IS3 Project Final Report (Part 3)

SysML to SAN model transformation

By Peter Popov

Centre for Software Reliability

City, University of London

p.t.popov@city.ac.uk

This document provides a sketch of an example of model transformation which illustrates the concepts of model transformation adopted in the development of the CHESS plugin.

As a test case to check that the plug-in operates correctly an existing SysML model, developed the subcontractor Intecs, of the ATM system, described in Part 1 of the report.

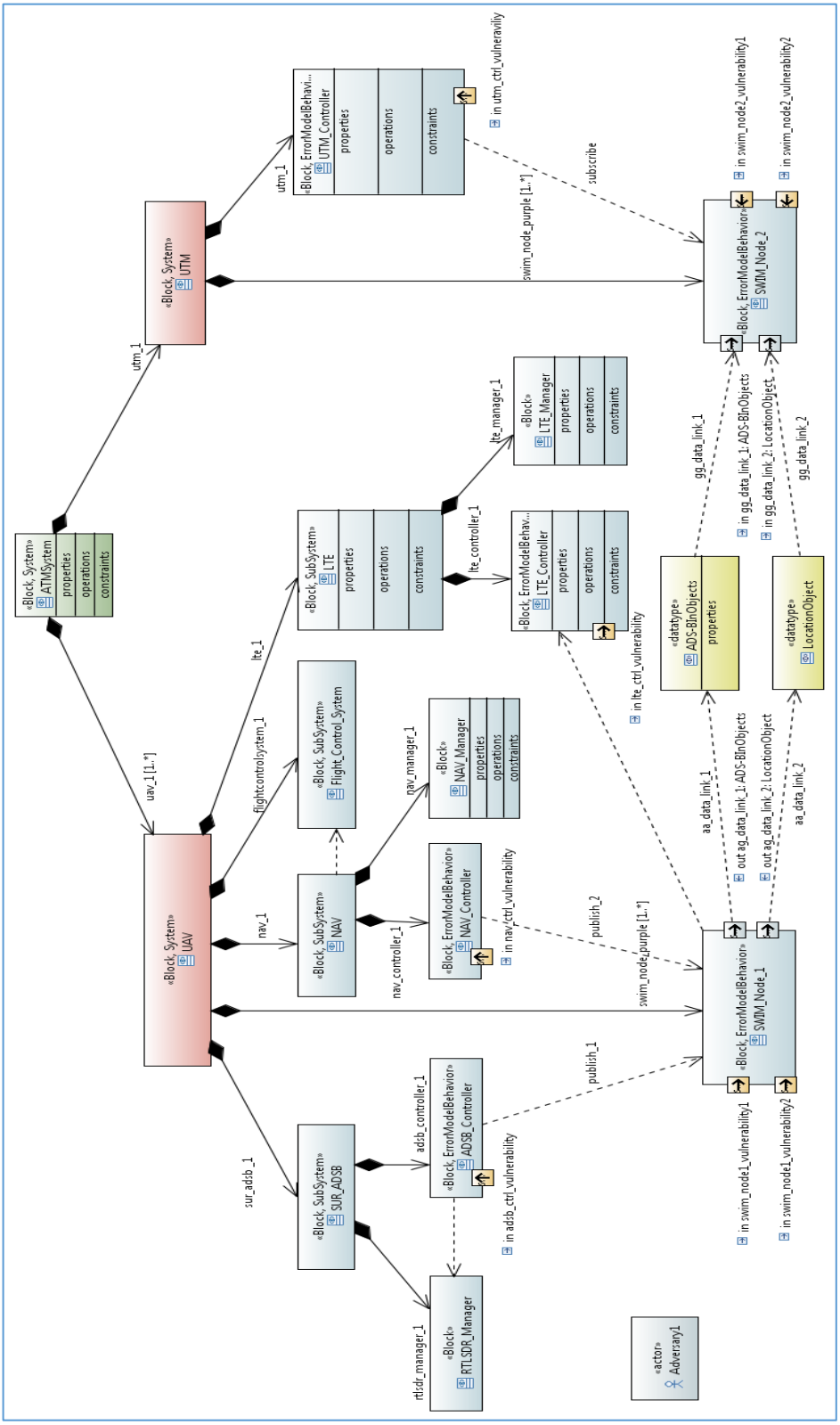
In Appendix 1 the essential parts of the SysML model are covered, which are processed by the plug-in in producing a SAN model.

In Appendix 2, the SAN model which was produced by the plug-in is documented, using the Mobius tool.

Appendix 1: The SysML system model

The example is of a SysML model developed for the ATM system, described elsewhere in the Final report. We include a few fragments of the model necessary to illustrate the SysML -> SAN transformation.

A1.1. System architecture as Block diagram

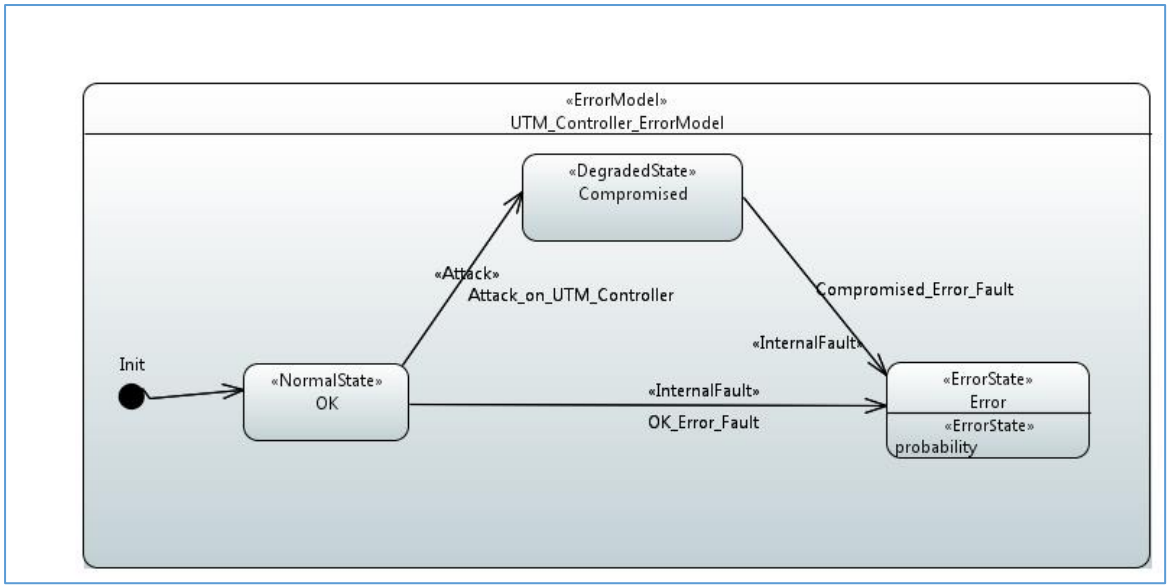


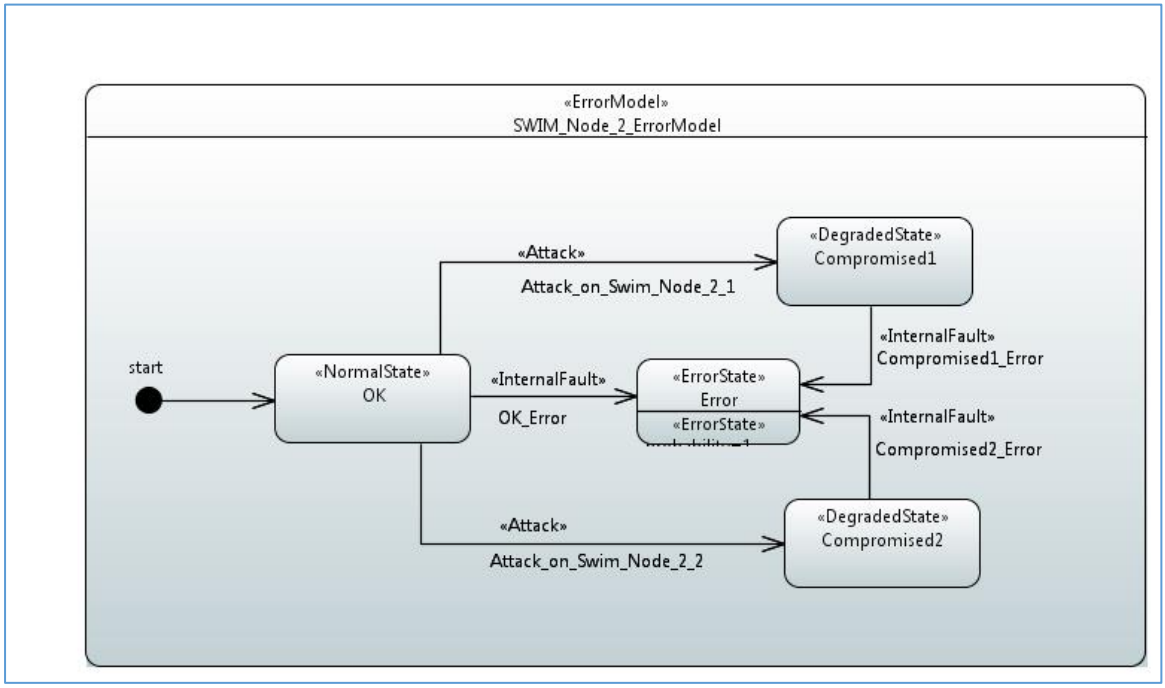
A1.2. State Machines used to model failure of SysML Blocks

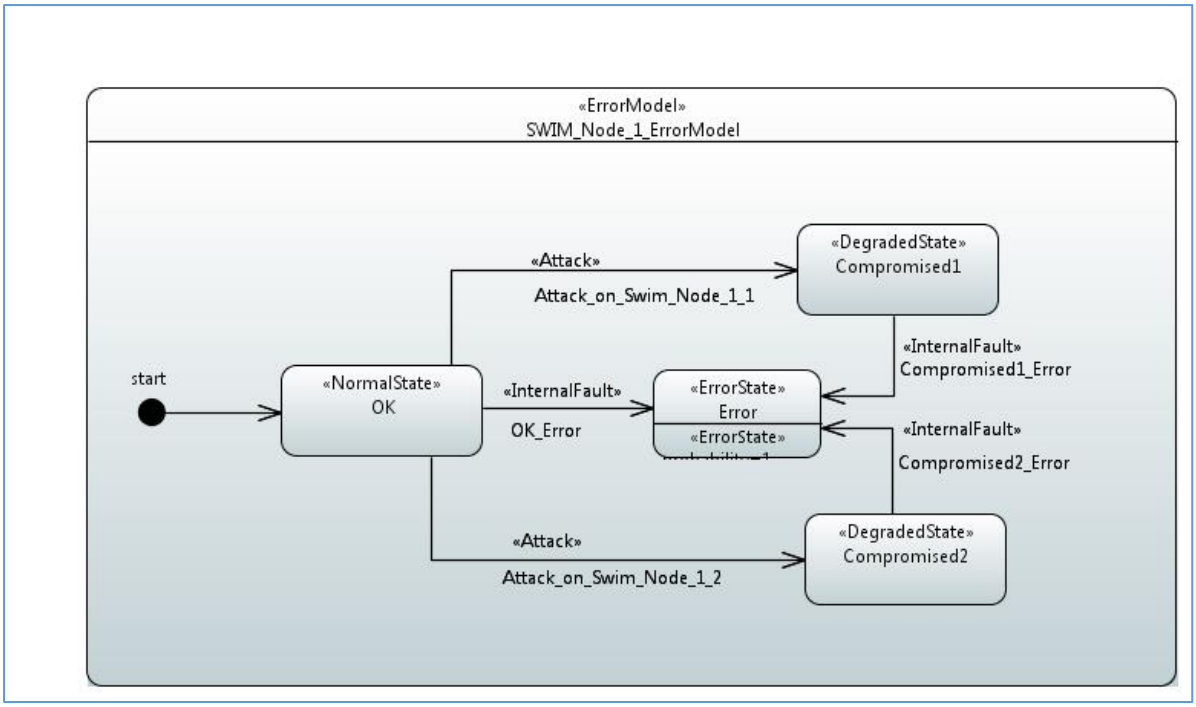
The failure behaviour of a SysML block, e.g. of a software component, is captured by state machines, which are stereotyped as “ErrorMode”. In such state machines there is a state representing the normal behaviour, typically called “OK” with stereotype “NormalState” and a number of states with stereotype “DegradedState” or “ErrorState” used to capture the effect of successful attacks (“CompromisedState”) and errors (or failures) of the component, respectively.

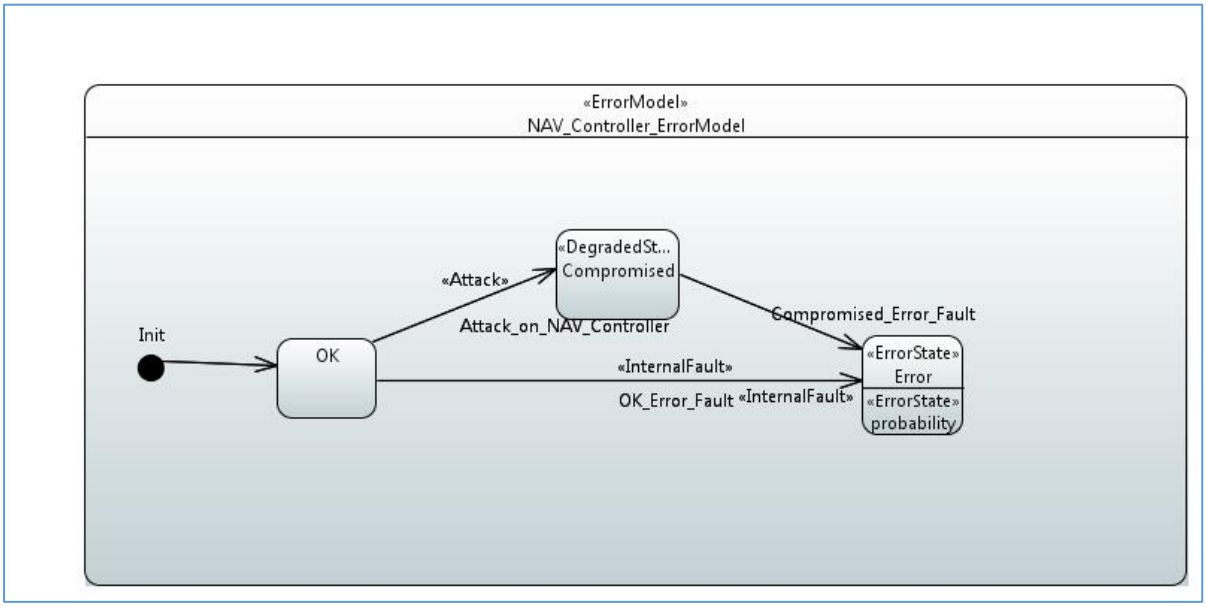
The state machine with stereotype “ErrorMode” can consist of a number of “ErrorState” states (modelling different failure modes) and “DegradedState” states, which model different ways of compromising a block (e.g. the software component is compromised by different attack types).

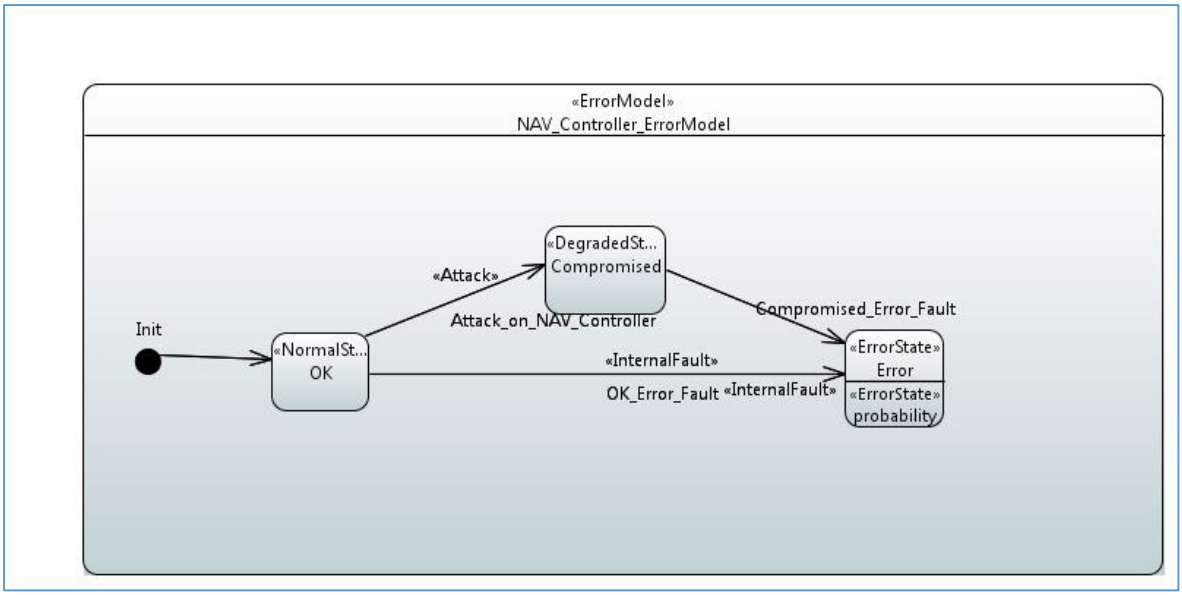
The diagrams shown below illustrate the use of “ErrorMode” state machines. There is a degree of similarity between these state machines, which is a consequence of the number of failure models/compromises that are modelled for the particular software components.

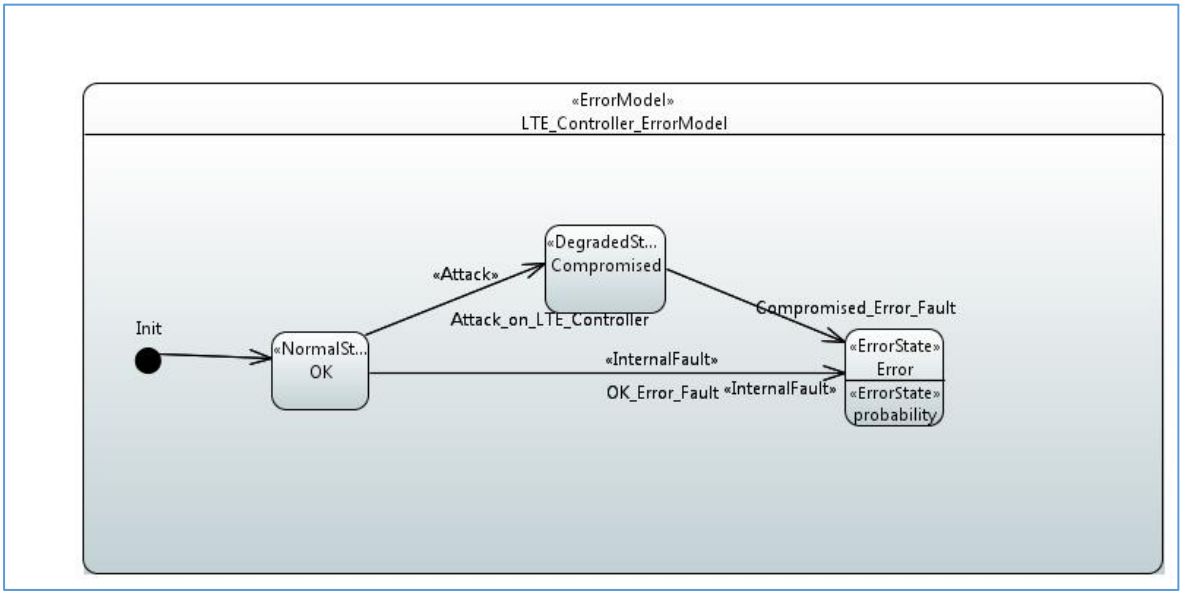


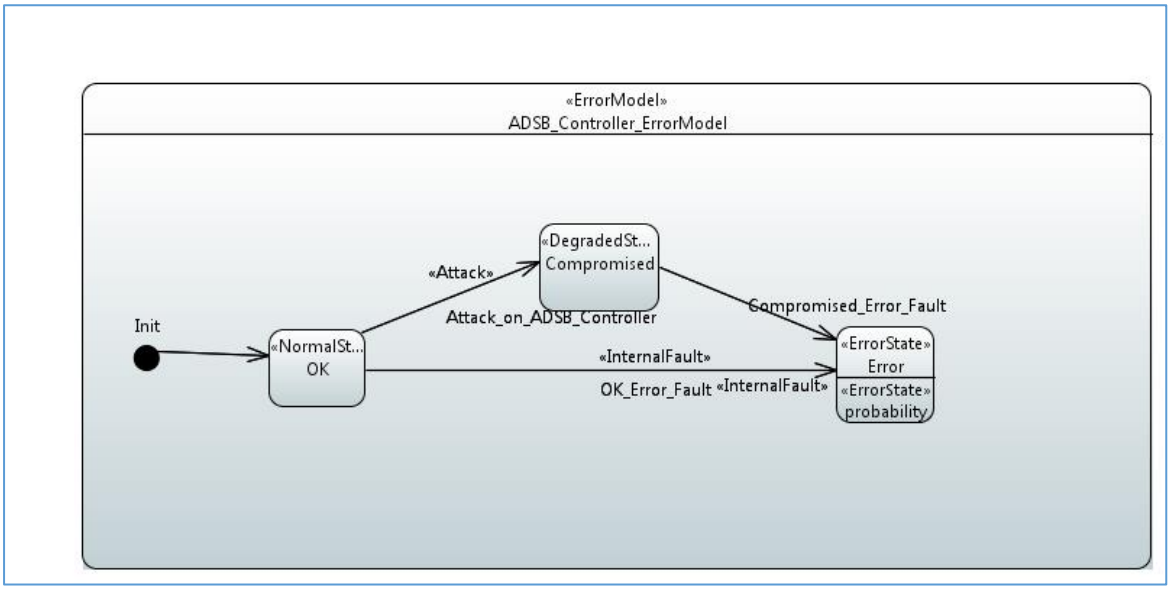



«ErrorModel»
SWIM_Node_1_ErrorModel
«DegradedState»
Compromised1
«Attack»
Attack_on_Swim_Node_1_1
«InternalFault»
Compromised1_Error
start
«NormalState»
OK
«InternalFault»
OK_Error
«ErrorState»
Error
«ErrorState»
«InternalFault»
Compromised2_Error
«Attack»
Attack_on_Swim_Node_1_2
«DegradedState»
Compromised2


«ErrorModel»
NAV_Controller_ErrorModel
«DegradedSt...
Compromised
«Attack»
Attack_on_NAV_Controller
Compromised_Error_Fault
Init
OK
«InternalFault»
OK_Error_Fault
«InternalFault»
«ErrorState»
Error
«ErrorState»
probability


«ErrorModel»
NAV_Controller_ErrorModel
«DegradedSt...
Compromised
«Attack»
Attack_on_NAV_Controller
Compromised_Error_Fault
Init
«NormalSt...
OK
«InternalFault»
OK_Error_Fault «InternalFault»
«ErrorState»
Error
«ErrorState»
probability

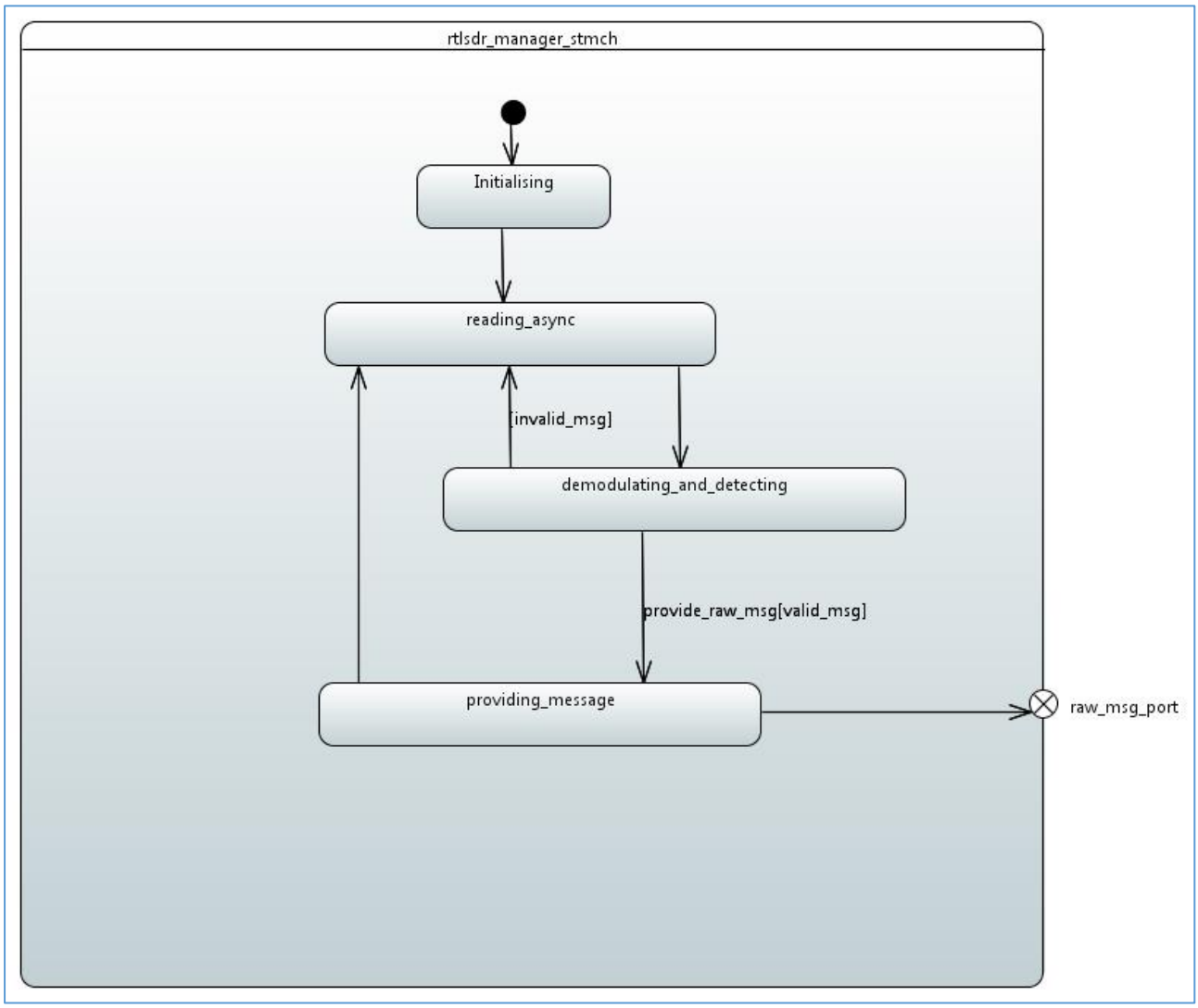

«ErrorModel»
LTE_Controller_ErrorModel
«DegradedSt...
Compromised
«Attack»
Attack_on_LTE_Controller
Compromised_Error_Fault
Init
«NormalSt...
OK
«InternalFault»
OK_Error_Fault «InternalFault»
«ErrorState»
Error
«ErrorState»
probability

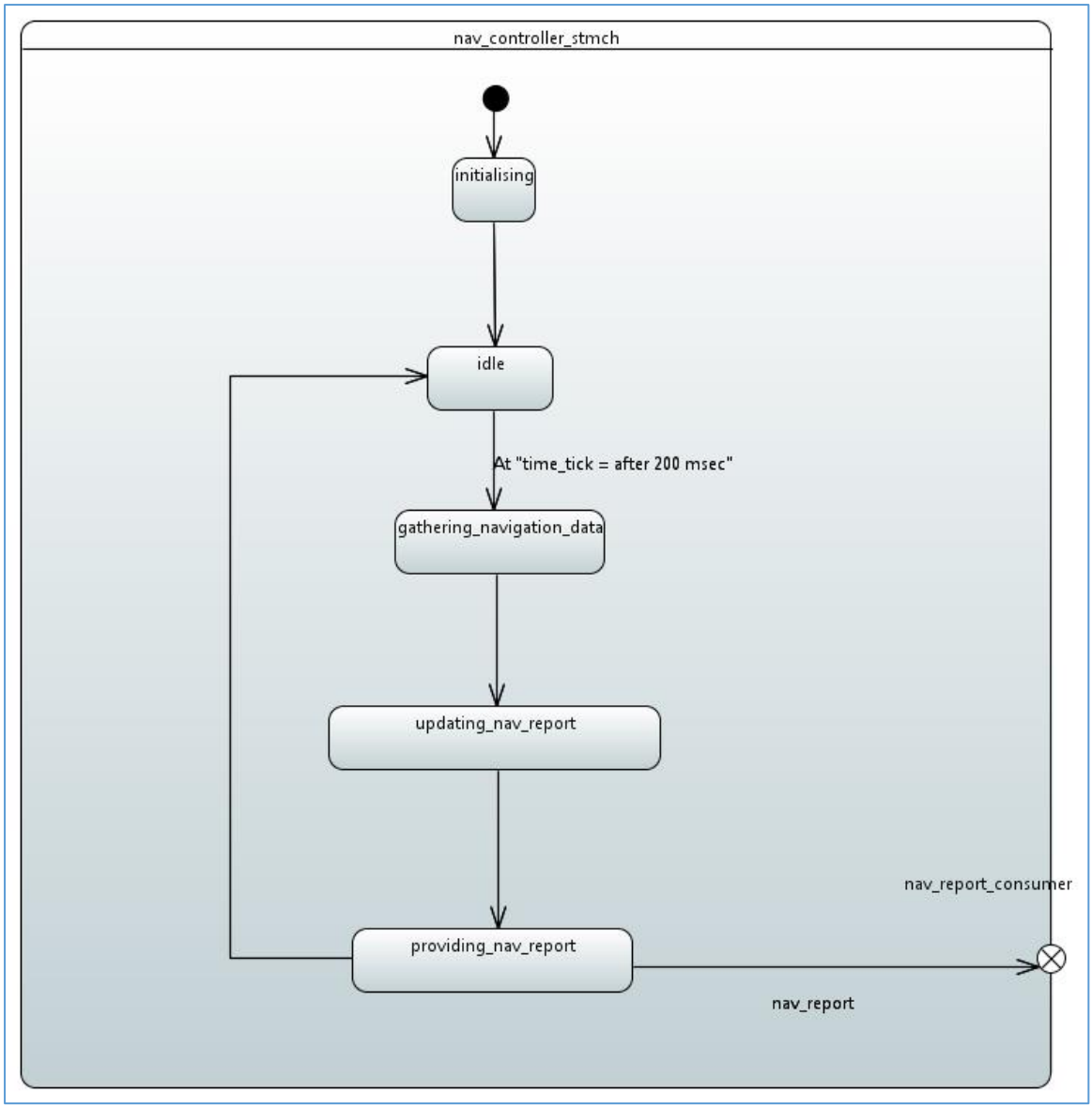

«ErrorModel»
ADSB_Controller_ErrorModel
«DegradedSt...
Compromised
«Attack»
Attack_on_ADSB_Controller
Compromised_Error_Fault
Init
«NormalSt...
OK
«InternalFault»
OK_Error_Fault «InternalFault»
«ErrorState»
Error
«ErrorState»
probability

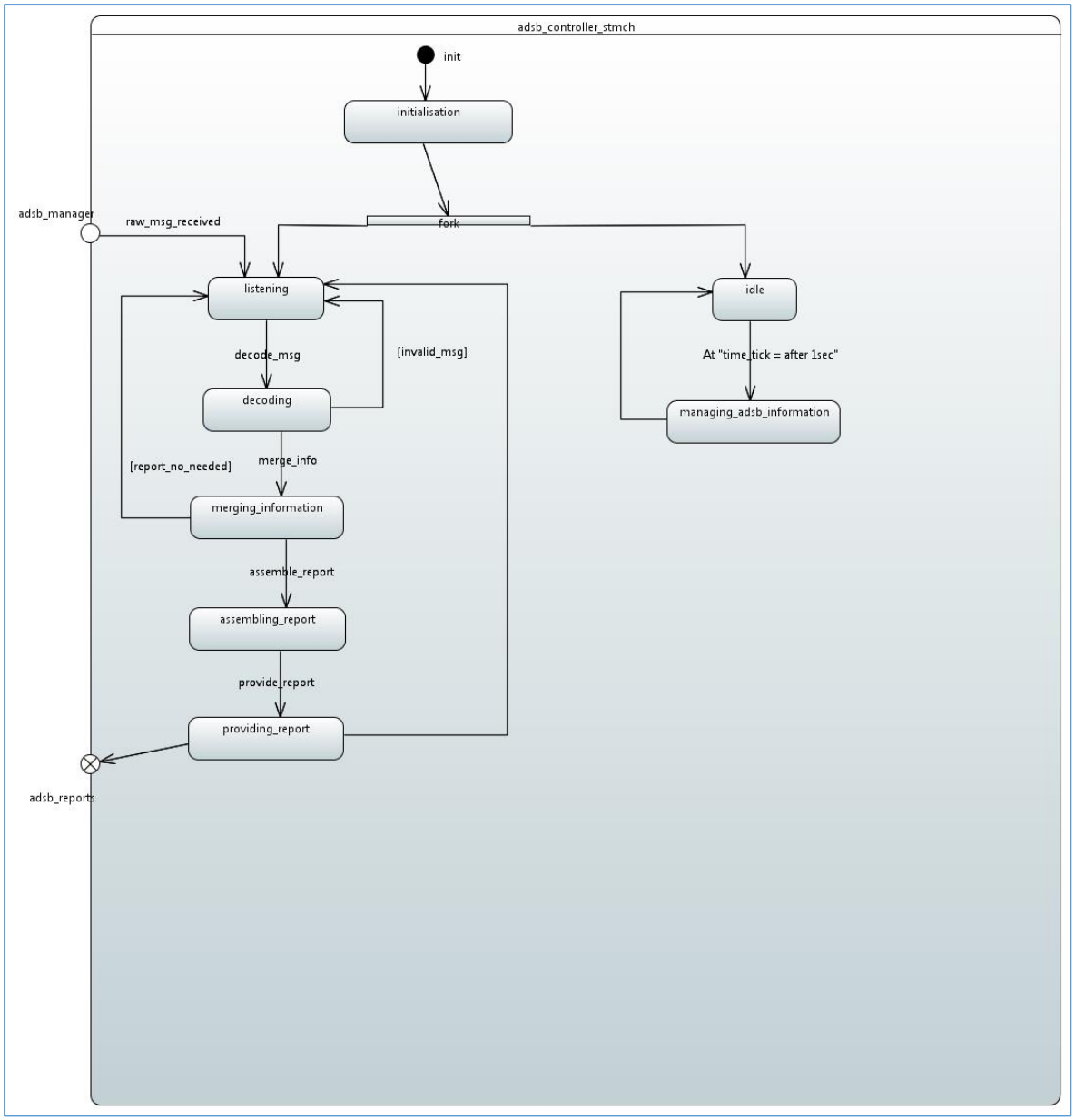
A1.3. State machines to model functionality of different blocks

State machines can also be used to model the functionality of the software blocks. Examples are shown below. While the syntax of state machines is the same (derived from the UML syntax), the state machines themselves are not stereotyped (as “ArrorMode”), which signify the difference in purpose between the state machines.

The plug-in will only use the “ErrorMode” state machines in the model transformation and will ignore the state machines which with stereotypes other than “ErrorMode” or no stereotypes at all.




nav_controller_stmch
initialising
idle
At "time_tick = after 200 msec"
gathering_navigation_data
updating_nav_report
nav_report_consumer
providing_nav_report
nav_report


adsb_controller_stmch
init
initialisation
fork
adsb_manager
raw_msg_received
listening
idle
decode_msg
[invalid_msg]
At "time_tick = after 1sec"
decoding
managing_adsb_information
[report_no_needed]
merge_info
merging_information
assemble_report
assembling_report
provide_report
providing_report
adsb_reports

A1.4. Cyber-attacks as sequence diagrams

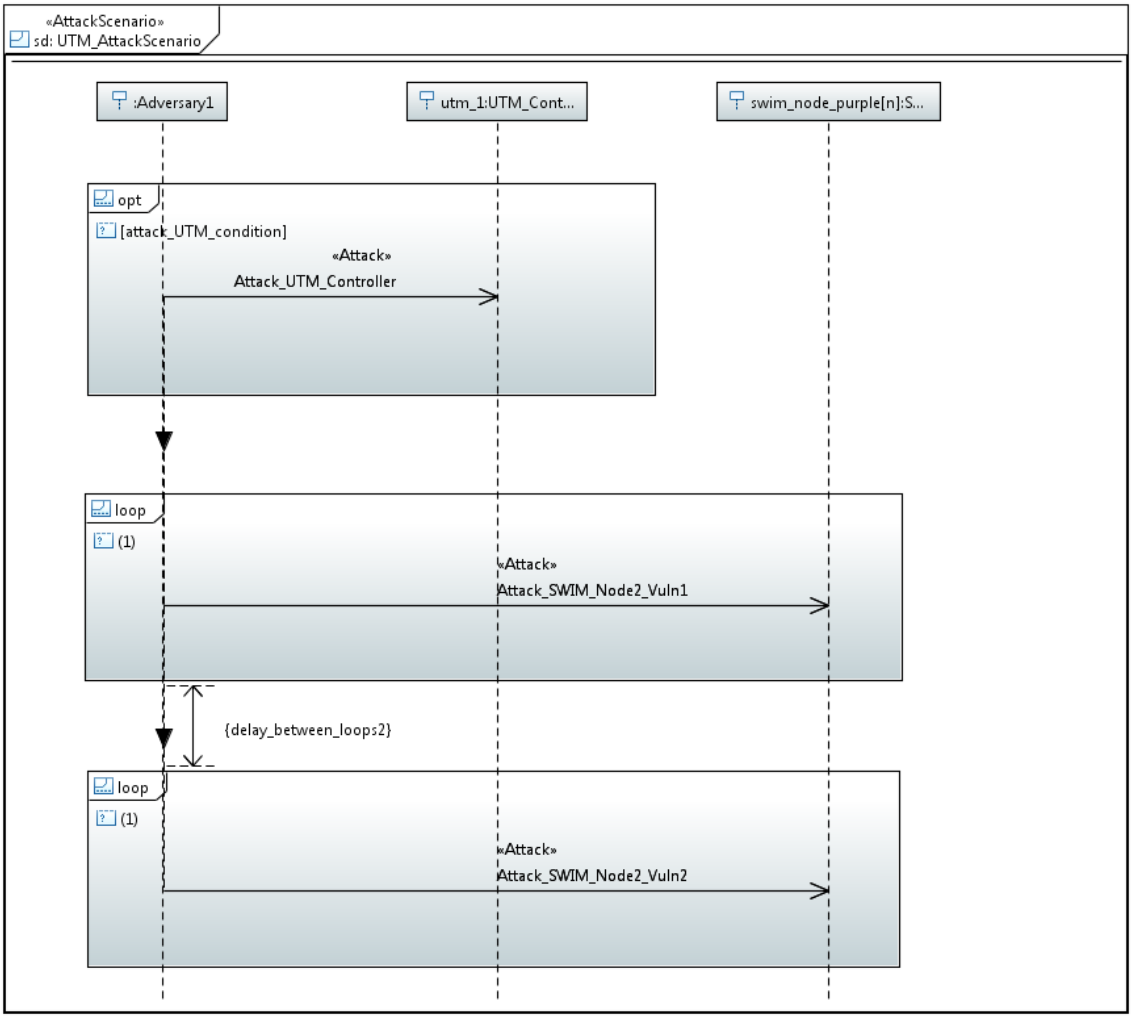
Sequence diagrams are used to model the steps of a cyber-attacks which may lead to a compromise of some of the software components.

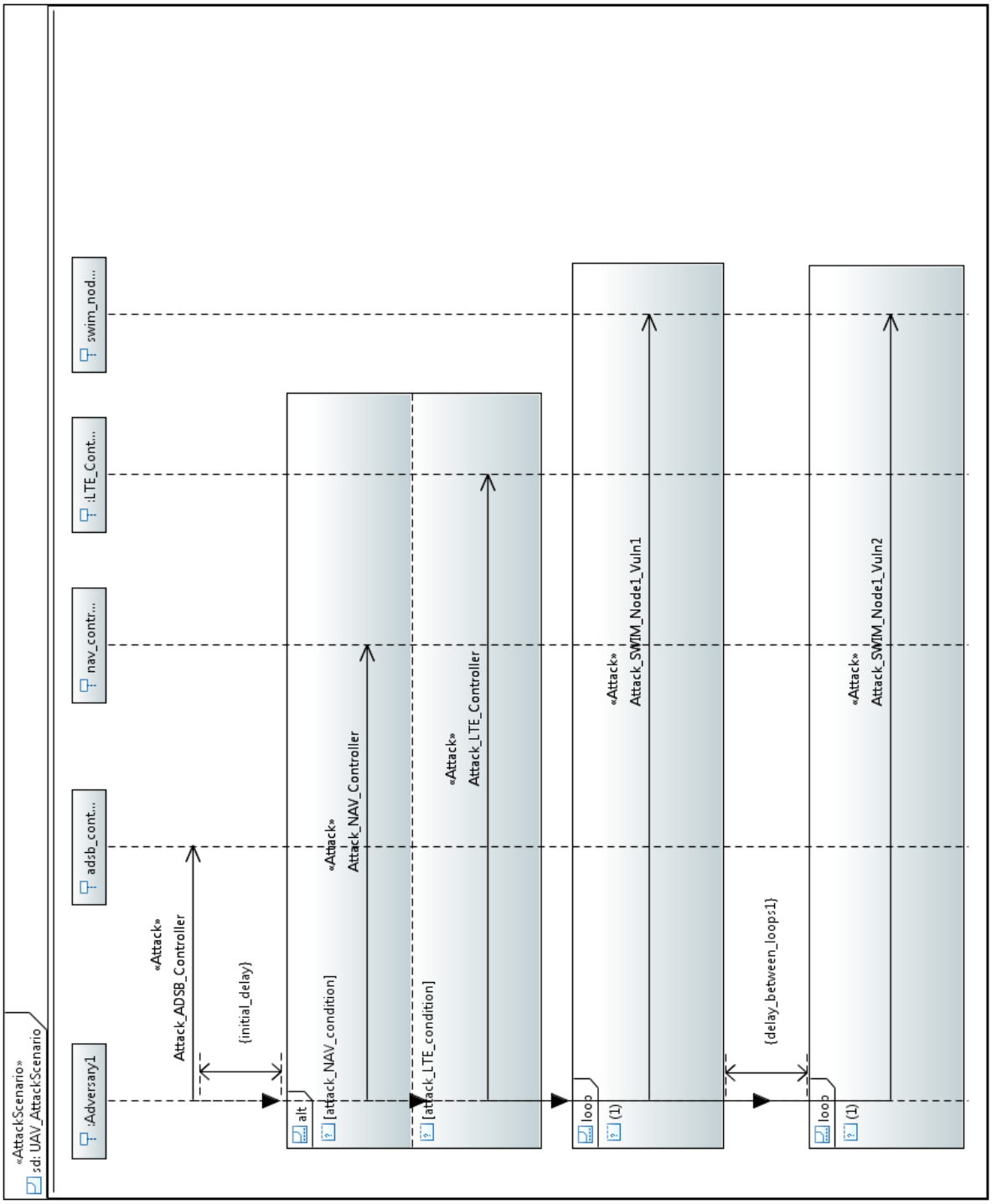
A few examples are given below to illustrate how sequence diagrams can be used to model cyber attacks.

The sequence diagrams are stereotyped as “AttackScenario” to signify the fact that these diagrams represent not a functional interaction (i.e. a realisation of use cases) between the lifelines (the normal use of sequence diagrams in UML/SysML), but model malicious interaction (i.e. “misuse” cases).

The messages represents an attempt to exploit vulnerabilities defined for the corresponding block. In case there are more than one vulnerability per software block then the message should refer explicitly to the vulnerability it targets (as in next diagram: the messages “Attack_SWIM_Node2_X”, where X is either Vuln1 or Vuln2), which refer to the each of the two vulnerabilities defined for the component “SWIM_Node2”.

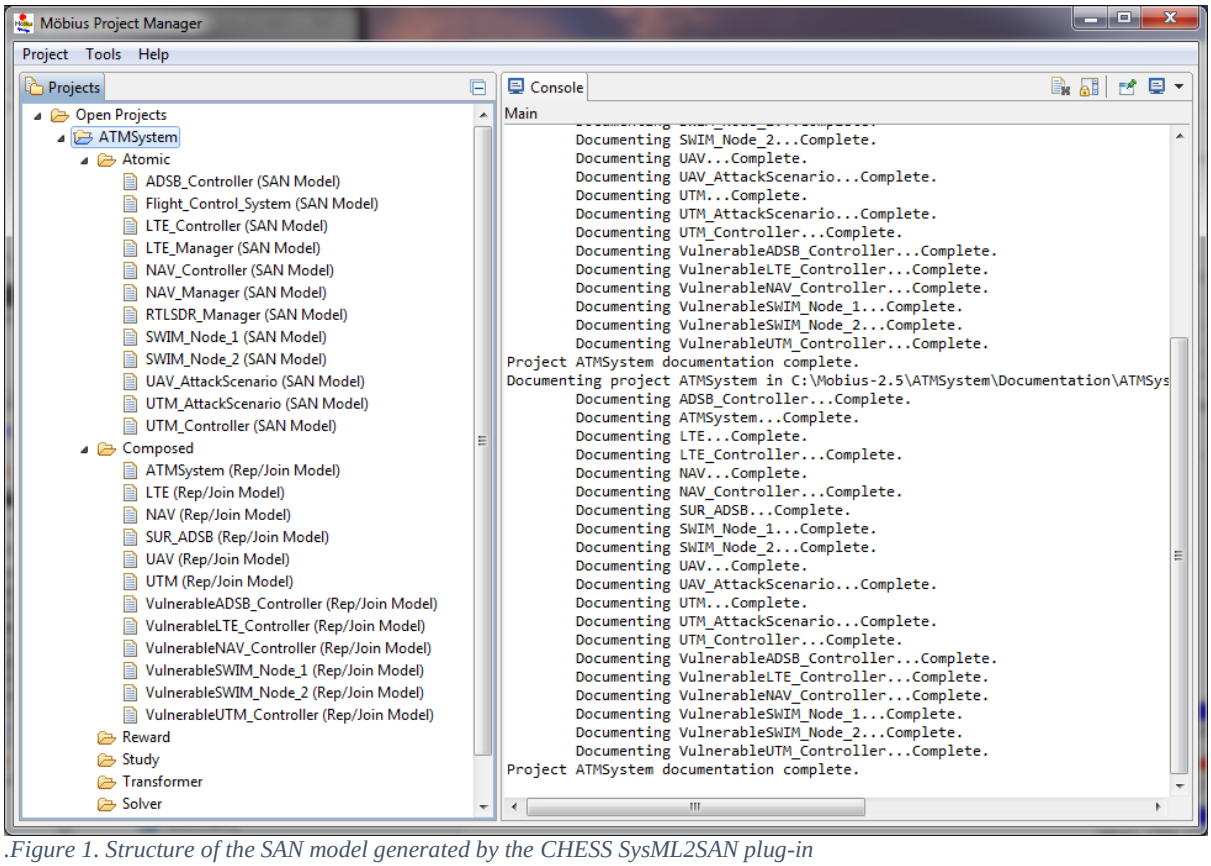
Apart from the stereotype and the details provided above, the syntax of sequence diagrams is the same as in UML. The lifelines represent instances of the blocks captured in a block diagram, combined fragments can model the logic (branching, loops, etc.)




«AttackScenario»
sd: UAV_AttackScenario
:Adversary1
adsb_cont...
nav_contr...
:LTE_Cont...
swim_nod...
«Attack»
Attack_ADSB_Controller
{initial_delay}
alt
[attack_NAV_condition]
«Attack»
Attack_NAV_Controller
[attack_LTE_condition]
«Attack»
Attack_LTE_Controller
loop
(1)
«Attack»
Attack_SWIM_Node1_Vuln1
{delay_between_loops:1}
loop
(1)
«Attack»
Attack_SWIM_Node1_Vuln2

Appendix 2: The SAN model created by the CHESS plugin

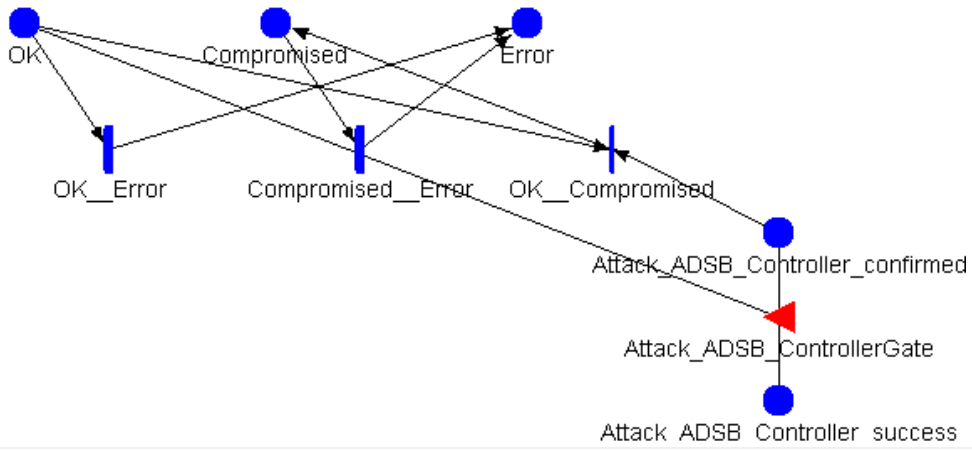
The structure of the SAN model, generated by the plug-in is shown on



.Figure 1. Structure of the SAN model generated by the CHESS SysML2SAN plug-in

Each of the models in the project is detailed below. .

A2.1. Model: ADSB_Controller



Place Attributes:	
Place Names	**Initial Markings**

Attack_ADSB_Controller_confirmed	0
Attack_ADSB_Controller_success	0
Compromised	0
Error	0
OK	1

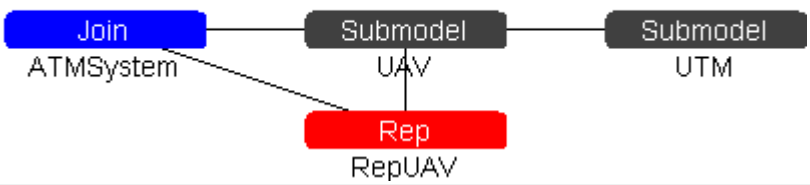
Timed Activity:	Compromised__Error
Distribution Parameters	Rate 0.01
Activation Predicate	(none)
Reactivation Predicate	(none)

Timed Activity:	OK__Error
Distribution Parameters	Rate 0.001
Activation Predicate	(none)
Reactivation Predicate	(none)

Instantaneous Activities Without Cases:
OK__Compromised

Input Gate:	Attack_ADSB_ControllerGate
Predicate	(Attack_ADSB_Controller_success->Mark() > 0) && (OK->Mark() > 0) && (Attack_ADSB_Controller_confirmed->Mark() == 0))
Function	Attack_ADSB_Controller_confirmed->Mark() = 1;

A2.2. Model: ATMSystem

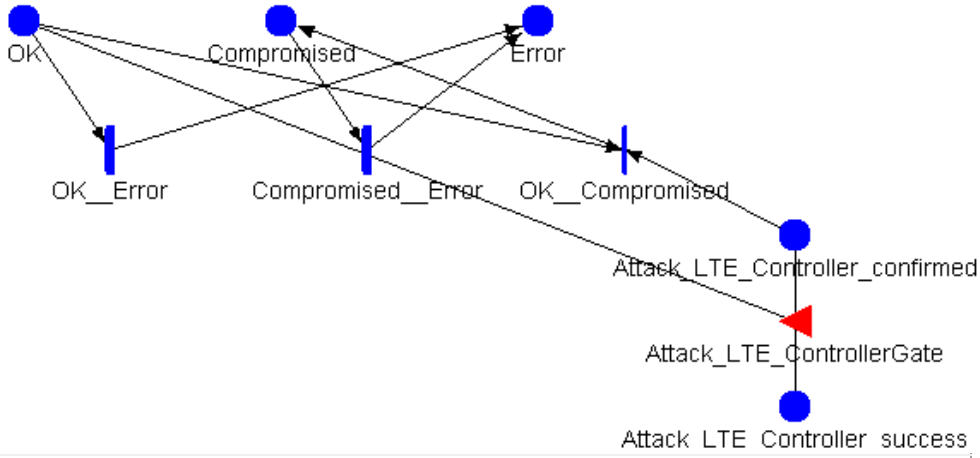


Rep Node	Reps	Shared State Variables

RepUAV	NumUAV	-

A2.3. Model: LTE

A2.4. Model: LTE_Controller



Place Attributes:	
Place Names	**Initial Markings**
Attack_LTE_Controller_confirmed	0
Attack_LTE_Controller_success	0
Compromised	0
Error	0
OK	1

Timed Activity:	**Compromised__Error**
Distribution Parameters	**Rate** 0.01
Activation Predicate	(none)
Reactivation Predicate	(none)

Timed Activity:	**OK__Error**

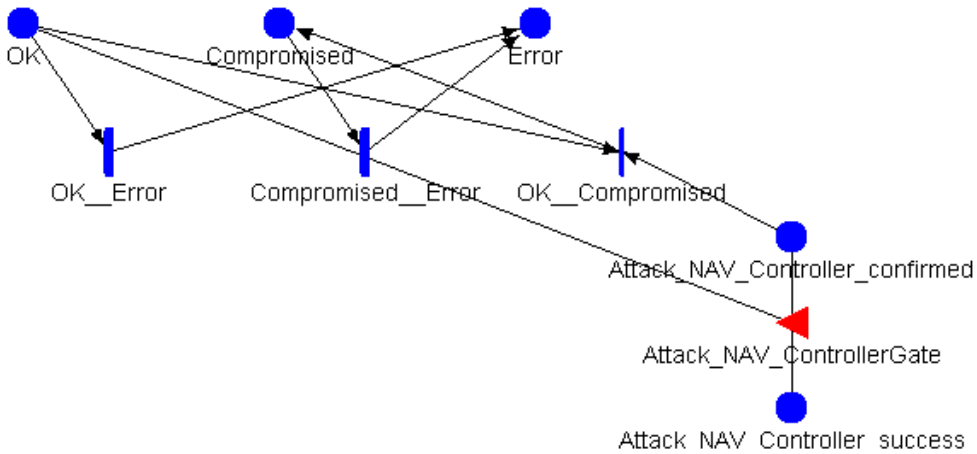
Distribution Parameters	**Rate** 0.001
Activation Predicate	(none)
Reactivation Predicate	(none)

Instantaneous Activities Without Cases:
OK__Compromised

Input Gate:	**Attack_LTE_ControllerGate**
Predicate	(Attack_LTE_Controller_success->Mark() > 0) && (OK->Mark() > 0) && (Attack_LTE_Controller_confirmed->Mark() == 0))
Function	Attack_LTE_Controller_confirmed->Mark() = 1;

A2.5. Model: NAV

A2.6. Model: NAV_Controller



Place Attributes:	
Place Names	**Initial Markings**
Attack_NAV_Controller_confirmed	0

Attack_NAV_Controller_success	0
Compromised	0
Error	0
OK	1

Timed Activity:	**Compromised__Error**
Distribution Parameters	**Rate** 0.01
Activation Predicate	(none)
Reactivation Predicate	(none)

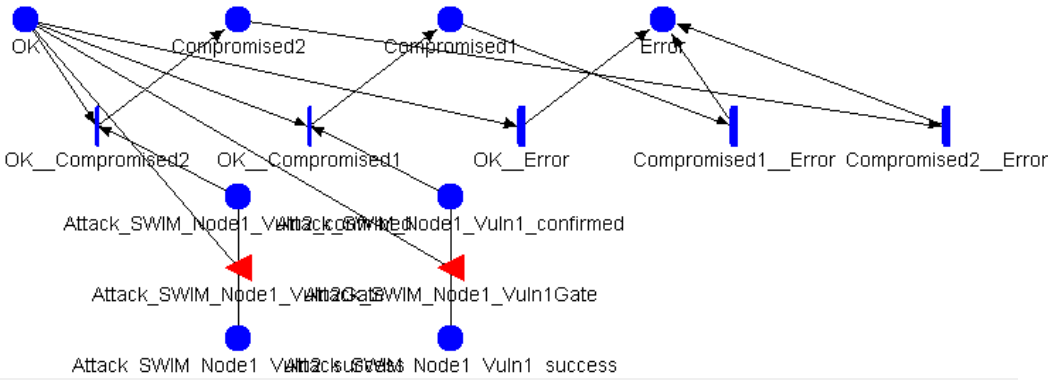
Timed Activity:	**OK__Error**
Distribution Parameters	**Rate** 0.001
Activation Predicate	(none)
Reactivation Predicate	(none)

Instantaneous Activities Without Cases:
OK__Compromised

Input Gate:	**Attack_NAV_ControllerGate**
Predicate	(Attack_NAV_Controller_success->Mark() > 0) && (OK->Mark() > 0) && (Attack_NAV_Controller_confirmed->Mark() == 0))
Function	Attack_NAV_Controller_confirmed->Mark() = 1;

A2.7. Model: SUR_ADSB

A2.8. Model: SWIM_Node_1



Place Attributes:

Place Names	Initial Markings
Attack_SWIM_Node1_Vuln1_confirmed	0
Attack_SWIM_Node1_Vuln1_success	0
Attack_SWIM_Node1_Vuln2_confirmed	0
Attack_SWIM_Node1_Vuln2_success	0
Compromised1	0
Compromised2	0
Error	0
OK	1

Timed Activity:	**Compromised1__Error**
Distribution Parameters	**Rate** Compromised1__ErrorRate
Activation Predicate	(none)
Reactivation Predicate	(none)

Timed Activity:	**Compromised2__Error**

Distribution Parameters	**Rate** Compromised2__ErrorRate
Activation Predicate	(none)
Reactivation Predicate	(none)

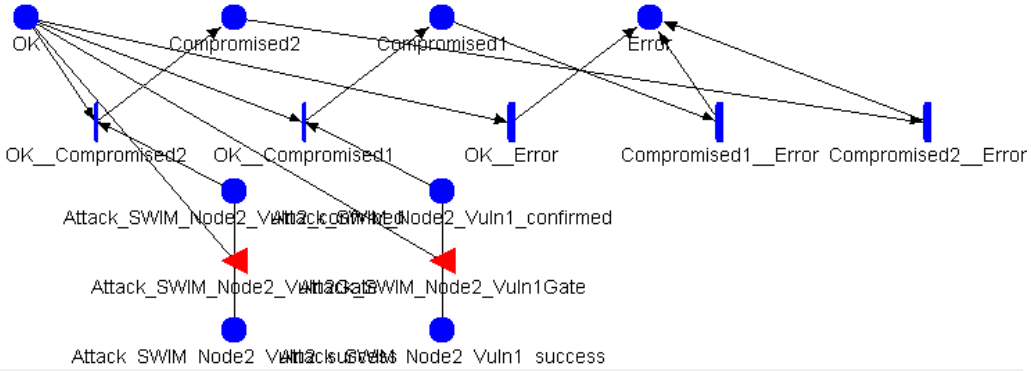
Timed Activity:	**OK__Error**
Distribution Parameters	**Rate** 0.001
Activation Predicate	(none)
Reactivation Predicate	(none)

Instantaneous Activities Without Cases:
OK__Compromised1
OK__Compromised2

Input Gate:	**Attack_SWIM_Node1_Vuln1Gate**
Predicate	(Attack_SWIM_Node1_Vuln1_success->Mark() > 0) && (OK->Mark() > 0) && (Attack_SWIM_Node1_Vuln1_confirmed->Mark() == 0))
Function	Attack_SWIM_Node1_Vuln1_confirmed->Mark() = 1;

Input Gate:	**Attack_SWIM_Node1_Vuln2Gate**
Predicate	(Attack_SWIM_Node1_Vuln2_success->Mark() > 0) && (OK->Mark() > 0) && (Attack_SWIM_Node1_Vuln2_confirmed->Mark() == 0))
Function	Attack_SWIM_Node1_Vuln2_confirmed->Mark() = 1;

A2.9. Model: SWIM_Node_2



Place Attributes:	
Place Names	**Initial Markings**
Attack_SWIM_Node2_Vuln1_confirmed	0
Attack_SWIM_Node2_Vuln1_success	0
Attack_SWIM_Node2_Vuln2_confirmed	0
Attack_SWIM_Node2_Vuln2_success	0
Compromised1	0
Compromised2	0
Error	0
OK	1

Timed Activity:	**Compromised1__Error**
Distribution Parameters	**Rate** Compromised1__ErrorRate
Activation Predicate	(none)
Reactivation Predicate	(none)

Timed Activity:	**Compromised2__Error**
Distribution Parameters	**Rate** Compromised2__ErrorRate
Activation Predicate	(none)
Reactivation Predicate	(none)

Timed Activity:	OK__Error
Distribution Parameters	Rate 0.001
Activation Predicate	(none)
Reactivation Predicate	(none)

Instantaneous Activities Without Cases:
OK__Compromised1
OK__Compromised2

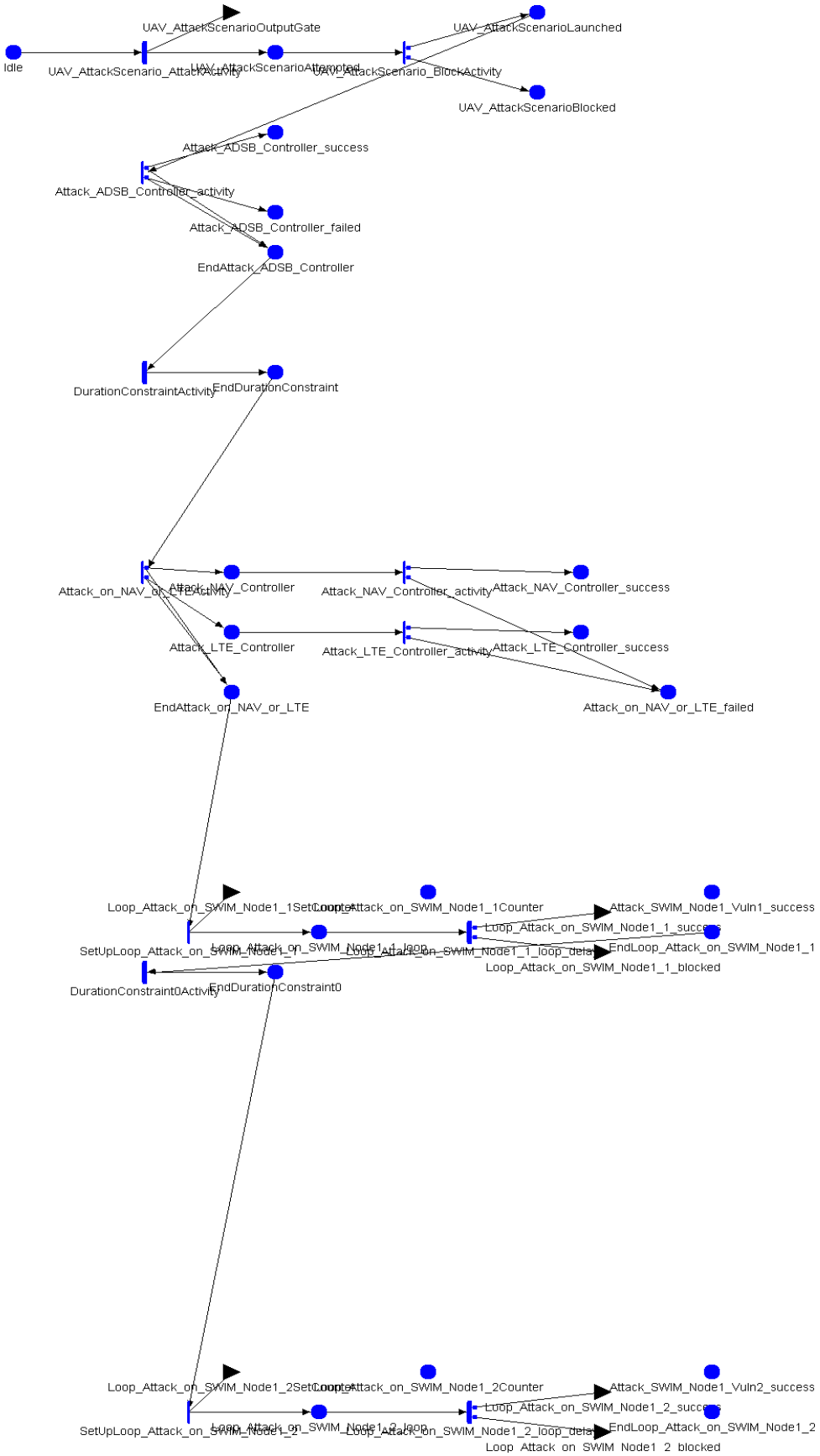
Input Gate:	Attack_SWIM_Node2_Vuln1Gate
Predicate	(Attack_SWIM_Node2_Vuln1_success->Mark() > 0) && (OK->Mark() > 0) && (Attack_SWIM_Node2_Vuln1_confirmed->Mark() == 0))
Function	Attack_SWIM_Node2_Vuln1_confirmed->Mark() = 1;

Input Gate:	Attack_SWIM_Node2_Vuln2Gate
Predicate	(Attack_SWIM_Node2_Vuln2_success->Mark() > 0) && (OK->Mark() > 0) && (Attack_SWIM_Node2_Vuln2_confirmed->Mark() == 0))
Function	Attack_SWIM_Node2_Vuln2_confirmed->Mark() = 1;

A2.10 Model: UAV

Rep Node	Reps	Shared State Variables
RepSWIM_Node_1	NumSWIM_Node_1	-

A2.11. Model: UAV_AttackScenario



Place Attributes:	
Place Names	**Initial Markings**
Attack_ADSB_Controller_failed	0
Attack_ADSB_Controller_success	0
Attack_LTE_Controller	0
Attack_LTE_Controller_success	0
Attack_NAV_Controller	0
Attack_NAV_Controller_success	0
Attack_SWIM_Node1_Vuln1_success	0
Attack_SWIM_Node1_Vuln2_success	0
Attack_on_NAV_or_LTE_failed	0
EndAttack_ADSB_Controller	0
EndAttack_on_NAV_or_LTE	0
EndDurationConstraint	0
EndDurationConstraint0	0
EndLoop_Attack_on_SWIM_Node1_1	0
EndLoop_Attack_on_SWIM_Node1_2	0
Idle	1
Loop_Attack_on_SWIM_Node1_1Counter	0
Loop_Attack_on_SWIM_Node1_1_loop	0
Loop_Attack_on_SWIM_Node1_2Counter	0
Loop_Attack_on_SWIM_Node1_2_loop	0
UAV_AttackScenarioAttempted	0
UAV_AttackScenarioBlocked	0
UAV_AttackScenarioLaunched	0

Timed Activity:	**DurationConstraint0Activity**
Distribution Parameters	**Rate** `return(delay_between_loops1);`

Activation Predicate	(none)
Reactivation Predicate	(none)

Timed Activity:	**DurationConstraintActivity**
Distribution Parameters	**Rate** `return(initial_delay);`
Activation Predicate	(none)
Reactivation Predicate	(none)

Timed Activity:	**Loop_Attack_on_SWIM_Node1_1_loop_delay**
Distribution Parameters	**Rate** `return(Loop_Attack_on_SWIM_Node1_1_attack_delay);`
Activation Predicate	(none)
Reactivation Predicate	(none)

Case Distributions	**case 1**
	`return(Attack_SWIM_Node1_Vuln1_success_prob); // Attack succeeds`
	case 2
	`return(1.0 - Attack_SWIM_Node1_Vuln1_success_prob);  // Attack fails`

Timed Activity:	**Loop_Attack_on_SWIM_Node1_2_loop_delay**
Distribution Parameters	**Rate** `return(Loop_Attack_on_SWIM_Node1_2_attack_delay);`
Activation Predicate	(none)
Reactivation Predicate	(none)

Case Distributions	**case 1**
	`return(Attack_SWIM_Node1_Vuln2_success_prob); // Attack succeeds`
	case 2
	`return(1.0 - Attack_SWIM_Node1_Vuln2_success_prob);  // Attack fails`

Timed Activity:	UAV_AttackScenario_AttackActivity
Distribution Parameters	**Rate** `return(UAV_AttackScenario_intensity);`
Activation Predicate	(none)
Reactivation Predicate	(none)

Instantaneous Activity:	Attack_ADSB_Controller_activity

Case Distributions	**case 1**
	`return(Attack_ADSB_Controller_success_prob); // Attack succeeds`
	case 2
	`return(1.0 - Attack_ADSB_Controller_success_prob); // Attack fails`

Instantaneous Activity:	Attack_LTE_Controller_activity

Case Distributions	**case 1**
	`return(Attack_LTE_Controller_success_prob); // Attack succeeds`
	case 2
	`return(1.0 - Attack_LTE_Controller_success_prob); // Attack fails`

Instantaneous Activity:	Attack_NAV_Controller_activity

Case Distributions	**case 1**
	`return(Attack_NAV_Controller_success_prob); // Attack succeeds`
	case 2
	`return(1.0 - Attack_NAV_Controller_success_prob); // Attack fails`

Instantaneous Activity:	Attack_on_NAV_or_LTEActivity

Case Distributions	**case 1**
	`return(attack_NAV_condition);`
	case 2
	`return(attack_LTE_condition);`

Instantaneous Activity:	UAV_AttackScenario_BlockActivity

Case Distributions	**case 1**
	`return(UAV_AttackScenario_succ_prob);`
	case 2
	`return(1.0 - UAV_AttackScenario_succ_prob); // Attacked blocked.`

Instantaneous Activities Without Cases:
SetUpLoop_Attack_on_SWIM_Node1_1
SetUpLoop_Attack_on_SWIM_Node1_2

Output Gate:	Loop_Attack_on_SWIM_Node1_1SetCounter
Function	`Loop_Attack_on_SWIM_Node1_1Counter->Mark() = N;`

Output Gate:	Loop_Attack_on_SWIM_Node1_1_blocked

Function	`Loop_Attack_on_SWIM_Node1_1Counter->Mark()--;`
	`if(Loop_Attack_on_SWIM_Node1_1Counter->Mark() > 0)`
	`{`
	`// This is the branch which corresponds to unsuccessful attacks on component.`
	`// Hence no token is added to attack place`
	`Loop_Attack_on_SWIM_Node1_1_loop->Mark() = 1; // Continue with the loop`
	`}`
	`else`
	`EndLoop_Attack_on_SWIM_Node1_1->Mark() = 1; // Exit from the loop`

Output Gate:	Loop_Attack_on_SWIM_Node1_1_success

Function	`Loop_Attack_on_SWIM_Node1_1Counter->Mark()--;`
	`if(Loop_Attack_on_SWIM_Node1_1Counter->Mark() > 0)`
	`{`
	`Attack_SWIM_Node1_Vuln1_success->Mark()++; // Add an attack. This place may be shared among many instances.`
	`// The token will be taken by one of the instances: they will compete for the token.`
	`Loop_Attack_on_SWIM_Node1_1_loop->Mark() = 1; // Continue with the loop`
	`}`
	`else`
	`EndLoop_Attack_on_SWIM_Node1_1->Mark() = 1; // Exit from the loop`

Output Gate:	Loop_Attack_on_SWIM_Node1_2SetCounter
Function	Loop_Attack_on_SWIM_Node1_2Counter->Mark() = N;

Output Gate:	Loop_Attack_on_SWIM_Node1_2_blocked

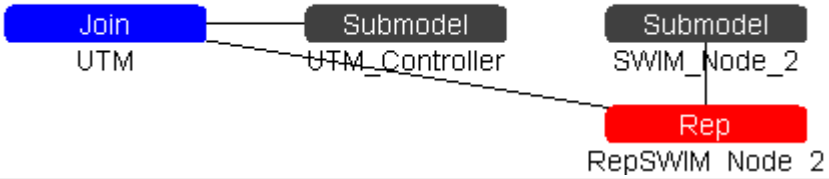
Function	Loop_Attack_on_SWIM_Node1_2Counter->Mark()--;
	if(Loop_Attack_on_SWIM_Node1_2Counter->Mark() > 0)
	{
	// This is the branch which corresponds to unsuccessful attacks on component.
	// Hence no token is added to attack place
	Loop_Attack_on_SWIM_Node1_2_loop->Mark() = 1; // Continue with the loop
	}
	else
	EndLoop_Attack_on_SWIM_Node1_2->Mark() = 1; // Exit from the loop

Output Gate:	Loop_Attack_on_SWIM_Node1_2_success

Function	Loop_Attack_on_SWIM_Node1_2Counter->Mark()--;
	if(Loop_Attack_on_SWIM_Node1_2Counter->Mark() > 0)
	{
	Attack_SWIM_Node1_Vuln2_success->Mark()++; // Add an attack. This place may be shared among many instances.
	// The token will be taken by one of the instances: they will compete for the token.
	Loop_Attack_on_SWIM_Node1_2_loop->Mark() = 1; // Continue with the loop
	}
	else
	EndLoop_Attack_on_SWIM_Node1_2->Mark() = 1; // Exit from the loop

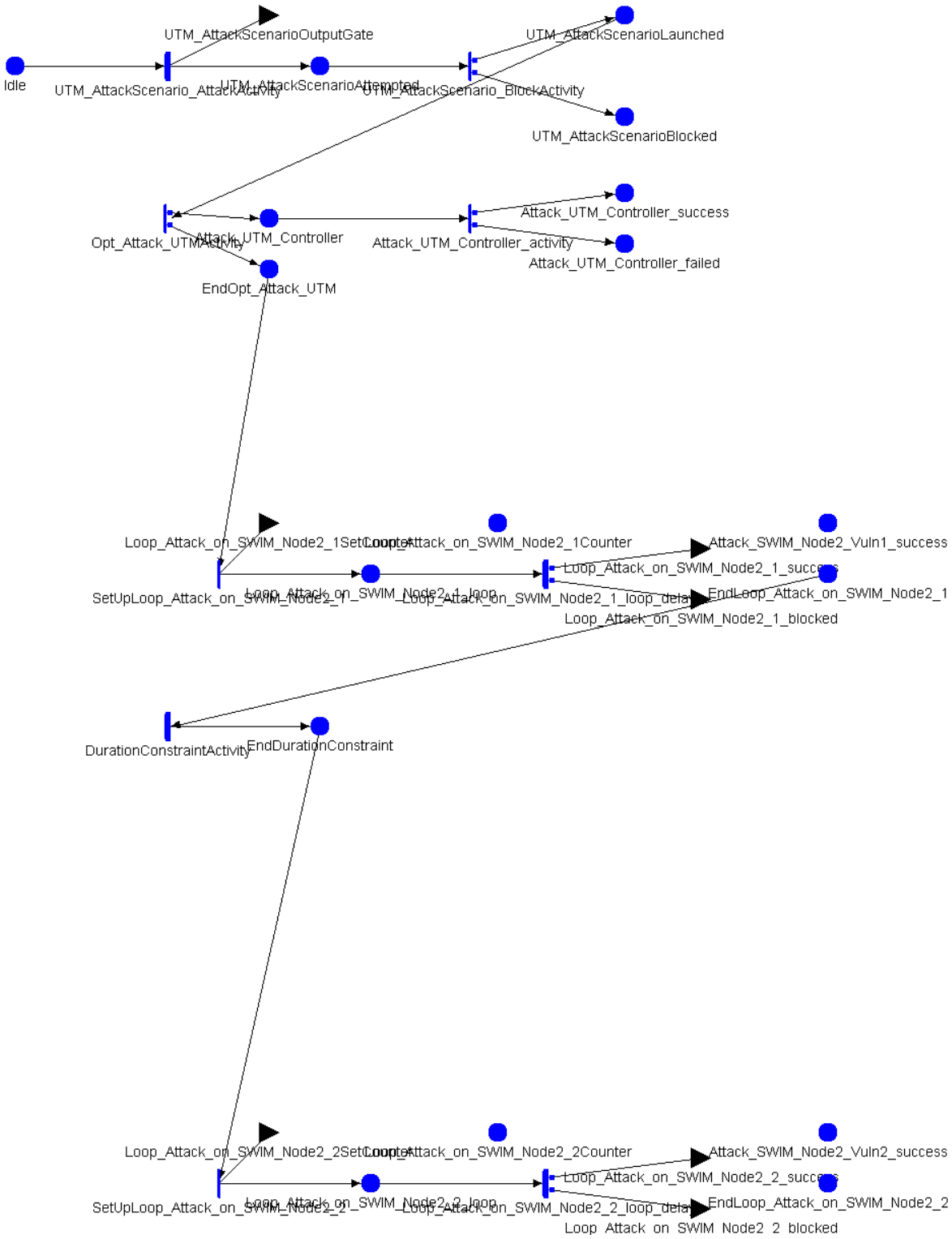
Output Gate:	UAV_AttackScenarioOutputGate
Function	Idle->Mark() = 1; // Implies continuous attacks with given intensity;

A2.12. Model: UTM



Rep Node	Reps	Shared State Variables
RepSWIM_Node_2	NumSWIM_Node_2	-

A2.13. Model: UTM_AttackScenario



Place Attributes:	
Place Names	**Initial Markings**
Attack_SWIM_Node2_Vuln1_success	0

Attack_SWIM_Node2_Vuln2_success	0
Attack_UTM_Controller	0
Attack_UTM_Controller_failed	0
Attack_UTM_Controller_success	0
EndDurationConstraint	0
EndLoop_Attack_on_SWIM_Node2_1	0
EndLoop_Attack_on_SWIM_Node2_2	0
EndOpt_Attack_UTM	0
Idle	1
Loop_Attack_on_SWIM_Node2_1Counter	0
Loop_Attack_on_SWIM_Node2_1_loop	0
Loop_Attack_on_SWIM_Node2_2Counter	0
Loop_Attack_on_SWIM_Node2_2_loop	0
UTM_AttackScenarioAttempted	0
UTM_AttackScenarioBlocked	0
UTM_AttackScenarioLaunched	0

Timed Activity:	**DurationConstraintActivity**
Distribution Parameters	**Rate** `return(delay_between_loops2);`
Activation Predicate	(none)
Reactivation Predicate	(none)

Timed Activity:	**Loop_Attack_on_SWIM_Node2_1_loop_delay**
Distribution Parameters	**Rate** `return(Loop_Attack_on_SWIM_Node2_1_attack_delay);`
Activation Predicate	(none)
Reactivation Predicate	(none)
Case Distributions	**case 1**

	return(Attack_SWIM_Node2_Vuln1_success_prob); // Attack succeeds
	case 2
	
	return(1.0 - Attack_SWIM_Node2_Vuln1_success_prob); // Attack fails

Timed Activity:	**Loop_Attack_on_SWIM_Node2_2_loop_delay**
Distribution Parameters	**Rate** return(Loop_Attack_on_SWIM_Node2_2_attack_delay);
Activation Predicate	(none)
Reactivation Predicate	(none)

Case Distributions	**case 1**
	
	return(Attack_SWIM_Node2_Vuln2_success_prob); // Attack succeeds
	case 2
	
	return(1.0 - Attack_SWIM_Node2_Vuln2_success_prob); // Attack fails

Timed Activity:	**UTM_AttackScenario_AttackActivity**
Distribution Parameters	**Rate** return(UTM_AttackScenario_intensity);
Activation Predicate	(none)
Reactivation Predicate	(none)

Instantaneous Activity:	**Attack_UTM_Controller_activity**

Case Distributions	**case 1**
	
	return(Attack_UTM_Controller_success_prob); // Attack succeeds
	case 2
	
	return(1.0 - Attack_UTM_Controller_success_prob); // Attack fails

Instantaneous Activity:	**Opt_Attack_UTMActivity**
Case Distributions	**case 1** return(attack_UTM_condition);

	case 2 `return(1.0 - attack_UTM_condition);`
Instantaneous Activity:	**UTM_AttackScenario_BlockActivity**

Case Distributions	**case 1**
	
	`return(UTM_AttackScenario_succ_prob);`
	case 2
	
	`return(1.0 - UTM_AttackScenario_succ_prob); // Attacked blocked.`

Instantaneous Activities Without Cases:
SetUpLoop_Attack_on_SWIM_Node2_1
SetUpLoop_Attack_on_SWIM_Node2_2

Output Gate:	**Loop_Attack_on_SWIM_Node2_1SetCounter**
Function	`Loop_Attack_on_SWIM_Node2_1Counter->Mark() = N;`

Output Gate:	**Loop_Attack_on_SWIM_Node2_1_blocked**

Function	`Loop_Attack_on_SWIM_Node2_1Counter->Mark()--;`
	`if(Loop_Attack_on_SWIM_Node2_1Counter->Mark() > 0)`
	`  {`
	`// This is the branch which corresponds to unsuccessful attacks on component.`
	`// Hence no token is added to attack place`
	`  Loop_Attack_on_SWIM_Node2_1_loop->Mark() = 1; // Continue with the loop`
	`  }`
	`else`
	`  EndLoop_Attack_on_SWIM_Node2_1->Mark() = 1; // Exit from the loop`

Output Gate:	**Loop_Attack_on_SWIM_Node2_1_success**

Function	`Loop_Attack_on_SWIM_Node2_1Counter->Mark()--;`
	`if(Loop_Attack_on_SWIM_Node2_1Counter->Mark() > 0)`
	`  {`
	`  Attack_SWIM_Node2_Vuln1_success->Mark()++; // Add an attack. This place may be shared among many instances.`
	`// The token will be taken by one of the instances: they will compete for the token.`
	`  Loop_Attack_on_SWIM_Node2_1_loop->Mark() = 1; // Continue with the loop`
	`  }`
	`else`

	EndLoop_Attack_on_SWIM_Node2_1->Mark() = 1; // Exit from the loop

Output Gate:	Loop_Attack_on_SWIM_Node2_2SetCounter
Function	Loop_Attack_on_SWIM_Node2_2Counter->Mark() = N;

Output Gate:	Loop_Attack_on_SWIM_Node2_2_blocked

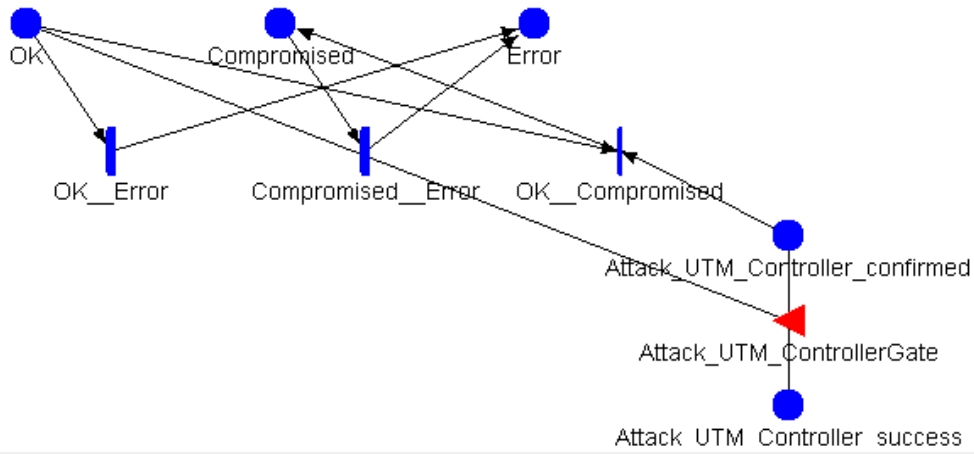
Function	Loop_Attack_on_SWIM_Node2_2Counter->Mark()--;
	if(Loop_Attack_on_SWIM_Node2_2Counter->Mark() > 0)
	{
	// This is the branch which corresponds to unsuccessful attacks on component.
	// Hence no token is added to attack place
	Loop_Attack_on_SWIM_Node2_2_loop->Mark() = 1; // Continue with the loop
	}
	else
	EndLoop_Attack_on_SWIM_Node2_2->Mark() = 1; // Exit from the loop

Output Gate:	Loop_Attack_on_SWIM_Node2_2_success

Function	Loop_Attack_on_SWIM_Node2_2Counter->Mark()--;
	if(Loop_Attack_on_SWIM_Node2_2Counter->Mark() > 0)
	{
	Attack_SWIM_Node2_Vuln2_success->Mark()++; // Add an attack. This place may be shared among many instances.
	// The token will be taken by one of the instances: they will compete for the token.
	Loop_Attack_on_SWIM_Node2_2_loop->Mark() = 1; // Continue with the loop
	}
	else
	EndLoop_Attack_on_SWIM_Node2_2->Mark() = 1; // Exit from the loop

Output Gate:	UTM_AttackScenarioOutputGate
Function	Idle->Mark() = 1; // Implies continuous attacks with given intensity;

A2.14. Model: UTM_Controller



Place Attributes:	
Place Names	**Initial Markings**
Attack_UTM_Controller_confirmed	0
Attack_UTM_Controller_success	0
Compromised	0
Error	0
OK	1

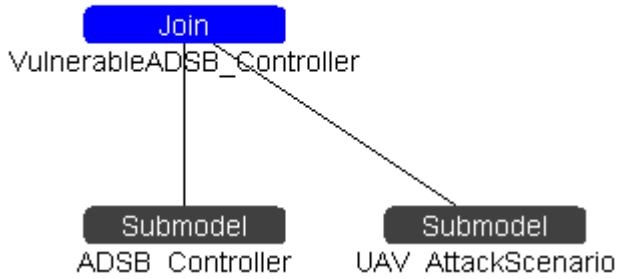
Timed Activity:	**Compromised__Error**
Distribution Parameters	**Rate** 0.01
Activation Predicate	(none)
Reactivation Predicate	(none)

Timed Activity:	**OK__Error**
Distribution Parameters	**Rate** 0.001
Activation Predicate	(none)
Reactivation Predicate	(none)

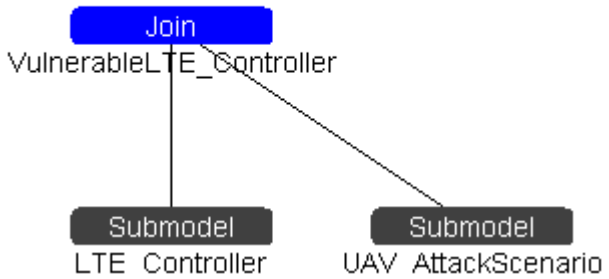
Instantaneous Activities Without Cases:
OK__Compromised

Input Gate:	Attack_UTM_ControllerGate
Predicate	(Attack_UTM_Controller_success->Mark() > 0) && (OK->Mark() > 0) && (Attack_UTM_Controller_confirmed->Mark() == 0))
Function	Attack_UTM_Controller_confirmed->Mark() = 1;

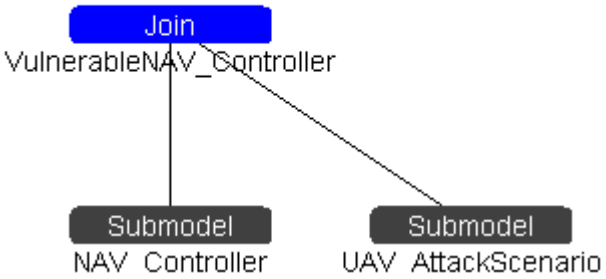
A2.15. Model: VulnerableADSB_Controller



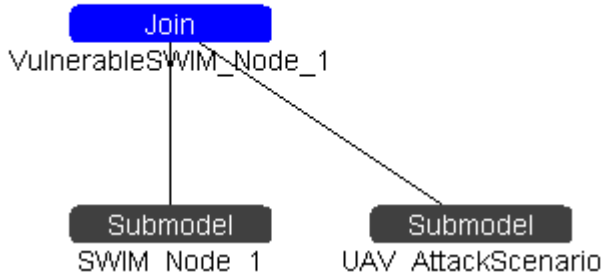
A2.16. Model: VulnerableLTE_Controller



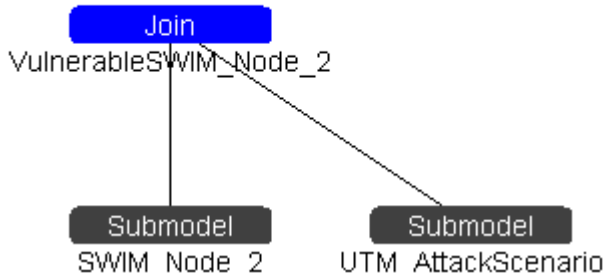
A2.17. Model: VulnerableNAV_Controller



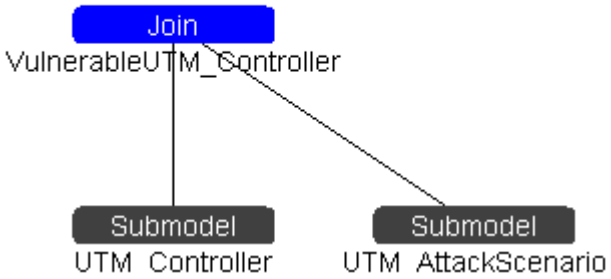
A2.18. Model: VulnerableSWIM_Node_1



A2.19. Model: VulnerableSWIM_Node_2



A2.20. Model: VulnerableUTM_Controller



Created by Peter Popov: 20 June 2019

Last updated: 22 August 2019