CHESS – Mobius Integration Guidelines

Document History

Version	Change
Current version	Update with respect to the delivery of tool version 1.0.0.

Table of Contents

List of Figures

1 Introduction

This document describes the modelling of safety and security concerns in CHESS and the automatic generation of a Mobius Project composed of SAN and REP/JOIN models.

CHESS is a model based methodology and toolset supporting the design and analysis of cyber-physical systems. CHESS relies on the UML and SysML modelling languages, which are largely adopted in the industry, and on ad-hoc and easy to use profile for the modelling and analysis of dependable-related information.

Möbius is a software tool for modelling the behaviour of complex systems which allows studying the reliability, availability, and performance of computer and network systems. A number of reliability analysis results can be obtained with probabilistic models built with Mobius using the stochastic activity networks (SAN) formalism, solved then via Monte-Carlo simulation.
The CHESS-Mobius plugin is conceived to support modeling of fault-tolerant software under cyber-attacks and facilitate the exploiting of the Mobius capabilities for analysis of reliability **Error! Reference source not found.**.

Editing Mobius models can be not trivial. To this purpose we have extended the CHESS profile and therefore the CHESS modelling language capabilities and user-friendly editor as a front-end to fully support the modelling of system architectures taking in consideration safety and security co-engineering, and using automatic transformations to SAN model for reliability analysis with MOBIUS. This approach provides a smooth integration, guarantees the consistency among SysML and SAN models, and drastically reduce the effort required to construct an appropriate SAN analysis model.

The CHESS-Mobius approach can be applied across several domains.

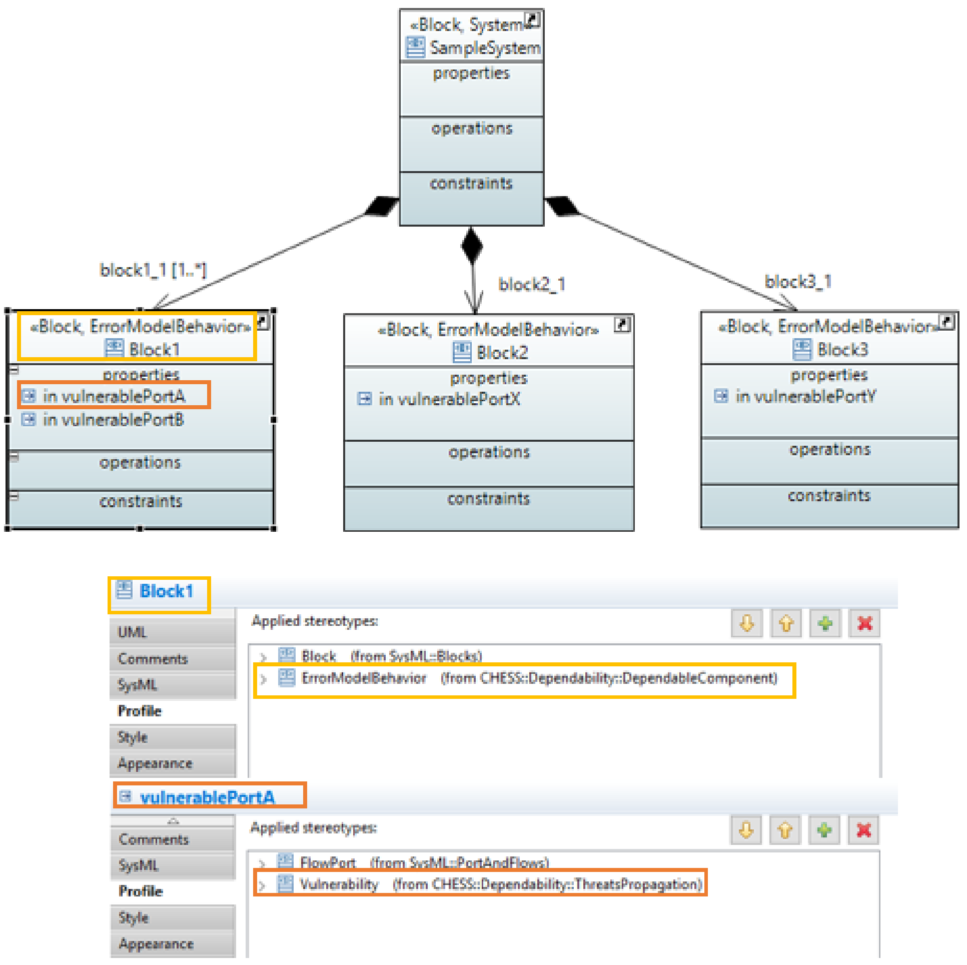
A detailed description of modelling in the CHESS Toolset is out of the scope of this document. Please refer to the “CHESS Toolst Guide".

The description of the basic notions of the CHESS dependability profile is available at https://github.com/montex/CHESS-SBA/wiki/CHESS-SBA-Extensions.

2 Modelling safety and security

2.1 SysML Block Modelling

A Vulnerability is modelled as a UML Port owned by a SysML Block: apply the stereotype <<**Vulnerability**>> on a Port. A block can have multiple vulnerabilities. A block with one or more vulnerabilities should be stereotyped as <<**ErrorModelBehavior**>>

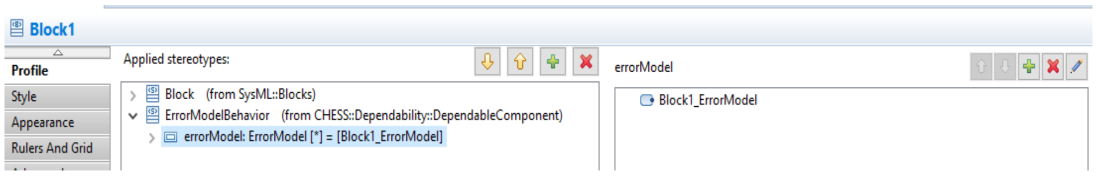


Applied stereotypes:
Block (from SysML::Blocks)
ErrorModelBehavior (from CHESS::Dependability::DependableComponent)

Applied stereotypes:
FlowPort (from SysML::PortAndFlows)
Vulnerability (from CHESS::Dependability::ThreatsPropagation)

Figure 1: Blocks with Vulnerability Ports

The property “errorModel” (of an ErrorModelBehavior stereotype) should be set to a state machine stereotyped as <<**ErrorModel**>> and described in the following section. Only the first Error Model is taken into account when generating the Mobius Project (despite the fact that the “errorModel” property has multiplicity *).



Applied stereotypes:
Block (from SysML::Blocks)
ErrorModelBehavior (from CHESS::Dependability::DependableComponent)
errorModel: ErrorModel [*] = [Block1_ErrorModel]

errorModel
Block1_ErrorModel

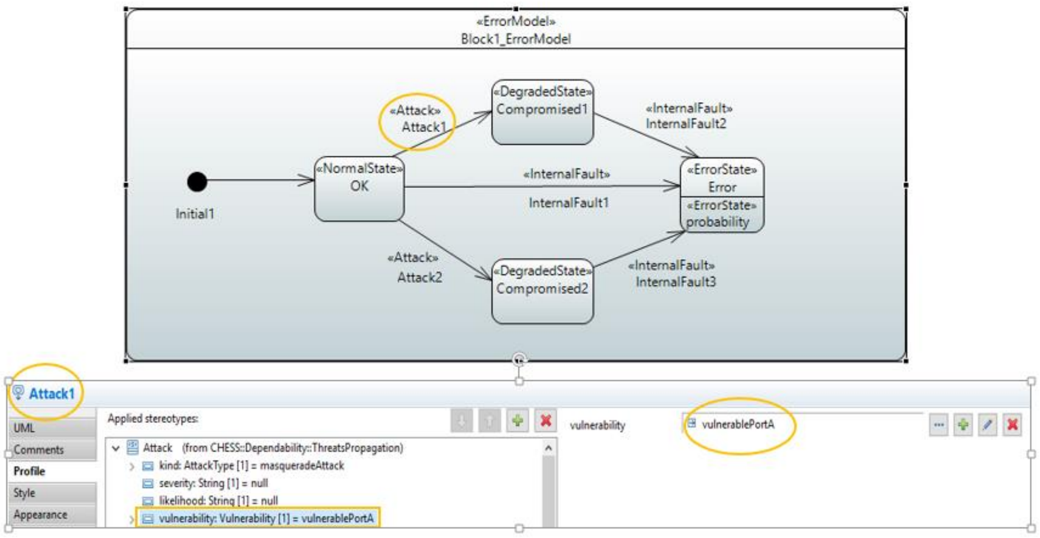
Figure 2: Set Error Model

In a decomposition it makes sense to model vulnerabilities and error models only for blocks which are “leaves” in the decomposition (no further decomposed) as other blocks are just containers.

2.2 Error Model State Machine

1. Create a new state machine diagram in the CHESS model (creates a new UML StateMachine element) as owned by a SysML block stereotyped as <<ErrorModelBehavior>> (see previous section).
2. States of an Error Model state machine should be stereotyped with one of the following:
    a. <<**NormalState**>> only one state should have this stereotype applied and it should be connected with the initial pseudo state. It represents the normal operational conditions of the Block, unaffected by external Attacks (i.e. Healthy)
    b. <<**DegradedState**>> multiple states can have this stereotype applied. They represent the operational conditions of the block affected by a successful external Attack (i.e. Compromised)
    c. <<**ErrorState**>> A state of the component that is considered erroneous, as in [2].
3. Transition of an Error Model state machine should be stereotyped with one of the following:
    a. <<**Attack**>> is used to connect the Healthy state with a Compromised state. It is mandatory to specify the value of the "vulnerability" property. Only a Port stereotyped as <<Vulnerability>> and owned by the enclosing Block is a valid value for this property.
    b. <<**InternalFault**>> is used to connect a Compromised or Healthy state to an Error state. The value of property "occurrence" can be set to model the Rate of the transition to the Error state, if left unspecified it will be represented by a global variable in the corresponding SAN model.

    c. **<<Repair>>** is used to connect and ErrorState to the NormalState. The value of property "probSuccess" can be set to model the Rate of the transition to the Error state, if left unspecified it will be represented by a global variable in the corresponding SAN model.



Attack1

Applied stereotypes:

Attack (from CHESS::Dependability::ThreatsPropagation)
kind: AttackType [1] = masqueradeAttack
severity: String [1] = null
likelihood: String [1] = null
vulnerability: Vulnerability [1] = vulnerablePortA

vulnerability vulnerablePortA

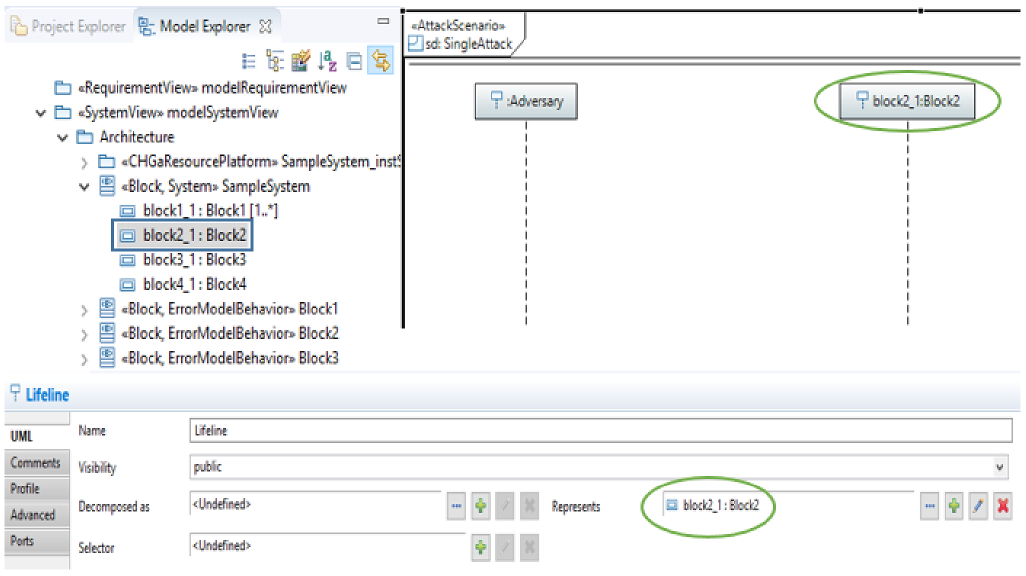
Figure 3: Error Model with Attacks

2.3 Attack Scenarios Modelling

Before modelling attack scenarios it is needed to add an adversary of the system in the model. In this context, an adversary is a UML Actor stereotyped as <<**Adversary**>>.

To model an attack scenario:

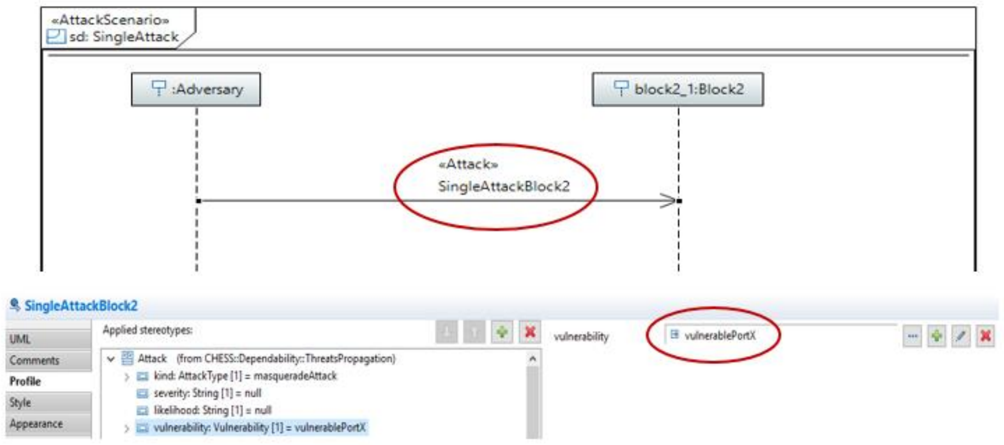
1. Create a new sequence diagram in the CHESS model (creates a new Interaction element)
2. Apply stereotype <<**AttackScenario**>> on the Interaction
   a. Optionally set "frequency" and "probSuccess" properties. If left unspecified -> global variables.
3. Create an anonymous UML Lifeline for the Adversary, and additional Lifelines for each block instance involved in the scenario where for instance here we mean a SysML Part of a composed Block (set the "Represent" property to the corresponding instance in the model)


Project Explorer
Model Explorer
«RequirementView» modelRequirementView
«SystemView» modelSystemView
Architecture
«CHGaResourcePlatform» SampleSystem_inst
«Block, System» SampleSystem
block1_1 : Block1 [1..*]
block2_1 : Block2
block3_1 : Block3
block4_1 : Block4
«Block, ErrorModelBehavior» Block1
«Block, ErrorModelBehavior» Block2
«Block, ErrorModelBehavior» Block3
«AttackScenario»
sd: SingleAttack
:Adversary
block2_1:Block2
Lifeline
UML
Comments
Profile
Advanced
Ports

Name: Lifeline
Visibility: public
Decomposed as: <Undefined>
Represents: block2_1 : Block2
Selector: <Undefined>

Figure 3 - Attack Scenario: Lifelines

4. Add attack fragments to the scenario. A fragment can be one of:
   a. **Single Attack**: a UML Message stereotyped as <<**Attack**>> from the lifeline representing the Adversary to a lifeline of a vulnerable block. It is mandatory to specify the value of the "vulnerability" property. Only a Port stereotyped as <<Vulnerability>> and owned by the representing block is a valid value for this property.



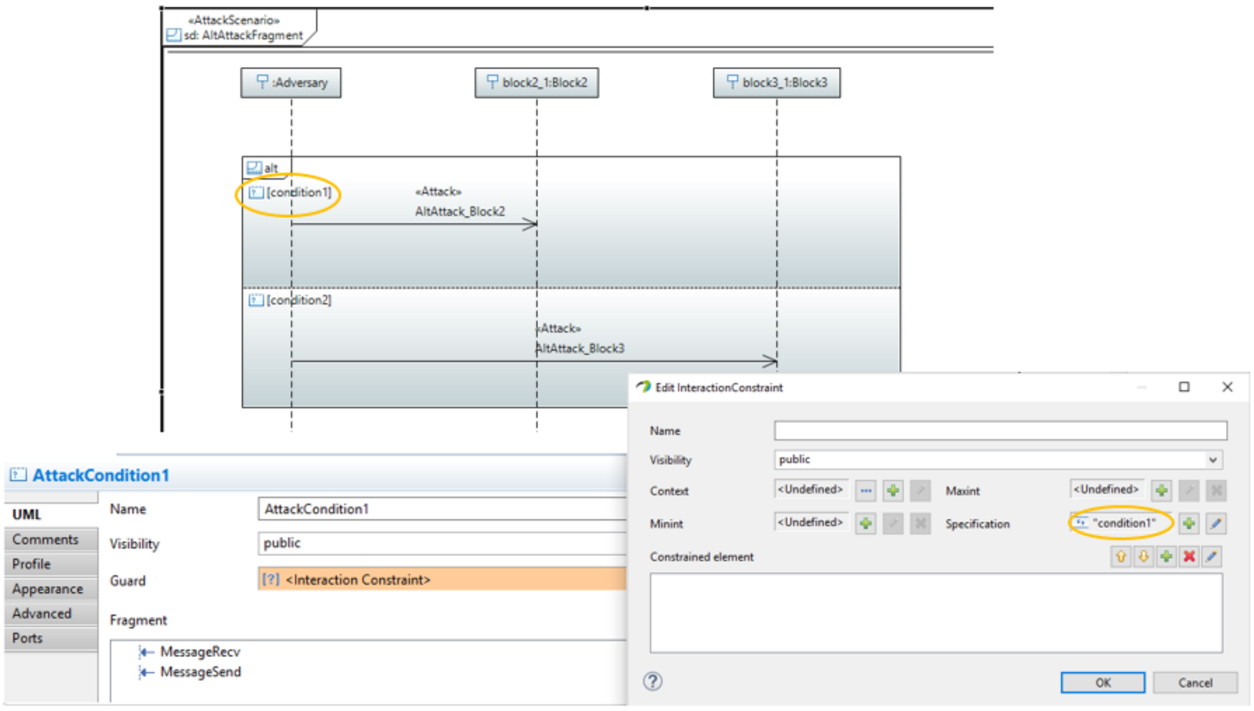
Applied stereotypes:

- Attack (from CHESS::Dependability::ThreatsPropagation)
  - kind: AttackType [1] = masqueradeAttack
  - severity: String [1] = null
  - likelihood: String [1] = null
  - vulnerability: Vulnerability [1] = vulnerablePortX

vulnerability: vulnerablePortX

Figure 4: Attack Fragment: Single Attack

b. **Alternative Attacks**: modelled as a UML CombinedFragment with operator equal to "alt". This CombinedFragment can contain two or more UML elements of type InteractionOperand:
   i. For each InteractionOperand specify its Guard (UML InteractionConstraint): create a UML LiteralString as "specification" of the InteractionConstraint Guard, the value of this LiteralString will become a global variable in the SAN model corresponding to the attack scenario.
   ii. A particular keyword can be used for the LiteralString value: "else". It can be used only once for each Alternative Attacks fragment and it must be used only for the last InteractionOperand.
   iii. Each InteractionOperand will contain exactly one Single Attack fragment (see case a.).



Name	AttackCondition1
Visibility	public
Guard	[?] <Interaction Constraint>

Name			
Visibility	public		
Context	<Undefined>	Maxint	<Undefined>
Minint	<Undefined>	Specification	"condition1"

Figure 5: Attack Fragment: Alternative Attacks

c. **Optional Attack**: modelled as a UML CombinedFragment with operator equal to "opt". This CombinedFragment contains one UML element of type InteractionOperand:
   i. Specify the Guard of the InteractionOperand as for case b.
   ii. "else" keyword cannot be used.
   iii. The InteractionOperand will contain a Single Attack fragment.

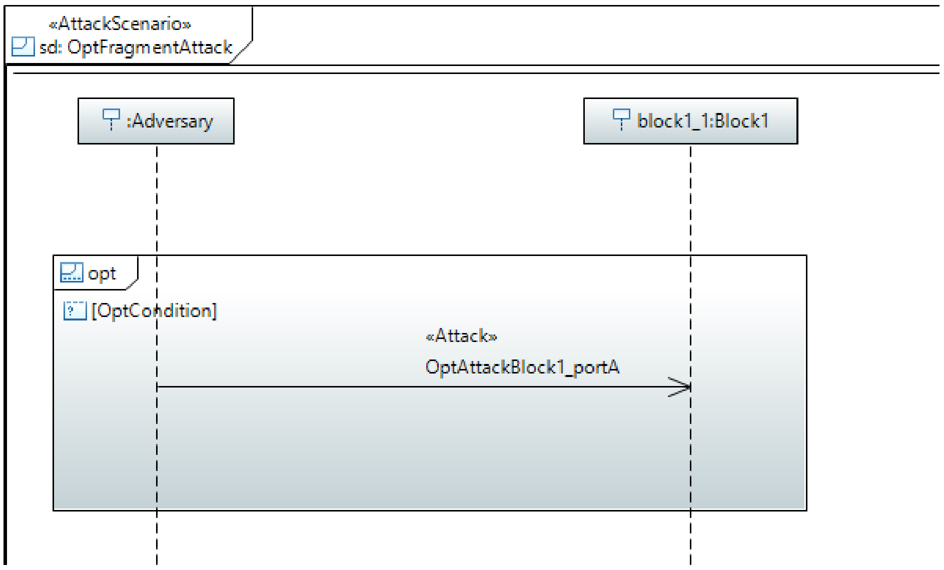
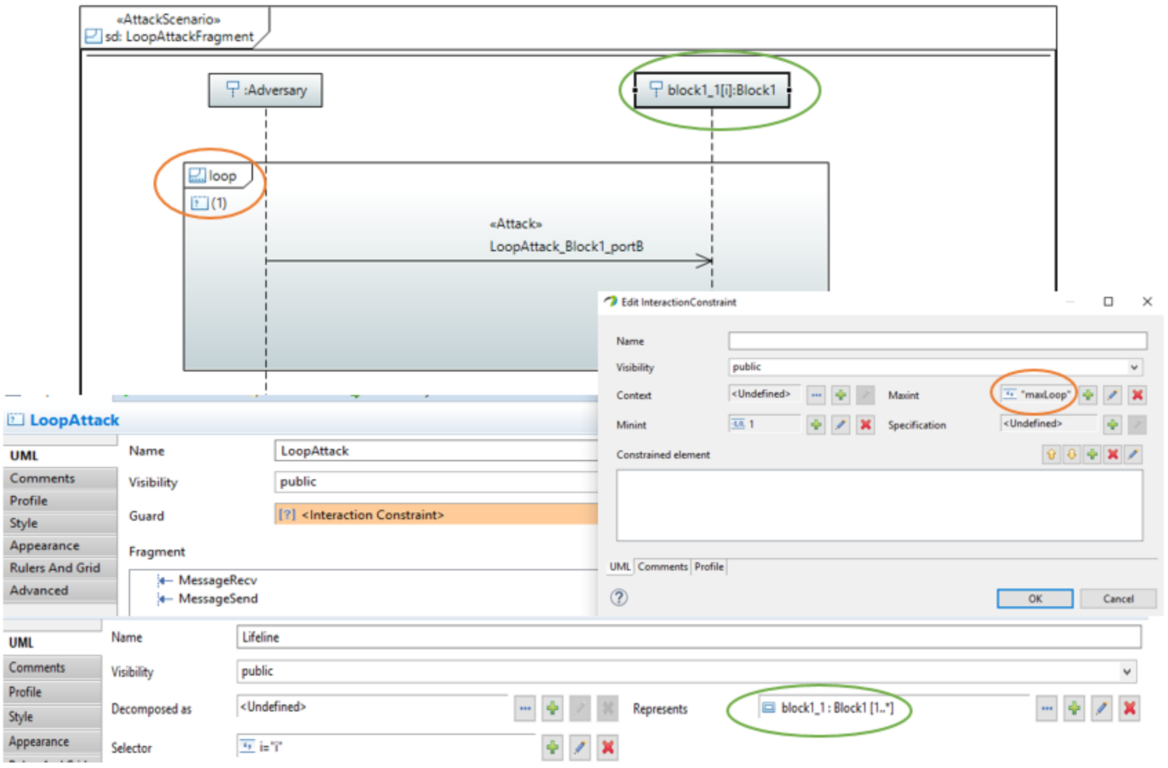


Figure 6: Attack Fragment: Optional Attack

d. **Loop Attack**: the semantics of a Loop Attack is to model a Single Attack repeated for each instance of a vulnerable block. The lifeline target of these attack messages must represent an instance of a block with multiplicity "*" or "1..*". This fragment is modelled as a UML CombinedFragment with operator equal to "loop". This CombinedFragment contains one UML element of type InteractionOperand:
   i. The Guard needs no Specification, instead set its property "maxInt" as UML LiteralInt or UML LiteralString and set their value. If using a LiteralString its value will become a global variable in the SAN model corresponding to the attack scenario.
   ii. The InteractionOperand will contain a Single Attack fragment.



LoopAttack

Name: LoopAttack
Visibility: public
Guard: [?] <Interaction Constraint>
Fragment:
MessageRecv
MessageSend

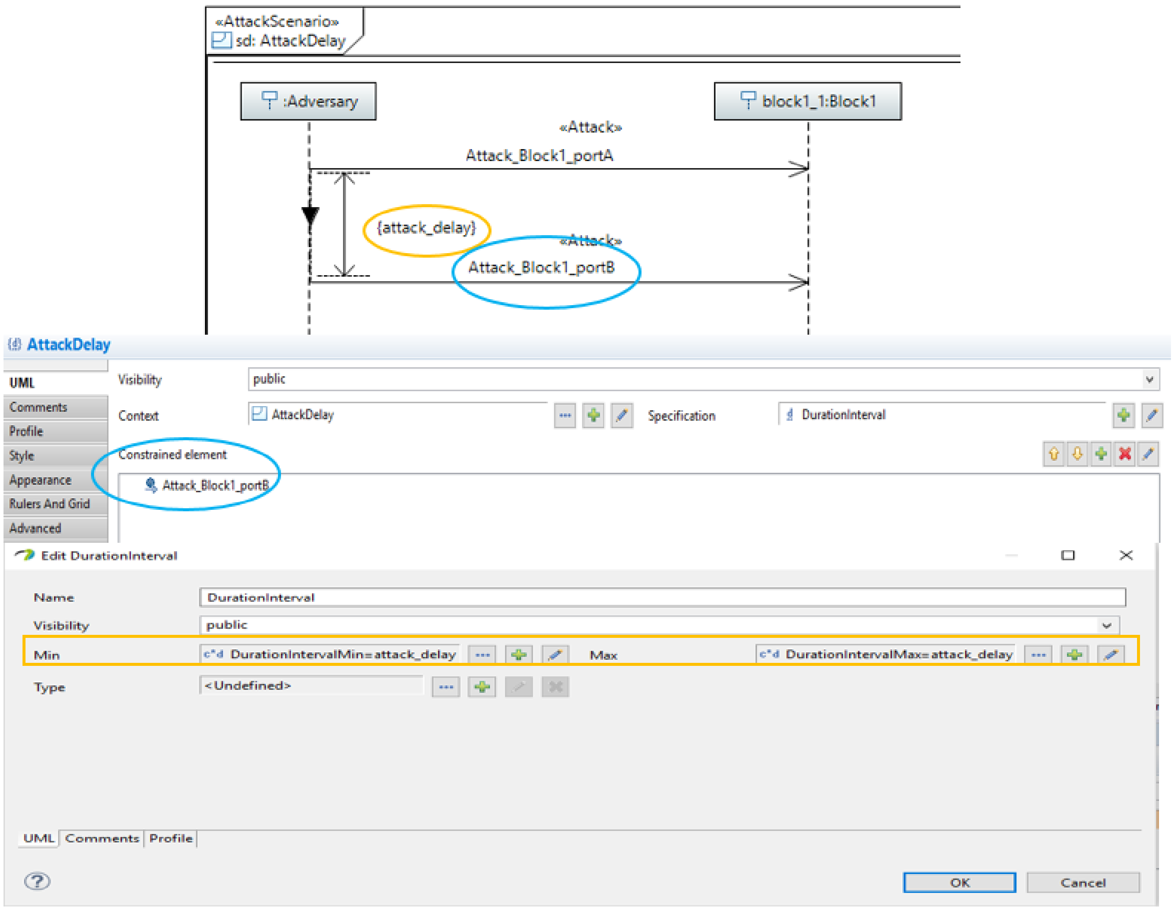
Edit InteractionConstraint

Name:
Visibility: public
Context: <Undefined>
Maxint: "maxLoop"
Minint: 1
Specification: <Undefined>
Constrained element:

Name: Lifeline
Visibility: public
Decomposed as: <Undefined>
Represents: block1_1 : Block1 [1..*]
Selector: i="i"

Figure 7: Attack Fragment: Loop Attack

e. Additionally it is possible to model **delays** between attacks. A delay is a UML DurationConstraint modelled as follows:
   i. Create a UML DurationConstraint in the sequence diagram.
   ii. Specify an attack fragment (described in bullets a. to d.) as constrained element of this DurationConstraint: semantically it is interpreted by the CHESS extended Toolset as a delay that occurs before the constrained element.
   iii. Specify the duration of the delay: check the "specification" of the DurationConstraint: it's a DurationInterval, it's "min" and "max" specify the range of the interval. Set them to the same value using UML LiteralString as expression ("expr") of "min" and "max". This value is again a global variable in the SAN model corresponding to the attack scenario.



Visibility	public
Context	AttackDelay
Specification	DurationInterval
Constrained element	Attack_Block1_portB

Name	DurationInterval
Visibility	public
Min	DurationIntervalMin=attack_delay
Max	DurationIntervalMax=attack_delay
Type	<Undefined>

Figure 8: Attack Fragment: Delay

An Attack Scenario is composed of one or more fragments, to keep the order of the attack messages is highly recommended to use UML GeneralOrdering by connecting each start message (UML MessageSend) with the next one in sequence.

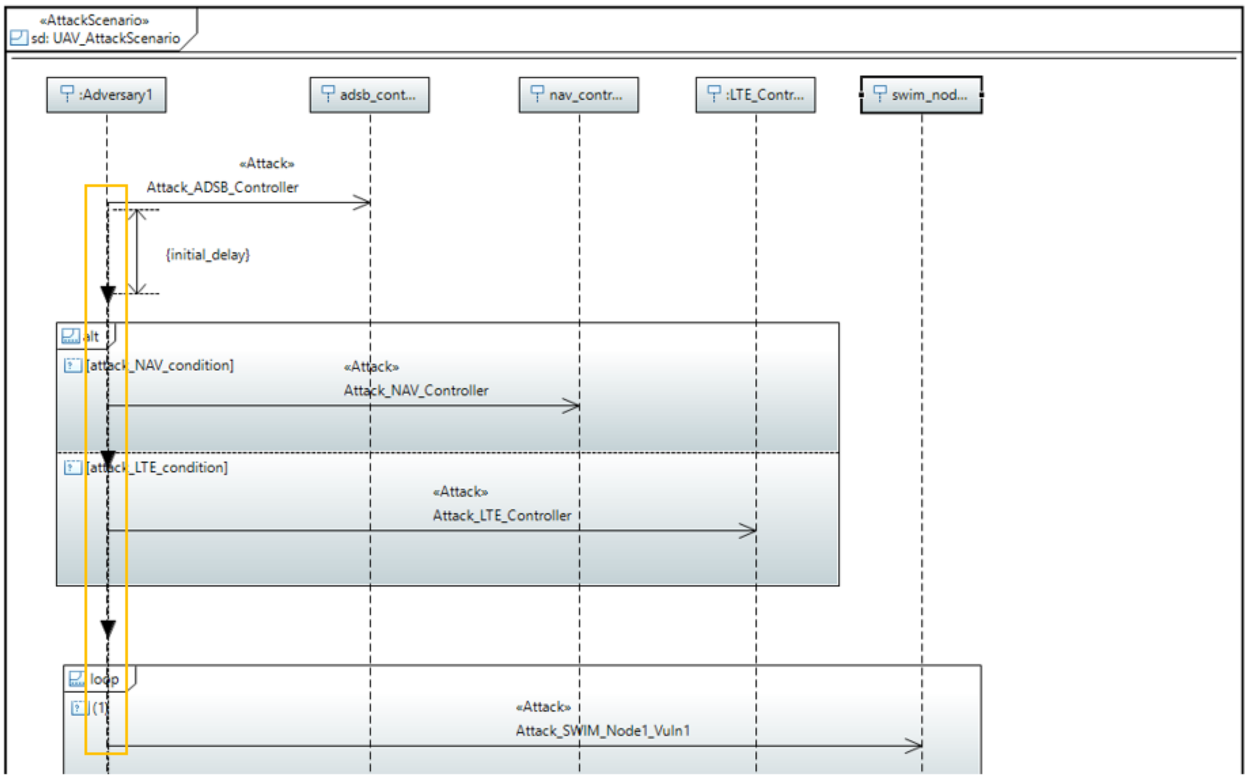


Figure 9: Attack Scenario and GeneralOrdering of Attack Messages

3 More about modelling error behaviour in CHESS

CHESS support for error model behaviour allows to model information about incoming and outgoing failures for a given component; this information if available is translated to SAN during the generation of the Mobius project.

In particular:

- An error propagation originated as a consequence of incoming external fault are represented as *InternalPropagation* transition; the *externalFaults* string attribute is used to specify the incoming failure with the following syntax:

  <input port name>.<failure type>

  while the *delay* numerical attribute is used to model the time after which propagation occurs.

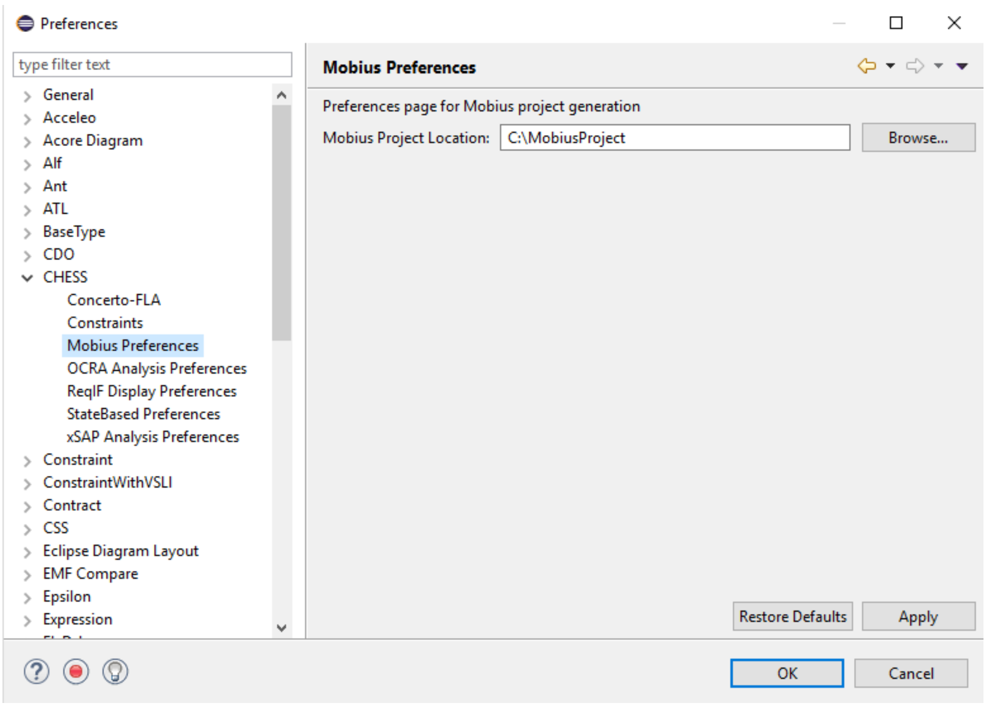
- The failure of a component with propagation of errors to its external ports is modelled with a *Failure* transition the mode string attribute is used to specify the outgoing failures by using the following syntax

  <output port name>.<failure type>

With the aforementioned information, and by having the system architecture described as (possible hierarchical) components instances connected together through their input and output ports, it is possible to then derived scenarios of failures propagation between the components instances.

4 Generate the Mobius Project

It is possible to generate Mobius models for the Cyber Security elements modelled in CHESS. Make sure to set Mobius Projects directory in the preferences page (menu: Window -> Preferences -> CHESS -> Mobius Preferences)



Preferences page for Mobius project generation

Mobius Project Location: C:\MobiusProject

Figure 11: Preference Page

Then follow these steps:

1. Model analysis context for SAN Analysis:
   a. Build components instances model as in [1] section 9.3.1.2. Note the component instance model has to be manually regenerated each time a component changes wrt its decomposition definition (i.e. internal components creation/deletion), ports creation/deletion, connectors creation/deletion. Figure 10 shows an example of instance model generated starting from the definition of the SampleSystem Block. The instance model is generated under a dedicated package (SampleSystem_instSpec) and owns all the information about the components instances, their hierarchical structure, ports and connections.

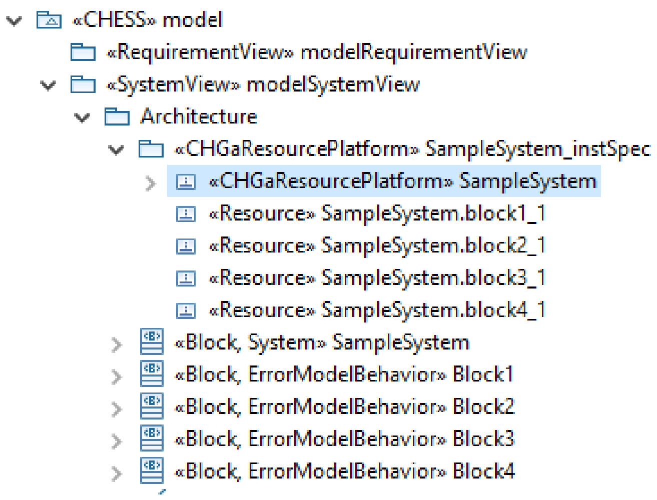
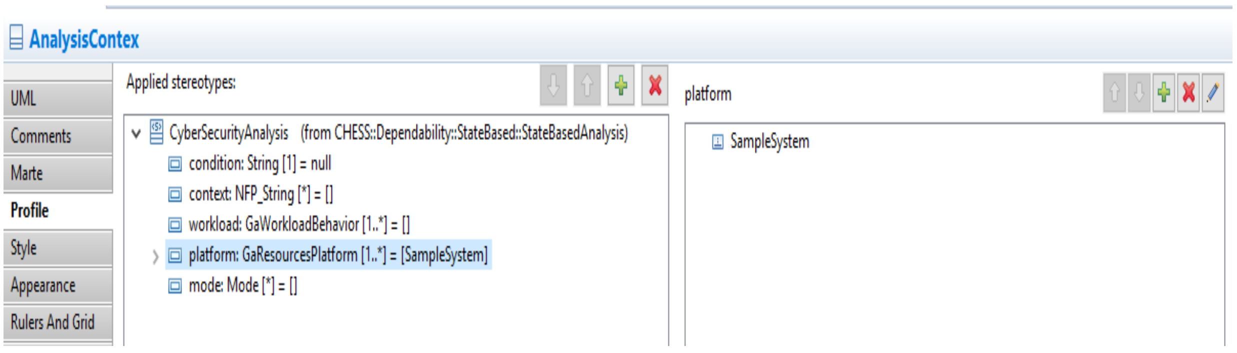

«CHESS» model
«RequirementView» modelRequirementView
«SystemView» modelSystemView
Architecture
«CHGaResourcePlatform» SampleSystem_instSpec
«CHGaResourcePlatform» SampleSystem
«Resource» SampleSystem.block1_1
«Resource» SampleSystem.block2_1
«Resource» SampleSystem.block3_1
«Resource» SampleSystem.block4_1
«Block, System» SampleSystem
«Block, ErrorModelBehavior» Block1
«Block, ErrorModelBehavior» Block2
«Block, ErrorModelBehavior» Block3
«Block, ErrorModelBehavior» Block4

Figure 10: components instances model

b. Create a UML Class stereotyped as <<CyberSecuritySANAnalysis>> in the DependabilityAnalysisView Package. Set the value of “platform” property as the root of the instances (UML InstanceSpecification) generated in step a.


AnalysisContex

Applied stereotypes:
CyberSecurityAnalysis (from CHESS::Dependability::StateBased::StateBasedAnalysis)
condition: String [1] = null
context: NFP_String [*] = []
workload: GaWorkloadBehavior [1..*] = []
platform: GaResourcesPlatform [1..*] = [SampleSystem]
mode: Mode [*] = []

platform
SampleSystem

Figure 11: Analysis Context

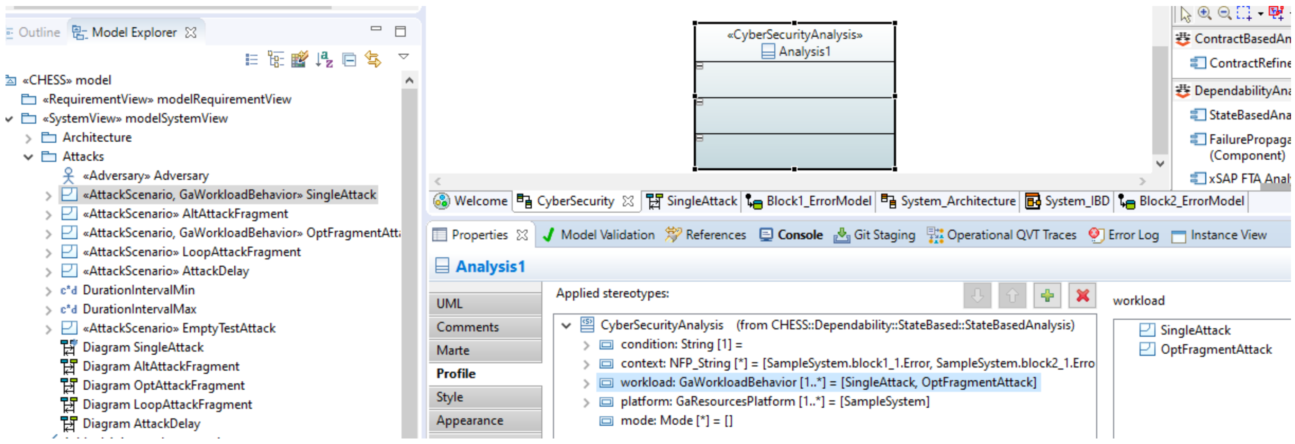
2. Different attack scenarios can be provided in the same CHESS model; it could be that a particular subset of scenarios has to be taken into account for the Mobius project generation, and so analysis. For this goal, in the Cyber Security Analysis ‘workload’ property it is possible to add the scenarios that have to be considered by the transformation and analysis.

   Figure 12 provides an example of ‘workload’ property referring the OptFragmentAttack and SingleAttack scenarios which are located in the SystemView under the Attack folder (the structure of the model is visible on the left side of the picture).

   To be able to assign a scenario to the workload property, the following steps have to be performed:

   - add the registered GQAM MARTE profile to the CHESS SystemView Package, if not still applied; this step has to be done by selecting the SystemView, and using the Profile tab of the Properties View (Figure 13 shows the wizard which appears when selecting the *Applying registered profile* command in the profile tab and which can be used to select and apply the GQAM MARTE profile).

- Add the MARTE GaWorkloadBehavior to the scenario, still by selecting the scenario and by using the Profile tab in the Properties View.



«CHESS» model
- «RequirementView» modelRequirementView
- «SystemView» modelSystemView
  - Architecture
  - Attacks
    - «Adversary» Adversary
    - «AttackScenario, GaWorkloadBehavior» SingleAttack
    - «AttackScenario» AltAttackFragment
    - «AttackScenario, GaWorkloadBehavior» OptFragmentAtt
    - «AttackScenario» LoopAttackFragment
    - «AttackScenario» AttackDelay
    - DurationIntervalMin
    - DurationIntervalMax
    - «AttackScenario» EmptyTestAttack
    - Diagram SingleAttack
    - Diagram AltAttackFragment
    - Diagram OptAttackFragment
    - Diagram LoopAttackFragment
    - Diagram AttackDelay

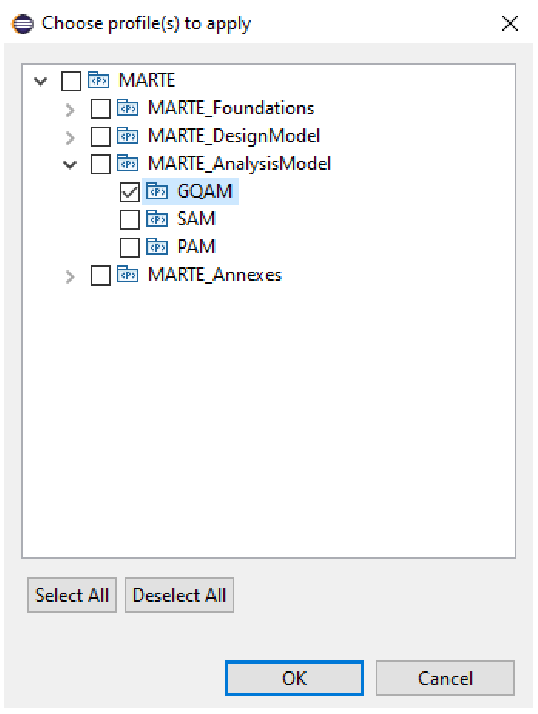
Analysis1

Applied stereotypes:

```
CyberSecurityAnalysis   (from CHESS::Dependability::StateBased::StateBasedAnalysis)
  condition: String [1] =
  context: NFP_String [*] = [SampleSystem.block1_1.Error, SampleSystem.block2_1.Erro
  workload: GaWorkloadBehavior [1..*] = [SingleAttack, OptFragmentAttack]
  platform: GaResourcesPlatform [1..*] = [SampleSystem]
  mode: Mode [*] = []
```

workload
- SingleAttack
- OptFragmentAttack

Figure 12: adding attack scenarios to the analysis context



Choose profile(s) to apply

- [ ] MARTE
  - [ ] MARTE_Foundations
  - [ ] MARTE_DesignModel
  - [ ] MARTE_AnalysisModel
    - [✓] GQAM
    - [ ] SAM
    - [ ] PAM
  - [ ] MARTE_Annexes

Figure 13: MARTE HRM profile

3. A further set-up step available for the Cyber Security Analysis context regards the generation of specific information in the Mobius project which can then be used to facilitate the definition of the reward function. In particular, the Cyber Security Analysis 'context' property is used to add the list of component instances states on which the given reward function of interest will be built (please

refer to section 5 about more details on this). The 'context' property is a list of String[1]; each value has to be provided with the following form:

<instance name>.<error model state name>

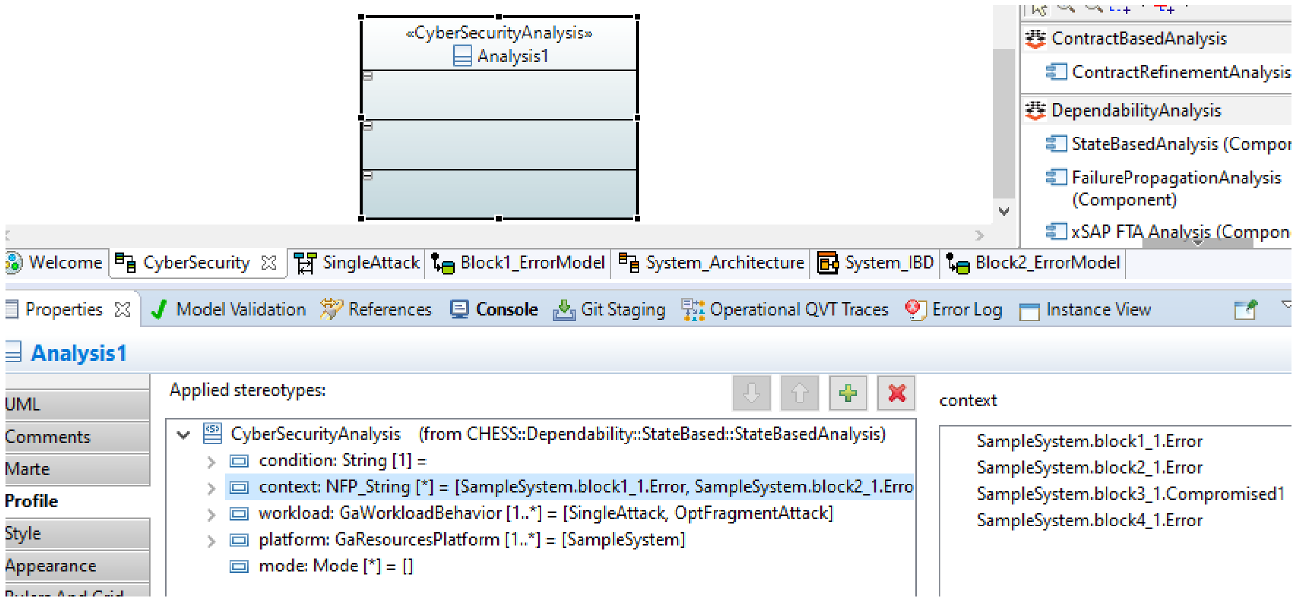
where <error model state name> must be the name of a state defined in the error model available for the component typing the given instance. As alternative, or in addition, a wildcard can be used in place of the instance name, so by using:

*.< error model state name >

meaning that all the instances having a state in the error model with name = <error model state> will be considered.

Figure 14 shows an example of 'context' property referring the Error state of the SampleSystem.block1_1, SampleSystem.block2_1, SampleSystem.block3_1 component and the Compomised1 state of the SampleSystem.block3_1 component.

It is worth noting that the usage of components instances allows a great flexibility in the analysis, and also for what regards the error state list definition; e.g. different states for different instances of the same component can be provided and then measured.



Analysis1

Applied stereotypes:

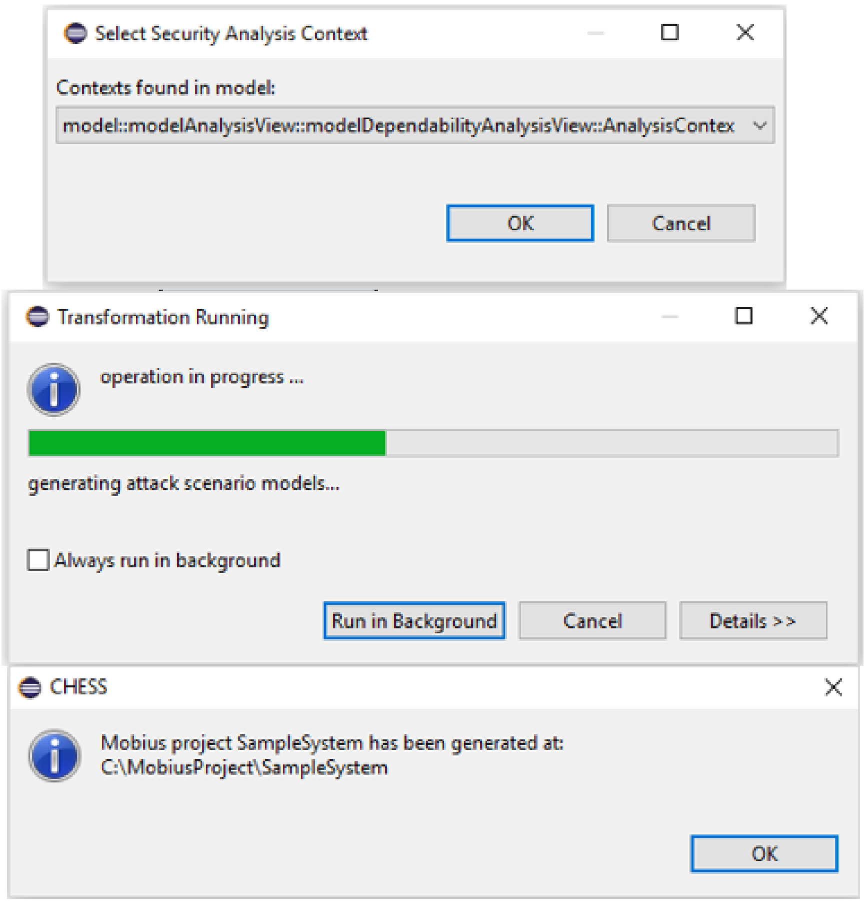
CyberSecurityAnalysis (from CHESS::Dependability::StateBased::StateBasedAnalysis)
condition: String [1] =
context: NFP_String [*] = [SampleSystem.block1_1.Error, SampleSystem.block2_1.Erro
workload: GaWorkloadBehavior [1..*] = [SingleAttack, OptFragmentAttack]
platform: GaResourcesPlatform [1..*] = [SampleSystem]
mode: Mode [*] = []

context
SampleSystem.block1_1.Error
SampleSystem.block2_1.Error
SampleSystem.block3_1.Compromised1
SampleSystem.block4_1.Error

Figure 14: adding component reward states

4. Launch "Generate Mobius Project" from CHESS->Analysis->Dependability Menu and select the Analysis Context:

[1] String is used here because the UML metamodel does not provide a way to model the concept of state at instance level.


Select Security Analysis Context
Contexts found in model:
model::modelAnalysisView::modelDependabilityAnalysisView::AnalysisContex
OK
Cancel
Transformation Running
operation in progress ...
generating attack scenario models...
Always run in background
Run in Background
Cancel
Details >>
CHESS
OK

Mobius project SampleSystem has been generated at:
C:\MobiusProject\SampleSystem

Figure 15: Launch Mobius Project Generation

5 CHESS to Mobius Transformation

5.1 Overview

In order to generate a valid Mobius project a model to model transformation in QVT-O (http://www.eclipse.org/mmt/qvto) from CHESS to an intermediate Ecore SAN model and a set of transformation in Acceleo (https://www.eclipse.org/acceleo/) to transform entities from the Ecore SAN model to SAN or REP/JOIN models in Mobius have been designed and developed. The following list sums up this transformation process:

1. **Atomic Component** (SAN Model): SysML Block no further decomposed and stereotyped as <<ErrorModelBehavior>> with one or more Port stereotyped as “Vulnerability” and StateMachine stereotyped as <<ErrorModel>> and modelled as in section 2.2 and 3.
2. **Reward atomic model** (SAN Model**):** contains the set of component states as specified in the CHESS Analysis Context
3. **Attack Scenario** (SAN Model): Interaction stereotyped as <<AttackScenario>> and modelled following the guidelines of section 3.3
4. **Composed model** (REP/JOIN Model): contains the representation of the complete hierarchical structure of the CHESS components instances. Composite instances with number of parts greater than 1 are mapped to Join nodes. Composite instances with one single inner part are mapped to

Rep nodes. Rep is also used to represent instances with multiplicity greater than 1. Components with no error model are mapped only if they contain some internal part with error model. Submodels are used to represent leaf component instances, with the link to the corresponding atomic components. Submodels for the reward and the attack scenarios atomic models are also connected to the top level Join node. Join state variable related to the atomic components, the attack scenarios and reward states are also created. Figure 16 shows an example of Rep/Join composed model generated from CHESS for a system with a level of decomposition up to 3; on the right, the reward and attack scenario submodels linked to the top level Join are also visible.

Additionally the transformation generates a Mobius Project File (.prj) to ensure that the project is recognized and read by Mobius correctly.

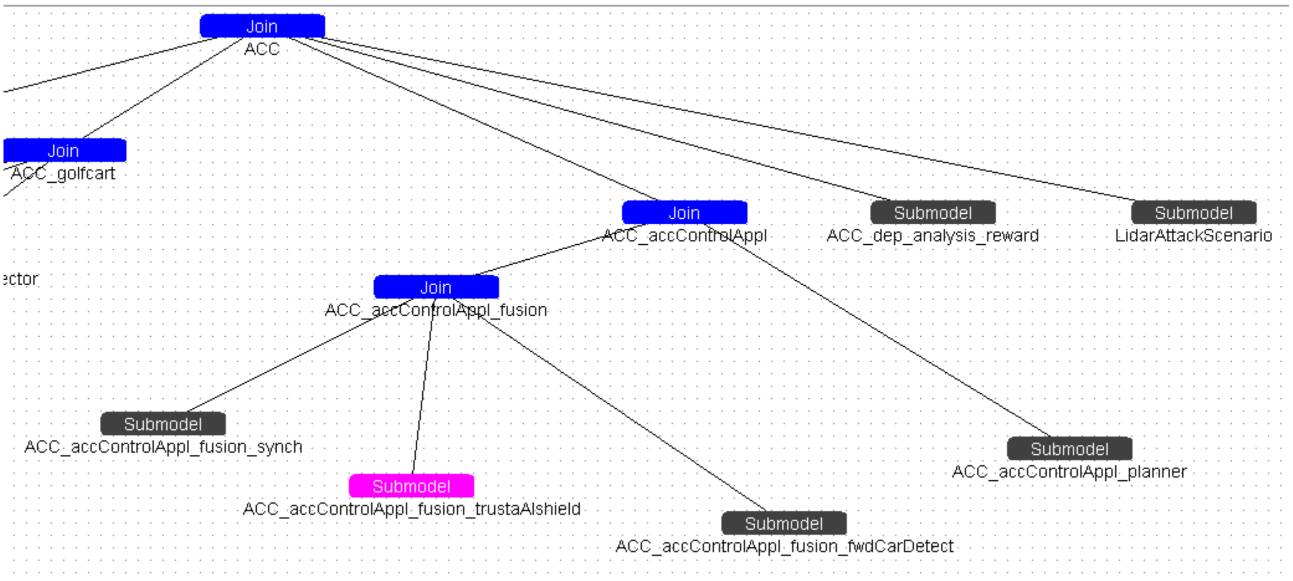

Join
ACC
Join
ACC_golfcart
ector
Join
ACC_accControlAppl
Submodel
ACC_dep_analysis_reward
Submodel
LidarAttackScenario
Join
ACC_accControlAppl_fusion
Submodel
ACC_accControlAppl_fusion_synch
Submodel
ACC_accControlAppl_fusion_trustaAIshield
Submodel
ACC_accControlAppl_fusion_fwdCarDetect
Submodel
ACC_accControlAppl_planner

Figure 16 : SAN composite

5.2 About transformation of failures propagation

For a given terminal/atomic component, i.e. a CHESS SysML Block no further decomposed and stereotyped as <<ErrorModelBehavior>>, the tool checks its internal propagation transitions; in particular, for any internal propagation transition (if any), it checks if the propagation can actually occur by looking at the possible failures that can appear at the input ports of the component. The failures that can appear at the input ports of the given components are the ones that can be generated by direct connected components; so in particular the tool checks if a direct connected terminal component can act as a source of the failure which has been declared to fire the given internal propagation transition.

For instance, let's suppose to have a terminal component X having an error model with an internal propagation representing a failure F occurring at a given input port i_X. In order to have the transition fired, there must exist a terminal component Y with an output port o_Y *direct connected* to i_X; in addition the error model of component Y must come with a Failure transition declaring F as output failure for port o_Y. Two terminal components are direct connected when:

- they are the same level of the hierarchical design (see block2_1 and block3_1 in Figure 17)
- they are at different levels of the hierarchy but directly reachable via delegations connectors (e.g. see block2_1 and block1_2 in Figure 18).

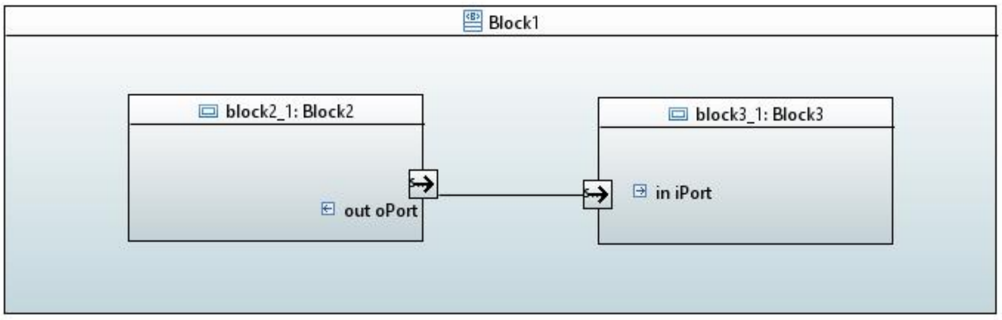


Figure 17: direct connected components, same level

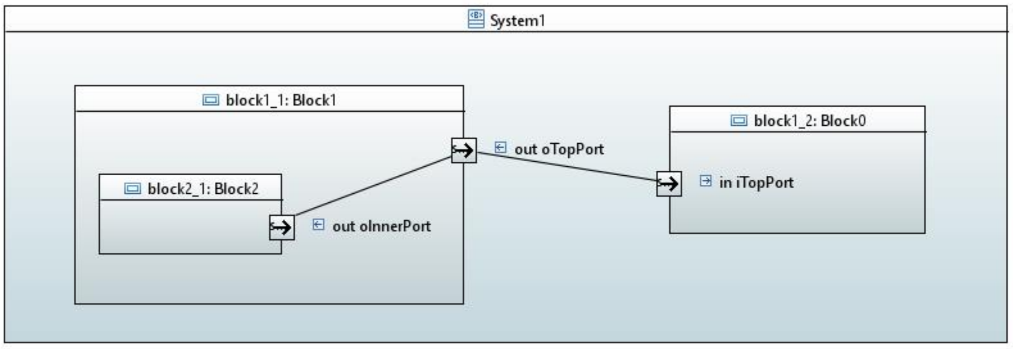


Figure 18: direct connected components, different level

In case a propagation between a source and target components, Y and X respectively in the example above, is found, then the transformation generates:

- an atomic model connected to the root Join node of the Mobius project representing the failure propagation model of the two components and having:
  - one place P_F_O representing the failure in output
  - one timed activity T
  - one place P_F_I representing the failure in input
  - the connections between P_F_O → T → P_F_I
- the atomic model of the source and target components are enriched with the information regarding the failure in output and input. Currently
  - one place representing the failure in output (P_Y_O) is generated in the atomic model of the source component
  - one place representing the failure in output (P_X_I) is generated in the atomic model of the target component
- the places (P_Y_O, P_F_O) and (P_X_I, P_F_I) are then joined in the mail model by using the Mobius shared variables mechanisms.

5.3 SAN Ecore model and traceability information

The intermediate SAN Ecore model derived from the CHESS model and finally used to generate the Mobius model is automatically saved in the SAN folder located under the current CHESS project (the file has .sanmodel as extension). The model can be navigated in the CHESS Eclipse environment by using the default tree editor (double click on the file to open it).

In addition, the traceability information about the CHESS entities and the entities generated in the Ecore SAN model is saved in the SAN folder as .qvtoTrace file. The file has a dedicated editor which can be used to check the mappings.

The structure of the SAN Ecore model is very close to the Mobius generated one, so e.g. given one entity in the Mobius model, it is quite easy to retrieve the corresponding one in the SAN Ecore and vice versa. By navigating the traceability model, it is possible to have a better understanding of the CHESS entity(ies) which generate an entity in Mobius.

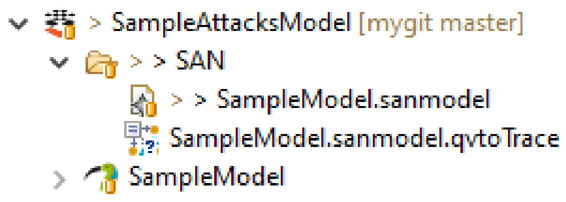
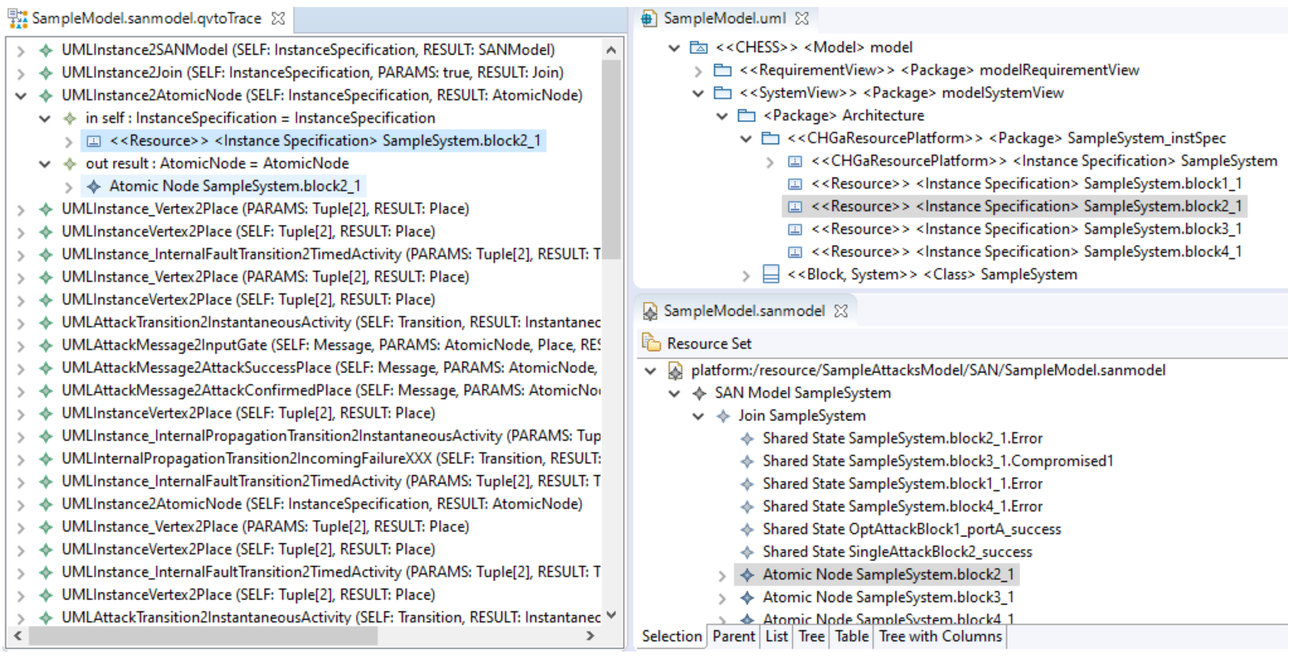

> SampleAttacksModel [mygit master]
> > SAN
> > SampleModel.sanmodel
SampleModel.sanmodel.qvtoTrace
SampleModel

Figure 19: CHESS project SAN Folder

Figure 20 shows an example of traceability model (on the right side), related to a specific CHESS-Mobius model generation, with the list of the mapping rules applied. In particular, a mapping rule between UML instance and atomic node has been expanded. The rule owns the detail of the source (in) and target (out) entities, UML instance component and SAN atomic node respectively: it is possible to double click on one entity to open the associated tree editor to then check it in the owning resource (UML model on the top right side and SAN Ecore model on bottom right side of the figure). In particular Figure 20 shows the traceability between the SampleSystem.block2_1 SAN atomic node and the homonym component instance in CHESS.



SampleModel.sanmodel.qvtoTrace

```
> UMLInstance2SANModel (SELF: InstanceSpecification, RESULT: SANModel)
> UMLInstance2Join (SELF: InstanceSpecification, PARAMS: true, RESULT: Join)
v UMLInstance2AtomicNode (SELF: InstanceSpecification, RESULT: AtomicNode)
  v in self : InstanceSpecification = InstanceSpecification
    > <<Resource>> <Instance Specification> SampleSystem.block2_1
  v out result : AtomicNode = AtomicNode
    > Atomic Node SampleSystem.block2_1
> UMLInstance_Vertex2Place (PARAMS: Tuple[2], RESULT: Place)
> UMLInstanceVertex2Place (SELF: Tuple[2], RESULT: Place)
> UMLInstance_InternalFaultTransition2TimedActivity (PARAMS: Tuple[2], RESULT: T
> UMLInstance_Vertex2Place (PARAMS: Tuple[2], RESULT: Place)
> UMLInstanceVertex2Place (SELF: Tuple[2], RESULT: Place)
> UMLAttackTransition2InstantaneousActivity (SELF: Transition, RESULT: Instantanec
> UMLAttackMessage2InputGate (SELF: Message, PARAMS: AtomicNode, Place, RES
> UMLAttackMessage2AttackSuccessPlace (SELF: Message, PARAMS: AtomicNode,
> UMLAttackMessage2AttackConfirmedPlace (SELF: Message, PARAMS: AtomicNo
> UMLInstanceVertex2Place (SELF: Tuple[2], RESULT: Place)
> UMLInstance_InternalPropagationTransition2InstantaneousActivity (PARAMS: Tup
> UMLInternalPropagationTransition2IncomingFailureXXX (SELF: Transition, RESULT:
> UMLInstance_InternalFaultTransition2TimedActivity (PARAMS: Tuple[2], RESULT: T
> UMLInstance2AtomicNode (SELF: InstanceSpecification, RESULT: AtomicNode)
> UMLInstance_Vertex2Place (PARAMS: Tuple[2], RESULT: Place)
> UMLInstanceVertex2Place (SELF: Tuple[2], RESULT: Place)
> UMLInstance_InternalFaultTransition2TimedActivity (PARAMS: Tuple[2], RESULT: T
> UMLInstanceVertex2Place (SELF: Tuple[2], RESULT: Place)
> UMLAttackTransition2InstantaneousActivity (SELF: Transition, RESULT: Instantanec
```

SampleModel.uml

```
v <<CHESS>> <Model> model
  > <<RequirementView>> <Package> modelRequirementView
  v <<SystemView>> <Package> modelSystemView
    v <Package> Architecture
      v <<CHGaResourcePlatform>> <Package> SampleSystem_instSpec
        > <<CHGaResourcePlatform>> <Instance Specification> SampleSystem
          <<Resource>> <Instance Specification> SampleSystem.block1_1
          <<Resource>> <Instance Specification> SampleSystem.block2_1
          <<Resource>> <Instance Specification> SampleSystem.block3_1
          <<Resource>> <Instance Specification> SampleSystem.block4_1
      > <<Block, System>> <Class> SampleSystem
```

SampleModel.sanmodel

```
Resource Set
v platform:/resource/SampleAttacksModel/SAN/SampleModel.sanmodel
  v SAN Model SampleSystem
    v Join SampleSystem
        Shared State SampleSystem.block2_1.Error
        Shared State SampleSystem.block3_1.Compromised1
        Shared State SampleSystem.block1_1.Error
        Shared State SampleSystem.block4_1.Error
        Shared State OptAttackBlock1_portA_success
        Shared State SingleAttackBlock2_success
      > Atomic Node SampleSystem.block2_1
      > Atomic Node SampleSystem.block3_1
      > Atomic Node SampleSystem.block4_1
```

Figure 20: CHESS to SAN Ecore traceability

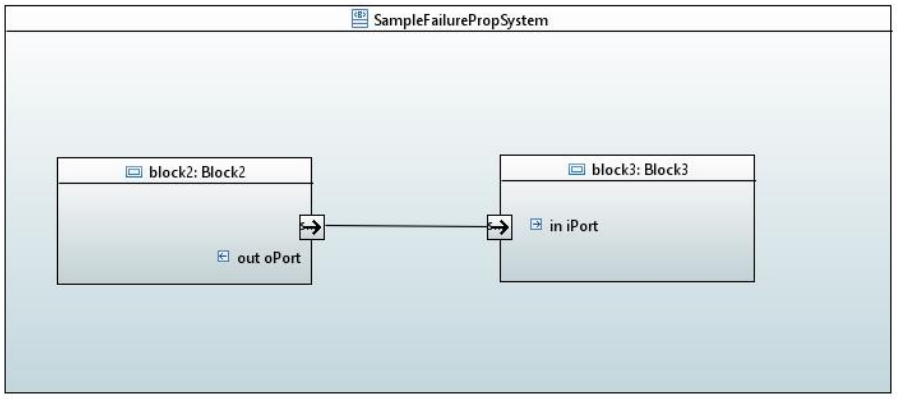
6 Current assumptions and possible improvements

- Regarding transformation of failure propagation: input failure expressions attached to the internal propagation transitions must have a single fault on a single input port. Output failure expression attached to failure transitions must have a single fault on a single output port. So complex failures propagations (like an internal propagation enabled by a boolean condition of failures occurring on different input ports) are not currently supported.
- A CHESS component having an ErrroModel state machine attached (through the ErrorModelBehaviour stereotype) is mapped to an atomic SAN node and is considered as terminal in CHESS even if it is decomposed. Given a composed components having and error model attached, internal components parts could also be checked and mapped to SAN.

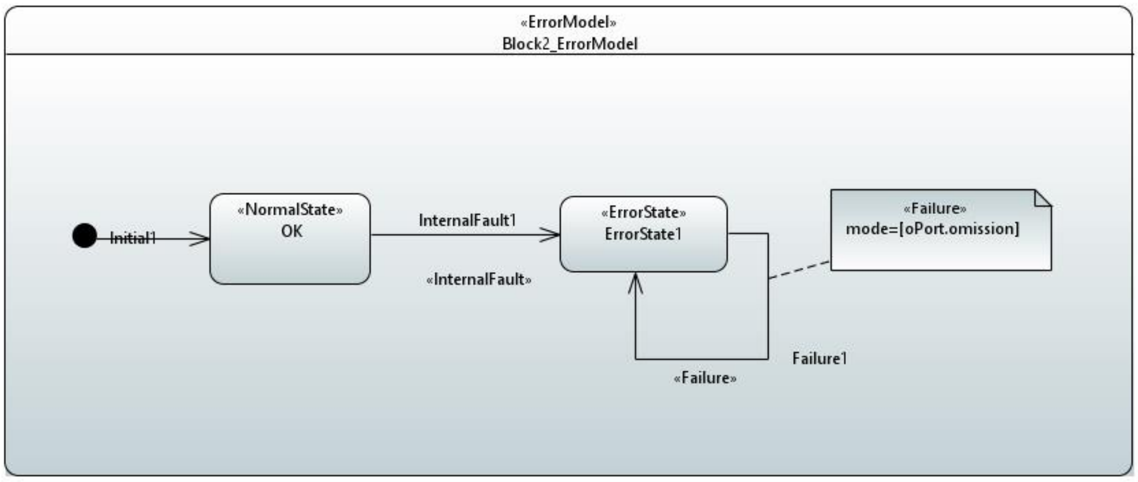
7 Example

7.1 Failure Propagation

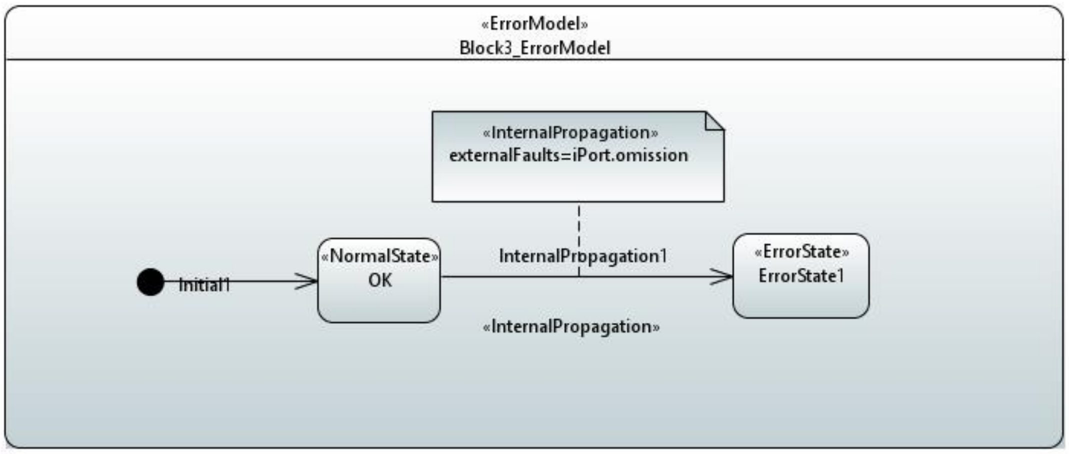
Let’s consider a system composed by two parts connected as in the following figure.



Block2 is able to generate an *omission* failure on its output port.

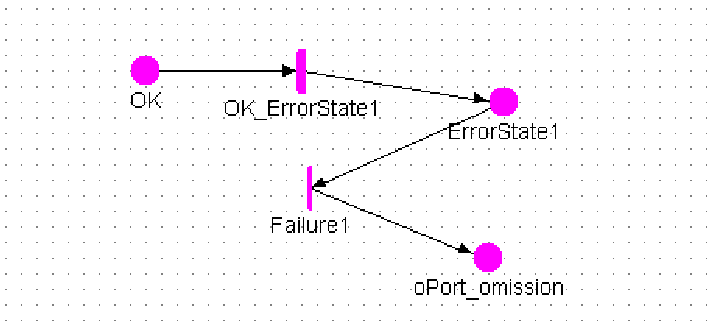


Block3 can switch to an error state when an *omission* failure is available at its input port.

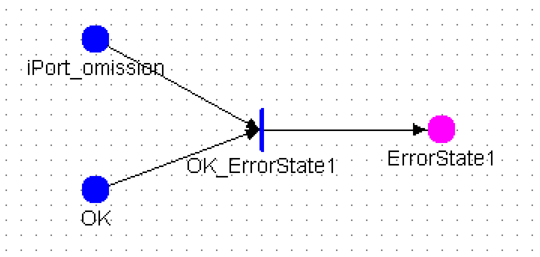


So considering the error models and the components input and output ports binding, a failure propagation between block2 and block3 can happen.

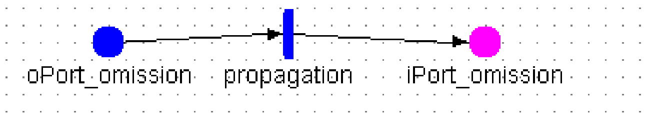
The error model of the block2 component is mapped to the following SAN model:



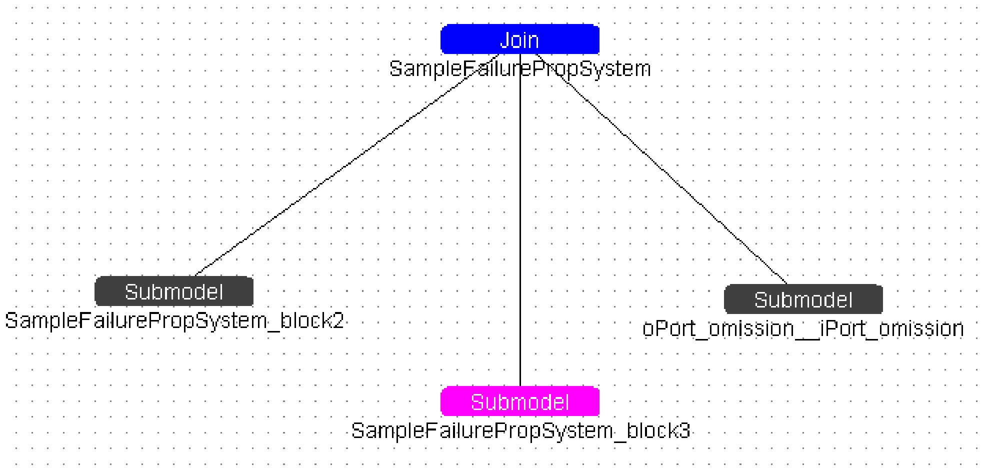
The error model of the block3 component is mapped to the following SAN model:



An atomic model representing the failure propagation flowing between the componets is also generated:

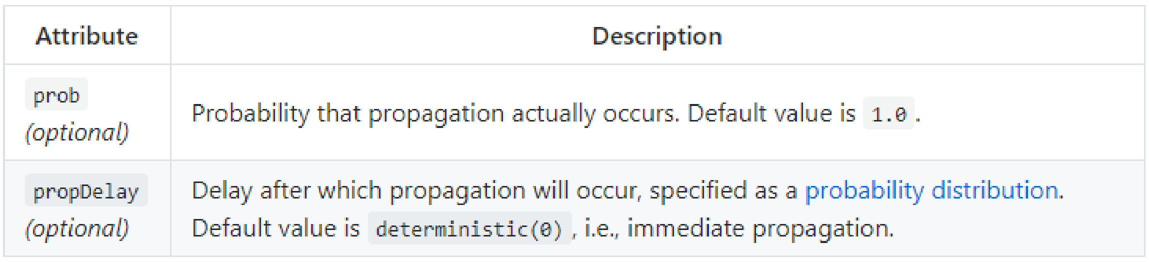


Finally the composite SAN model is generated by joining the blocks and failure propagation atomic models:


Join
SampleFailurePropSystem
Submodel
SampleFailurePropSystem_block2
Submodel
oPort_omission__iPort_omission
Submodel
SampleFailurePropSystem_block3

Information available in CHESS (error model) currently not considered by the mapping, to be considered for possible extensions:

- The delay (i.e. the time after which propagation occurs) of the InternalPropagation transition
- In CHESS, <<propagation>> stereotype can be added to the connector which binds the components; <<propagation>> comes with the following attributes:



Attribute	Description
`prob` *(optional)*	Probability that propagation actually occurs. Default value is `1.0` .
`propDelay` *(optional)*	Delay after which propagation will occur, specified as a probability distribution. Default value is `deterministic(0)` , i.e., immediate propagation.

8 References

[1] https://www.polarsys.org/chess/publis/CHESSToolset_UserGuide.pdf.

[2] https://www.polarsys.org/chess/publis/CHESSMLprofile.pdf.

[3] Popov, P., Models of reliability of fault-tolerant software under cyber-attacks in The 28th IEEE International Symposium on Software Reliability Engineering (ISSRE'2017). 2017, IEEE: Toulouse, France. p. 228-239. Available from: https://doi.org/10.1109/ISSRE.2017.23.