuite.view.modeling.ViewableDigraph;
import  DevsSuite.view.modeling.ViewableAtomic;
import  DevsSuite.model.simulation.*;
import  DevsSuite.view.modeling.*;
import  DevsSuite.view.simView.*;

public  class g_0_0 extends ViewableAtomic{

    protected  double processing_time;
    protected  double clock;
    protected  double int_arr_time;
    protected  int count;

    // Add Default Constructor
    public g_0_0(){
        this("g_0_0",30);    }

    // Add Parameterized Constructors
    public g_0_0(String name){
        super(name);
```

```
// Structure information start
        // Add input port names
        addInport("in");
        addInport("start");
        addInport("stop");

        // Add output port names
        addOutport("out");
// Structure information end

//add test input ports:
         addTestInput("start",new entity(""));
         addTestInput("stop",new entity(""));

        initialize();
    }



public g_0_0(String name,double Int_arr_time){
              this(name);
            int_arr_time = Int_arr_time ;
                  }

    // Add initialize function
    public void initialize(){
        super.initialize();
        phase = "passive";
        sigma = INFINITY;
        clock = 0;
        int_arr_time = 0;
        count = 0;
    }

    // Add external transition function
    public void deltext(double e, message x){

                Continue(e);

                clock = clock+e;
                if(phaseIs("passive")){
```

```
                for (int i=0; i< x.getLength();i++)
                        if (messageOnPort(x,"start",i))
                                holdIn("active",int_arr_time);
        }
        if(phaseIs("active"))
                for (int i=0; i< x.getLength();i++)
                        if (messageOnPort(x,"stop",i))
                                phase = "finishing";
}
```

// Add internal transition function

```
public void deltint(){

    clock=clock+sigma;
        System.out.println("----------"+clock);
        if(phaseIs("active")){
                count = count +1;
                holdIn("active",int_arr_time);
        }
        else passivate();

}
```

// Add confluent function

```
public void deltcon(double e, message x){
}
```

// Add output function

```
public message out(){
    message  m = new message();
        content con = makeContent("out",
                        new entity("job" + count));
        m.add(con);

        return m;
}

    public void showState(){
        super.showState();
        System.out.println("int_arr_t: " + int_arr_time);
    }
```

```
    public String getTooltipText(){
        return
        super.getTooltipText()
        +"\n"+" int_arr_time: " + int_arr_time
        +"\n"+" count: " + count;
    }

// Add Show State function
}
```

Arizona Center for Integrative Modeling and Simulation

Adding Test input/output ports

ADDING TEST INPUT/OUTPUT PORTS

The color shaded area shows the newly added test inputs to the model.

The section for adding the Test inputs/outputs is present outside the guarded area and can be added by the modeler using the CoSMoS IDE.

```
/*
*                    Copyright Author
* (USE & RESTRICTIONS - Please read COPYRIGHT file)

* Version                : XX.XX
* Date                   : 8/20/08 4:04 PM
*/

// Default Package
package MB_Models.myDB.JavaModels.GeneratedModelsDEVS_Suite;

import  java.awt.*;
import  java.io.*;
import  java.util.*;
import  DevsSuite.GenCol.*;
import  DevsSuite.model.modeling.*;
import  DevsSuite.view.modeling.ViewableDigraph;
import  DevsSuite.view.modeling.ViewableAtomic;
import  DevsSuite.model.simulation.*;
import  DevsSuite.view.modeling.*;
import  DevsSuite.view.simView.*;

public class g_0_0 extends ViewableAtomic{

    protected double processing_time;
```

```
    protected double clock;
    protected double int_arr_time;
    protected int count;

    // Add Default Constructor
    public g_0_0(){
        this("g_0_0",30);   }

    // Add Parameterized Constructors
    public g_0_0(String name){
        super(name);
// Structure information start
        // Add input port names
        addInport("in");
        addInport("start");
        addInport("stop");

        // Add output port names
        addOutport("out");

// Structure information end

//add test input ports:
          addTestInput("start",new entity(""));
      addTestInput("stop",new entity(""));


        initialize();
    }



public g_0_0(String name,double Int_arr_time){
                this(name);
            int_arr_time = Int_arr_time ;
                }

    // Add initialize function
    public void initialize(){
        super.initialize();
        phase = "passive";
        sigma = INFINITY;
```

```
    clock = 0;
    int_arr_time = 0;
    count = 0;
}

// Add external transition function
public void deltext(double e, message x){

        Continue(e);

        clock = clock+e;
        if(phaseIs("passive")){
            for (int i=0; i< x.getLength();i++)
                if (messageOnPort(x,"start",i))
                    holdIn("active",int_arr_time);
        }
        if(phaseIs("active"))
            for (int i=0; i< x.getLength();i++)
                if (messageOnPort(x,"stop",i))
                    phase = "finishing";
}

// Add internal transition function
public void deltint(){

    clock=clock+sigma;
        System.out.println("----------"+clock);
        if(phaseIs("active")){
            count = count +1;
            holdIn("active",int_arr_time);
        }
        else passivate();

}

// Add confluent function
public void deltcon(double e, message x){
}

// Add output function
public message out(){
    message  m = new message();
```

```
            content con = makeContent("out",
                        new entity("job" + count));
            m.add(con);

            return m;
    }

        public void showState(){
            super.showState();
            System.out.println("int_arr_t: " + int_arr_time);
        }

        public String getTooltipText(){
            return
            super.getTooltipText()
            +"\n"+" int_arr_time: " + int_arr_time
            +"\n"+" count: " + count;
        }

    // Add Show State function
    }
```

Arizona Center for Integrative Modeling and Simulation

Adding Behavior

ADDING BEHAVIOR

The modeler can add behavior to the transition functions (external and internal), confluent,out function. The code can be added to the section that is present in the definition of the function.
The color shaded shows the behavior that was added to the functions that were generated by CoSMoS.

```
/*
*                    Copyright Author
* (USE & RESTRICTIONS - Please read COPYRIGHT file)

* Version                : XX.XX
* Date                 : 8/20/08 4:04 PM
*/

// Default Package
package MB_Models.myDB.JavaModels.GeneratedModelsDEVS_Suite;

import  java.awt.*;
import  java.io.*;
import  java.util.*;
```

```
import DevsSuite.GenCol.*;
import DevsSuite.model.modeling.*;
import DevsSuite.view.modeling.ViewableDigraph;
import DevsSuite.view.modeling.ViewableAtomic;
import DevsSuite.model.simulation.*;
import DevsSuite.view.modeling.*;
import DevsSuite.view.simView.*;

public class t_0_0 extends ViewableAtomic{

    protected double processing_time;
    protected double clock;
    protected double total_ta;
    protected double observation_time;
    protected Map arrived, solved;

    // Add Default Constructor
    public t_0_0(){
        this("t_0_0",20);    }

    // Add Parameterized Constructors
    public t_0_0(String name){
        super(name);
// Structure information start
        // Add input port names
        addInport("ariv");
        addInport("in");
        addInport("solved");

        // Add output port names
        addOutport("out");

//add test input ports:

// Structure information end
        initialize();
    }


public t_0_0(String  name,double Observation_time){
                super(name);
                addInport("in");
                addOutport("out");
```

```
        addInport("ariv");
        addInport("solved");
        //addOutport("out");
        arrived = new HashMap();
        solved = new HashMap();
        observation_time = Observation_time;
        addTestInput("ariv",new entity("val"));
        addTestInput("solved",new entity("val"));
        initialize();
    }
```

// Add initialize function

```
public void initialize(){
    super.initialize();
    phase = "passive";
    sigma = INFINITY;
    clock = 0;
    total_ta = 0;
    observation_time = 0;
}
```

// Add external transition function

```
public void deltext(double e, message x){
            clock = clock + e;
     Continue(e);
     entity  val;
     for(int i=0; i< x.size();i++){
          if(messageOnPort(x,"ariv",i)){
               val = x.getValOnPort("ariv",i);
               arrived.put(val.getName(),new doubleEnt(clock));
          }
          if(messageOnPort(x,"solved",i)){
               val = x.getValOnPort("solved",i);
                                  if(arrived.containsKey(val.getName())){
                    //entity  ent = (entity)arrived.assoc(val.getName());
                    entity  ent = (entity)arrived.get(val.getName());

                    doubleEnt  num = (doubleEnt)ent;
                    double arrival_time = num.getv();

                    double turn_around_time = clock - arrival_time;
                    total_ta = total_ta + turn_around_time;
                    solved.put(val.getName(), new doubleEnt(clock));
               }
          }
     }
     show_state();

}
```

```
// Add internal transition function
public void deltint(){
        clock = clock + sigma;
    passivate();
    show_state();

}
// Add confluent function
public void deltcon(double e, message x){
}
// Add output function
public message out(){
        message m = new message();
    content con = makeContent("out",new entity("TA: "+compute_TA()));
    m.add(con);
    return m;

}
// Add Show State function

public void showState(){
        super.showState();
        System.out.println("arrived: " + arrived.size());
        System.out.println("solved: " + solved.size());
        System.out.println("TA: "+compute_TA());
        System.out.println("Thruput: "+compute_Thru());
    }


public double compute_TA(){
        double avg_ta_time = 0;
        if(!solved.isEmpty())
            avg_ta_time = ((double)total_ta)/solved.size();
        return avg_ta_time;
    }

    public double compute_Thru(){
        double thruput = 0;
        if(clock > 0)
            thruput = solved.size()/(double)clock;
        return thruput;
    }
```

```
public void show_state(){

        System.out.println("*************************************************");
        System.out.println("state of " + name + ": " );
        System.out.println("*************************************************");
        System.out.println("phase, sigma : "
                + phase + "," + sigma + "" );

        if (arrived != null && solved != null){
            System.out.println("no of jobs total :" + arrived.size() );
            System.out.println("total processed :" + solved.size() );
            System.out.println("AVG TA = " + compute_TA() );
            System.out.println("THRUPUT = " + compute_Thru() );
        }
    }

}
```

Arizona Center for Integrative Modeling and Simulation

Adding additional functions

ADDING OTHER KINDS OF BEHAVIORS

After the models have been generated additional functions such as showState(), getTooltipText() and other user defined functions can be added to the model. after the "// Add Show State function" marker in the generated model.

The color shaded area in the model shows the section of the code that was added after the model was generated.

```
/*
*                  Copyright Author
* (USE & RESTRICTIONS - Please read COPYRIGHT file)

* Version              : XX.XX
* Date                 : 8/20/08 4:04 PM
*/

// Default Package
package MB_Models.myDB.JavaModels.GeneratedModelsDEVS_Suite;

import java.awt.*;
import java.io.*;
import java.util.*;
import DevsSuite.GenCol.*;
import DevsSuite.model.modeling.*;
import DevsSuite.view.modeling.ViewableDigraph;
```

```
import DevsSuite.view.modeling.ViewableAtomic;
import DevsSuite.model.simulation.*;
import DevsSuite.view.modeling.*;
import DevsSuite.view.simView.*;

public class p_0_0 extends ViewableAtomic{


    protected double processing_time;
    protected entity job;

    // Add Default Constructor
    public p_0_0(){
        this("p_0_0",0);    }

    // Add Parameterized Constructors
    public p_0_0(String name){
        super(name);
// Structure information start
        // Add input port names
        addInport("in");
        addInport("none");

        // Add output port names
        addOutport("out");

//add test input ports:

// Structure information end
        initialize();
    }

public p_0_0(String  name,double  Processing_time){
                super(name);
                addInport("in");
                addOutport("out");
                addInport("none"); //allows testing for null input
                //which should cause only "continue"
                processing_time = Processing_time;
                addTestInput("in",new entity("job1"));
                addTestInput("in",new entity("job2"),20);
                addTestInput("none",new entity("job"));
        }
```

```
// Add initialize function
public void initialize(){
    super.initialize();
    phase = "passive";
    sigma = INFINITY;
    processing_time = 0;
    job =new entity("job");
}

// Add external transition function
public void deltext(double e, message x){
        Continue(e);

        if (phaseIs("passive"))
            for (int i=0; i< x.getLength();i++)
                if (messageOnPort(x,"in",i))
                {
                    job = x.getValOnPort("in",i);
                    holdIn("busy",processing_time);
                }

}

// Add internal transition function
public void deltint(){
        passivate();
        job = new entity("none");

}

// Add confluent function
public void deltcon(double e, message x){

        deltint();
        deltext(0,x);

}

// Add output function
public message out(){
        message   m = new message();
        if (phaseIs("busy")) {
```

```
                m.add(makeContent("out",job));
            }
            return m;

    }
```

// Add Show State function

```
public void showState(){
        super.showState();
        System.out.println("job: " + job.getName());
}

 public String getTooltipText(){
        return
        super.getTooltipText()
        +"\n"+"job: " + job.getName();
}
}
```

Arizona Center for Integrative Modeling and Simulation