



CHESS

State-based Dependability Analysis

University of Florence

Topics

- Analysis overview
- Which is the main purpose of the analysis?
- When and how should the analysis be invoked?
- Which is the information that users should provide in input to invoke the analysis?
- How the results of the analysis can be interpreted?



CHESS

Analysis Overview

Introduction

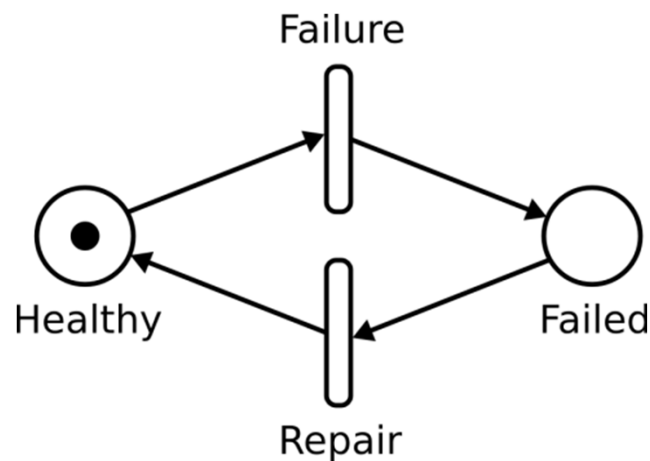
- State-based dependability analysis
 - ◆ A technique to **quantitatively** evaluate the dependability attributes of the system

- The term “state-based” refers to the fact that such techniques use a representation of the system *based on its possible states*, with respect to dependability, and on the *possible transitions* between them

- The analysis is performed on a stochastic model of the system (e.g. Continuous-Time Markov Chains – CTMC)
 - ◆ **In this case:** an extension of Stochastic Petri Nets (SPN), where the firing time of transitions can follow different probability distributions
 - ◆ Therefore, **solved by discrete-event simulation**

State-based analysis explained

- Simplest example – A component has two possible states
 - ◆ Healthy: the component correctly performs its function (provides a correct service)
 - ◆ Failed: the component delivers an incorrect service (or no service at all)

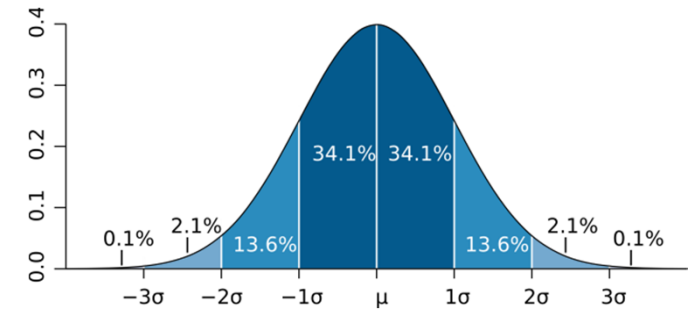


- The system starts in “healthy” state
- After a given amount of time it moves to the “failed” state
- Then it returns to “healthy”

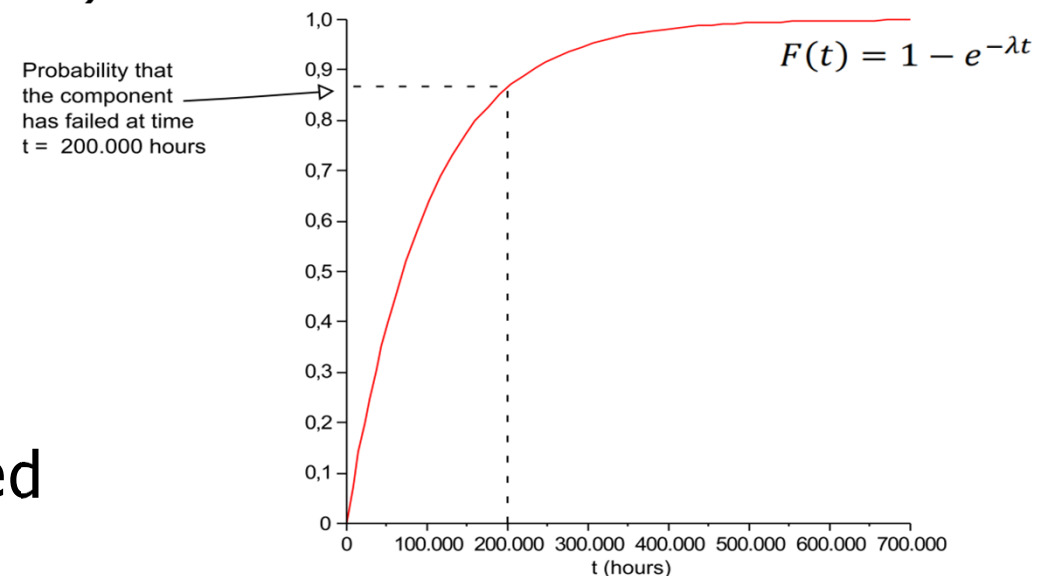
- Which is the probability that the component does not fail until time t ?
 - ◆ (Reliability at time t)
 - ◆ Probability that state “Failed” has not been visited at time t

Probability distributions

- A function that describes the probabilities of a random variable taking certain values
- In state-based analysis
 - ◆ Used to model the time required for the system to move from one state to another
- Exponential distribution
 - ◆ Commonly used to model the occurrence of faults and/or failures (especially for hardware)



- Example: Time to failure of a component
 - ◆ Time to move from "healthy" to "failed"
 - Exponentially distributed
 - With rate λ



Purpose of the analysis

- In principle, different metrics can be computed using the state-based analysis approach
- The CHES plugin implements the following metrics
- Reliability
 - ◆ *Instantaneous*: the probability that the system continuously remains in a “healthy” state from time 0 up to time t
- Availability
 - ◆ *Instantaneous*: the probability that the system is in a “healthy” state at time t
 - ◆ *Interval of time (averaged)*: the fraction of time that the system holds in a “healthy” state in a given interval
- Probability of Failure on Demand (PFD)
 - ◆ Domain-specific metric used in the *petroleum domain*. The probability that the system fails upon a demand to provide the requested service.
 - ◆ It is calculated as 1 minus the instantaneous reliability.

Modelling faults, errors, and failures

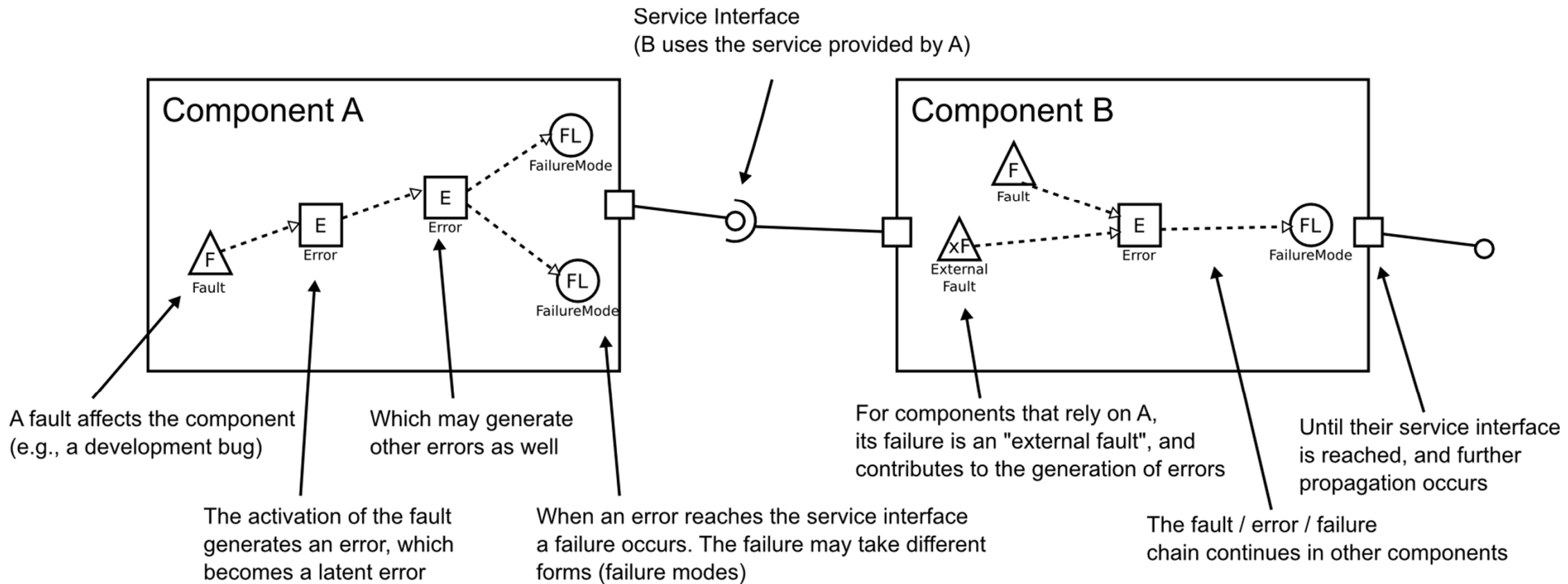
- Additional states can be considered for a given component
 - ◆ In general, the level of detail can vary from very high-level to very low-level (e.g. models for the analysis of protocols)

- Within CHES, we consider...
 - ◆ An initial state, i.e., the **healthy** condition of the component
 - ◆ A certain number of internal “erroneous” states, meaning that a **fault** has occurred, and the internal state of the component has some deviation (**errors**), but not necessarily meaning that the provided service is incorrect

- Transitions between states are characterized by probability values and stochastic time delays
 - ◆ **Failures** are particular kind of transitions, that imply a manifestation of incorrect behaviour at the component interface.
 - ◆ Failures may manifest themselves in different ways, called **failure modes** (e.g., value or timing)

Modelling the propagation chain – Example

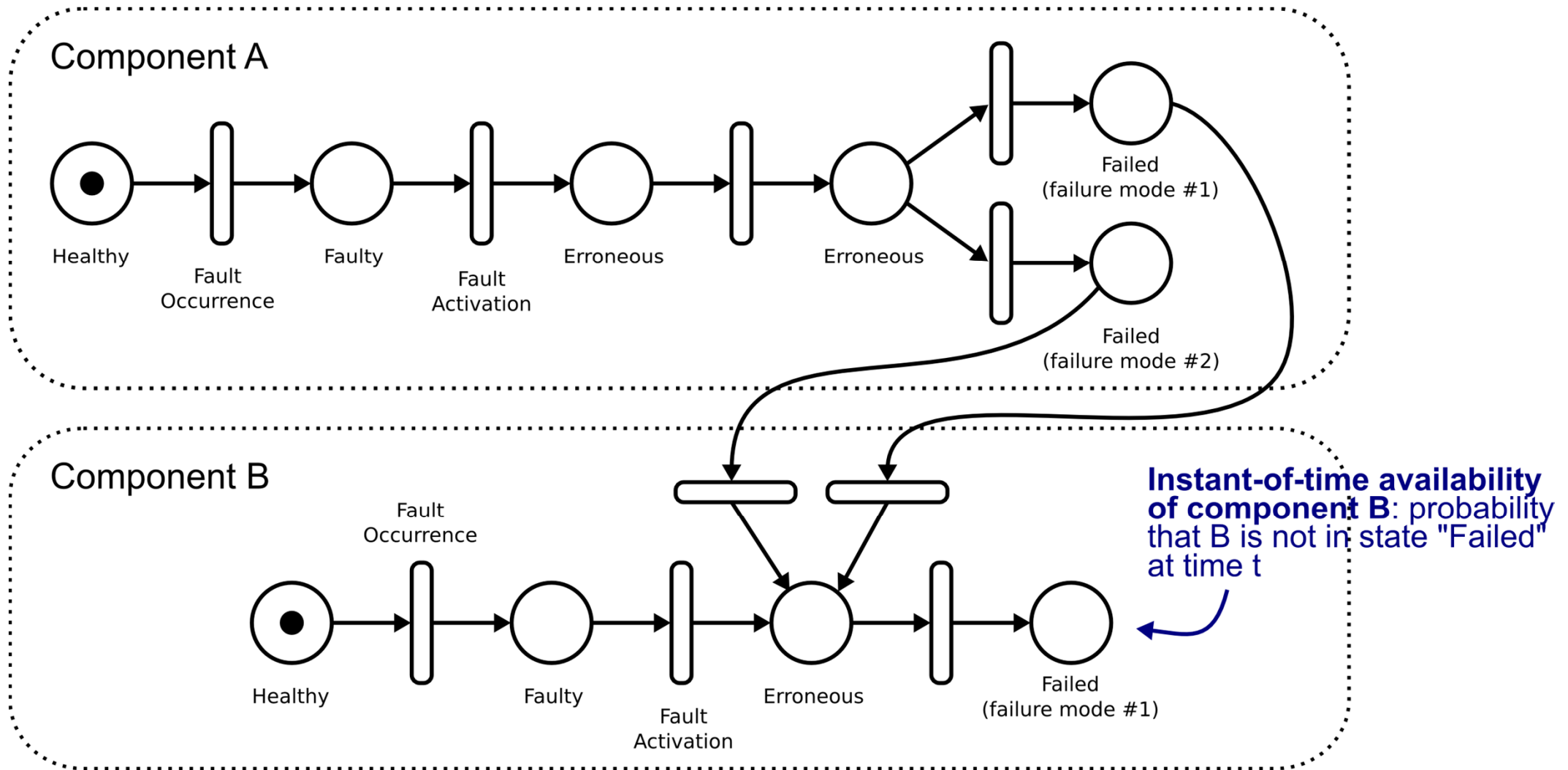
- Component B uses the service provided by Component A



Availability of component B ??

Modelling the propagation chain – Example (2)

- Component B uses the service provided by Component A





CHESS

Applying the CHESS State-Based Analysis

Steps for applying the technique within CHES

- State-based analysis can be applied as soon as a functional model of the system exists
 - ◆ Not necessarily a complete model of the overall system
 - ◆ Analysis is possible even with a partial specification

- Steps required to perform the analysis are the following
 1. Enrich the software, hardware, and system elements with information on dependability properties (e.g., fault occurrence)
 2. Enrich the connections between components, including allocation specifications, with dependability information
 3. (Optional) Define additional maintenance information about periodic maintenance activities
 4. Define the analysis context, i.e., which part of the system is going to be analysed by the tool
 5. Define of the measures of interest
 6. (Optional) Set the values for model parameters
 7. Run the analysis

Notation

■ Time Distributions

- ◆ Probability distributions for delays are specified using the MARTE's VSL syntax
- ◆ Supported distributions
 - Exponential: `exp(lambda)` `exponential(lambda)`
 - Deterministic: `det(value)` `deterministic(value)`
 - Uniform: `uni(a,b)` `uniform(a,b)`
 - Normal: `norm(mean,var)` `normal(mean,var)`
 - Gamma: `gam(alpha,beta)` `gamma(alpha,beta)`
 - Weibull: `wei(alpha,beta)` `weibull(alpha,beta)`
- ◆ Unit of measurement is not specified: should be implicitly the same across the entire model

■ Numbers

- ◆ Scientific notation is allowed, e.g., `1.0E-6`

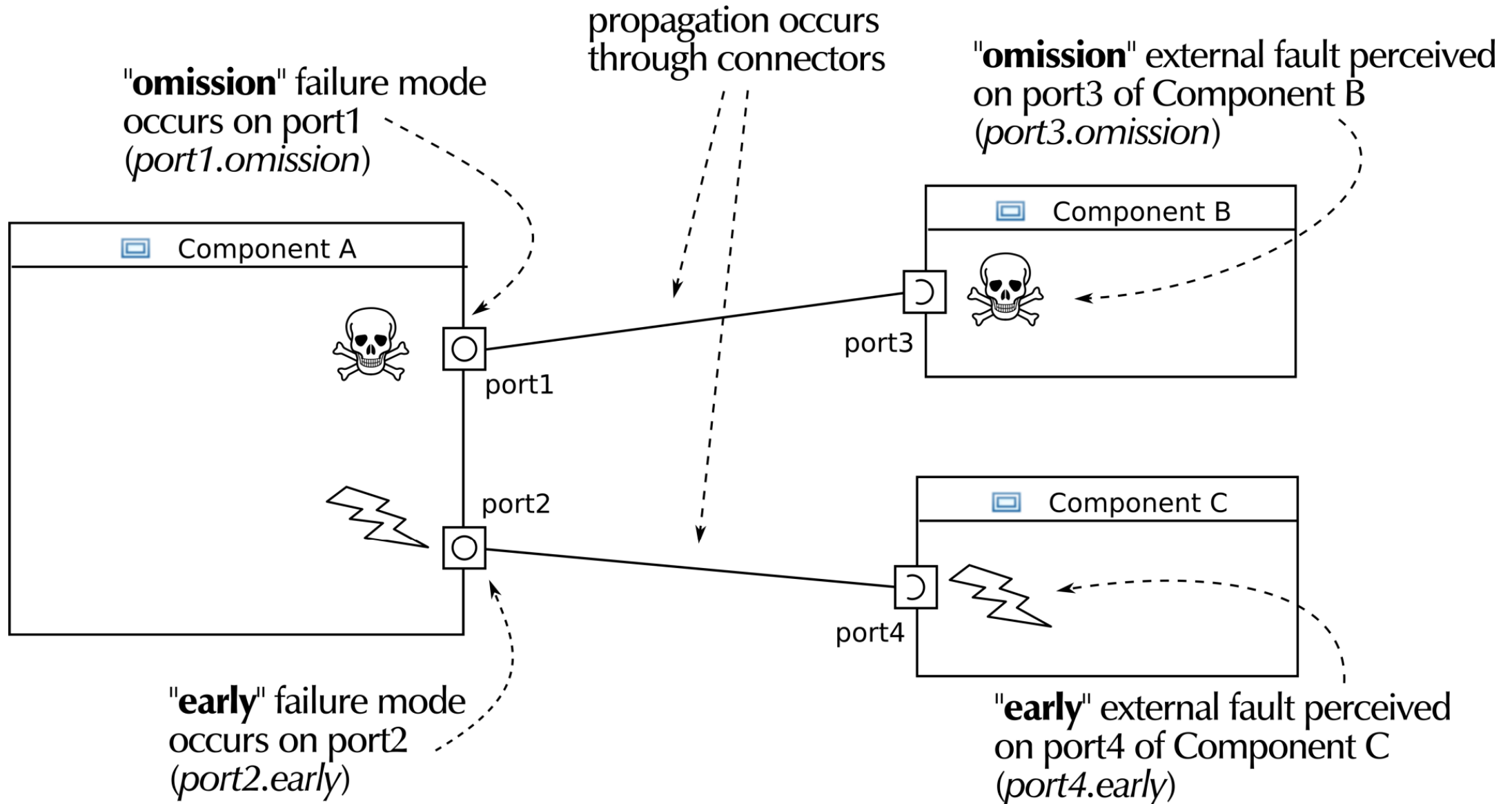
■ Failure modes

- ◆ We use the dot notation to refer to failure modes occurring on specific ports, e.g., `port1.omission`

Assumptions

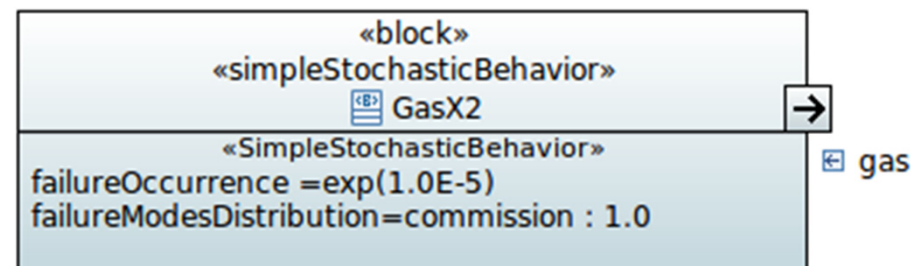
- Automatically generated state-based models in CHES are based on the following set of assumptions
 - ◆ The occurrence of internal faults in different components are independent from each other
 - ◆ The activation delay of any fault is zero: when a fault occurs it immediately generates an error
 - This could also mean: the fault is always present in the component, and it is activated with a certain rate (e.g. software development bugs)
 - ◆ When two components A and B are connected through a port, the external faults experienced by B on that port are the same as the failure modes that may affect A on the same port (and vice-versa)

Ports, Failures, and Propagation



Step 1: Enrich software, hardware, and system elements

- Three ways to add dependability information to a given component of the system
 - ◆ «SimpleStochasticBehavior»
 - ◆ «FLABehavior»
 - ◆ «ErrorModelBehavior»
- The «**SimpleStochasticBehavior**» is the most simple way. When using this stereotype it is assumed that
 - ◆ The component is affected by only one kind of internal faults,
 - ◆ That generate one kind of errors,
 - ◆ And can possibly result in different failure modes, with a certain probability



■ «SimpleStochasticBehavior»

◆ faultOccurrence

Specifies the average time to the occurrence of a fault. This is specified as a Time Distribution

◆ failureModesDistribution [optional]

Specifies the possible failure modes of the component, and their relative probabilities. Uses the grammar:

$\langle \text{FMD} \rangle ::= \langle \text{D} \rangle \mid \langle \text{PD} \rangle \mid \langle \text{PD} \rangle ; \langle \text{PD} \rangle$

$\langle \text{PD} \rangle ::= \langle \text{PORT} \rangle \langle \text{D} \rangle$

$\langle \text{D} \rangle ::= \{ \langle \text{FP} \rangle \}$

$\langle \text{FP} \rangle ::= \langle \text{F} \rangle : \langle \text{P} \rangle \mid \langle \text{F} \rangle : \langle \text{P} \rangle ; \langle \text{FP} \rangle$

$\langle \text{F} \rangle$ is a failure mode, $\langle \text{P} \rangle$ is a probability value,

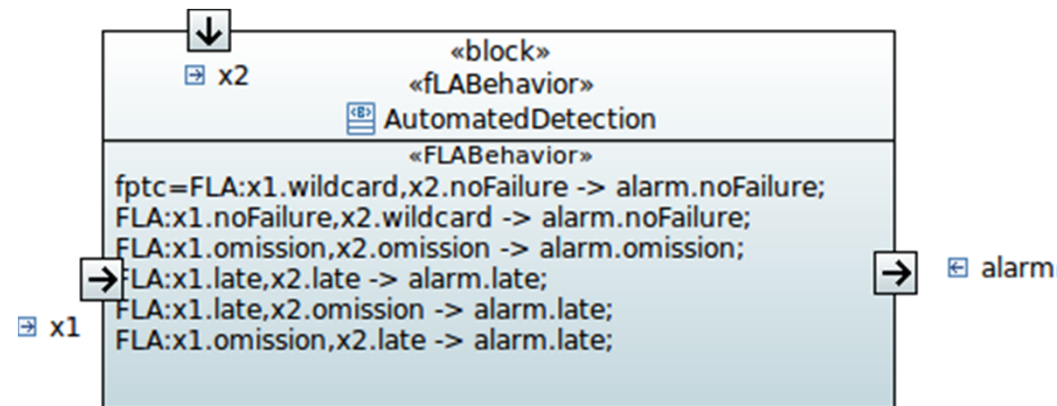
$\langle \text{PORT} \rangle$ is a port of the component

◆ repairDelay [optional]

Specifies the time needed to repair the component, as a Time Distribution

Step 1: Enrich software, hardware, and system elements

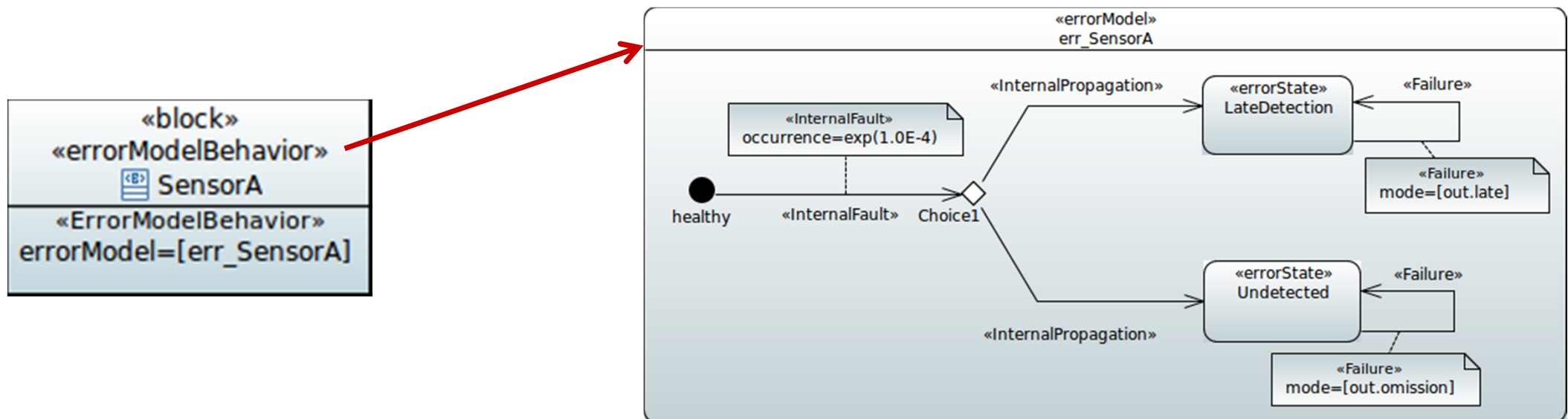
- The «**FLABehavior**» stereotype allow failure logic specifications to be defined for system elements.



- ◆ Specify how components propagate and/or transform failure modes experienced at their input
- ◆ Mainly used for FPTC, FI4FA, and related failure logic analysis techniques
- ◆ However, the information provided by those specifications can be used as input for state-based analysis as well

The Error Model

- The «**ErrorModelBehavior**» stereotype allows the provision of more details on faults, errors, and failure modes of system elements
 - ◆ errorModel attribute, references the actual error model specification
- Error Model
 - ◆ A particular kind of StateMachine diagram containing information on propagation internal to the component
 - ◆ Stereotyped with the «**ErrorModel**» stereotype



The Error Model (2)

- Initial state
 - ◆ It represents the “healthy” state of the component
- Errors
 - ◆ UML State, with the «**ErrorState**» stereotype
- Internal faults
 - ◆ UML Transitions, with the «**InternalFault**» stereotype
 - connecting the initial state and an error state
 - occurrence – time to fault occurrence (time distribution)
- Internal propagations
 - ◆ UML Transitions, with the «**InternalPropagation**» stereotype
 - delay – time after which propagation occur
 - weight - relative probability of occurrence
 - externalFaults – Boolean expression on the occurrence of external faults (i.e., failures incoming on input ports of the component)
- Failures
 - ◆ UML Transition, with the «**Failure**» stereotype
 - mode – the failure mode(s) under which the failure manifest itself on the port(s) of the component

Hierarchical Modeling

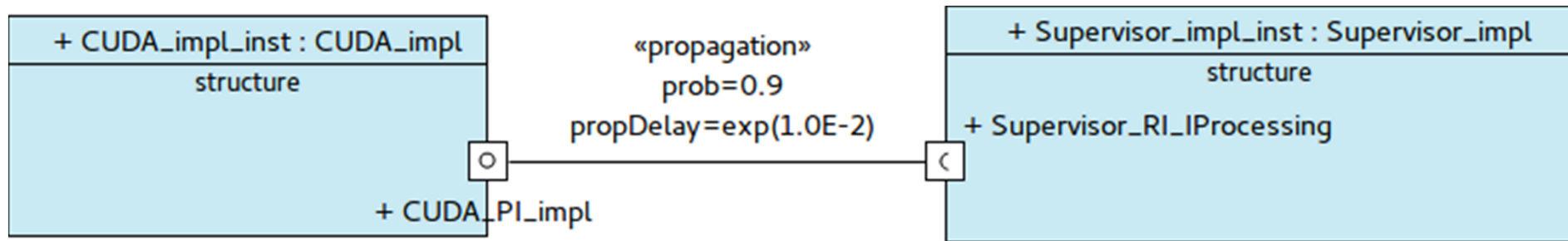
- Hierarchical modeling is supported by state-based analysis
- Modeling
 - ◆ In case the internal structure of a component is detailed by sub-components, those are taken into account for the analysis
 - ◆ Together with possible delegation and/or promotion of their ports
 - ◆ Unless information on the dependability behaviour of the container is specified. In this case, its internal structure is discarded
- Analysis
 - ◆ The specification of the services that must be analysed takes into account hierarchical structures
 - ◆ Delegation paths are followed to find the component that really implements that service

Step 2: Defining propagation paths between components

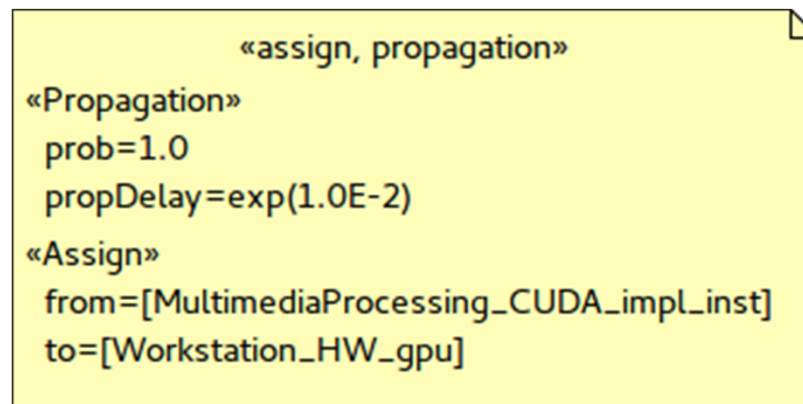
- In this step the possible propagation paths between components may be enriched with dependability information.
- Propagation may occur when
 - ◆ Functional relations exists between two components, either at hardware, software, or system level (UML::Connector)
 - ◆ An allocation relation exists between a software and a hardware component (MARTE::Assign)
- By default it assumed that *immediate* and *deterministic* propagation will occur through these paths.
- However, different behaviour may be specified using the **«Propagation»** stereotype
 - ◆ prob – probability that propagation occurs
 - ◆ propDelay – delay after which propagation occurs (time distribution)

Propagation paths – Example

- Propagation path between two software components

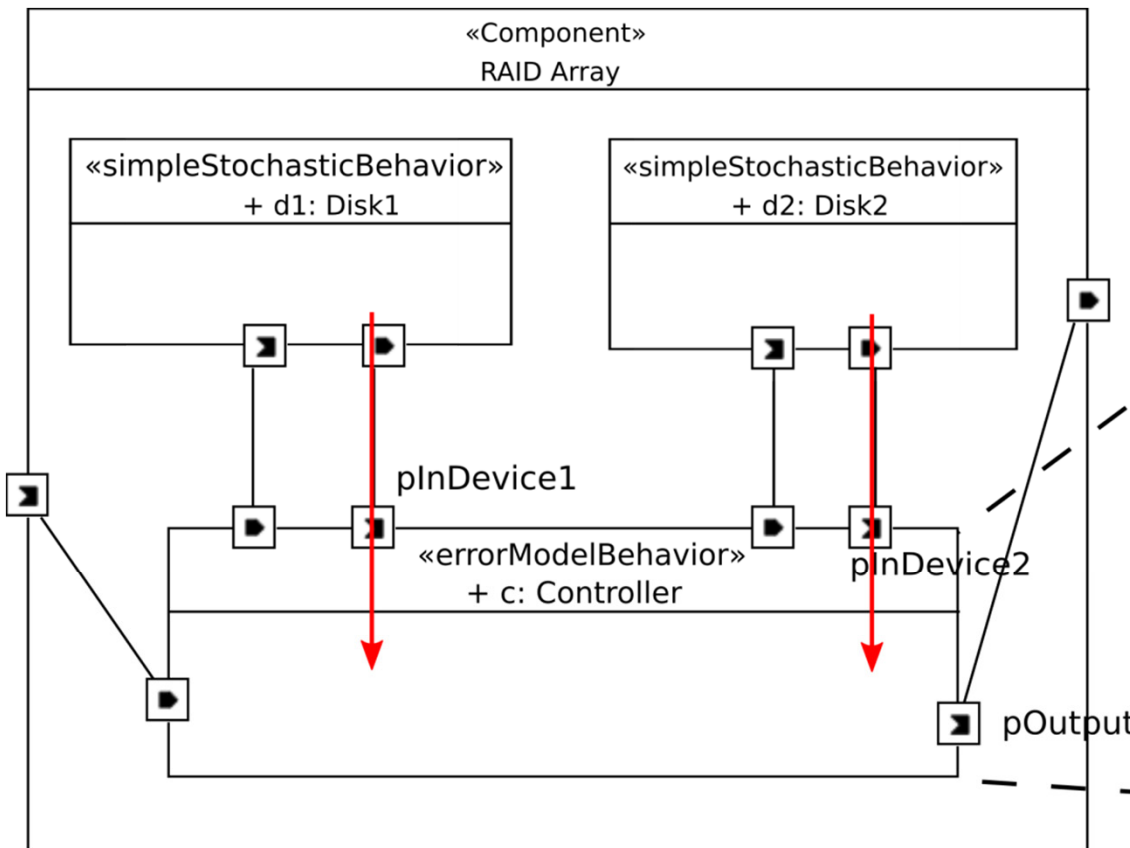


- Propagation path derived from the allocation of a software component on a hardware component

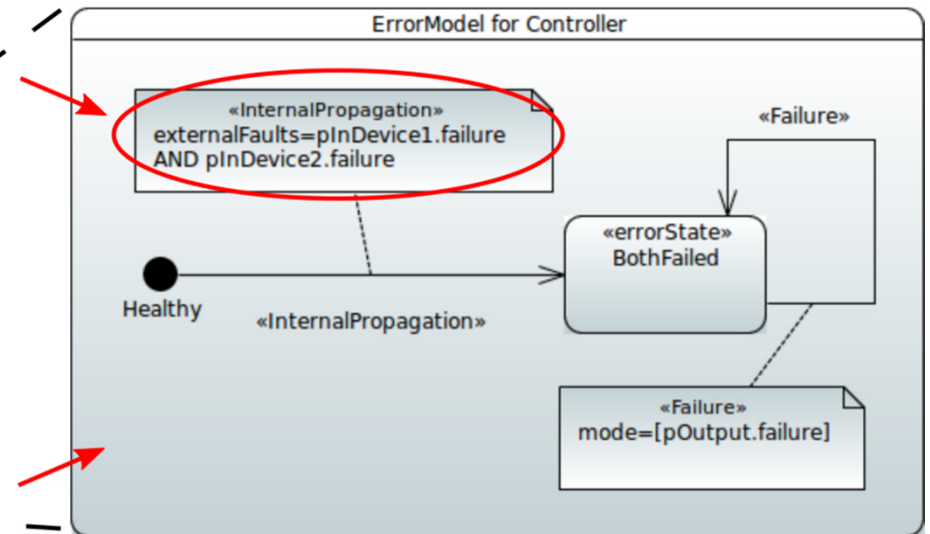


Redundant structures

- Support for redundancy structures modeling is provided through the use of
 - ◆ Composed components
 - ◆ Error models



- Example: 2-disks raid array with controller



Component instances

- When component instances are derived from component implementations (“CHES Build Instance”)
 - ◆ dependability information is automatically propagated to component instances.

- The user may however modify the property values for each individual instance
 - ◆ Making it possible to have different dependability attributes for different instances of the same component.

- For example
 - ◆ Two identical hard disks can have different fault occurrence rates
 - ◆ E.g. if one of them is subject to heavier vibrations than the other

■ Maintenance

- ◆ The repair delay of components can be specified using the “repairDelay” attribute of the «SimpleStochasticBehavior» stereotype (see Step 1)
- More detailed maintenance policies (both preventive and corrective) are modelled using the «**Repair**» stereotype
 - ◆ It is applied to UML Activity elements
- This way of modelling maintenance is also the only choice for components for which the failure behavior is specified using error model
 - ◆ The error model state machine does not define repairs

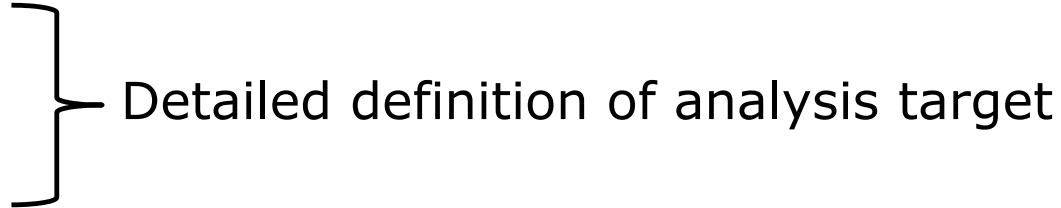
«Repair» Activities: Attributes

- When the activity is executed, the component returns to its original state, i.e., the “healthy” state
- Attributes
 - ◆ duration
 - The time required to perform the activity (time distribution)
 - ◆ when
 - The policy for the execution of the activity, specified using a custom grammar (see the CHES Profile Specification)
 - In particular
 - AtTime { t } – the activity is executed once at time t
 - Periodic { d } – the activity is executed periodically, with period d
 - ◆ probSuccess
 - The probability that the activity is successfully completed
 - ◆ targets
 - The component instances that will be repaired

Step 4: Definition of the analysis context

- In this step we define which part of the system is going to be analysed by the tool
 - ◆ Create a new **Class Diagram** in the DependabilityAnalysisView
 - ◆ Create a new Component, with the «**StateBasedAnalysis**» stereotype
 - There should be one of these stereotypes for every measure that the tool should produce as output

- Attributes involving the definition of the measure:
 - ◆ platform – identifies the subsystem under analysis
 - ◆ measure – definition of the measure to be evaluated
 - ◆ targetDepComponent
 - ◆ targetPort
 - ◆ targetFailureMode
 - ◆ measureEvaluationResult
where the result of the analysis is back-annotated



Step 4: Definition of the analysis context

- platform
 - ◆ Selection of the platform on which state-based analysis should be performed
 - ◆ The target element should be a component instance stereotyped with the «CHGaResourcePlatform» stereotype

- By default, **all output ports of the selected platform** are considered for the analysis
 - ◆ When one of them is failed, the system is considered failed
 - ◆ If the platform has not explicitly defined output ports, then all those of its subcomponents are considered

Step 4: Definition of the analysis context

- To refine this specification, the attributes `targetDepComponent`, `targetPort`, and `targetFailureMode` can be used

- `targetDepComponent`
 - ◆ One or more component instances, inside the platform, to which the analysis should be limited. The system is failed when the ports of those components are failed.

- `targetPort`
 - ◆ One or more specific ports of component instances. Only when one of those ports is failed, the system is considered failed.

- `targetFailureMode`
 - ◆ One or more specific failure modes, identified by their names as a string. The system is considered failed only when a failure of those kinds occurs on one of the ports of interest.

Step 5: Definition of the measures of interest

- The measure to be evaluated is Specified in the measure attribute of the «**StateBasedAnalysis**» stereotype, using the VSL-like syntax
 - ◆ Reliability { instantOfTime = t }
 - Instant of time reliability: probability that the component does not fail until time t
 - ◆ Availability { instantOfTime = t }
 - Instant of time availability: probability that at time t the component is not failed (it also considers repairs)
 - ◆ Availability { intervalEnd = t }
 - Fraction of time that the component is not failed in the interval [0,t]
 - ◆ PFD { t }
 - Probability of failure on demand, computed as $1 - \text{Reliability} \{ \text{instantOfTime} = t \}$

Step 6: Run the analysis

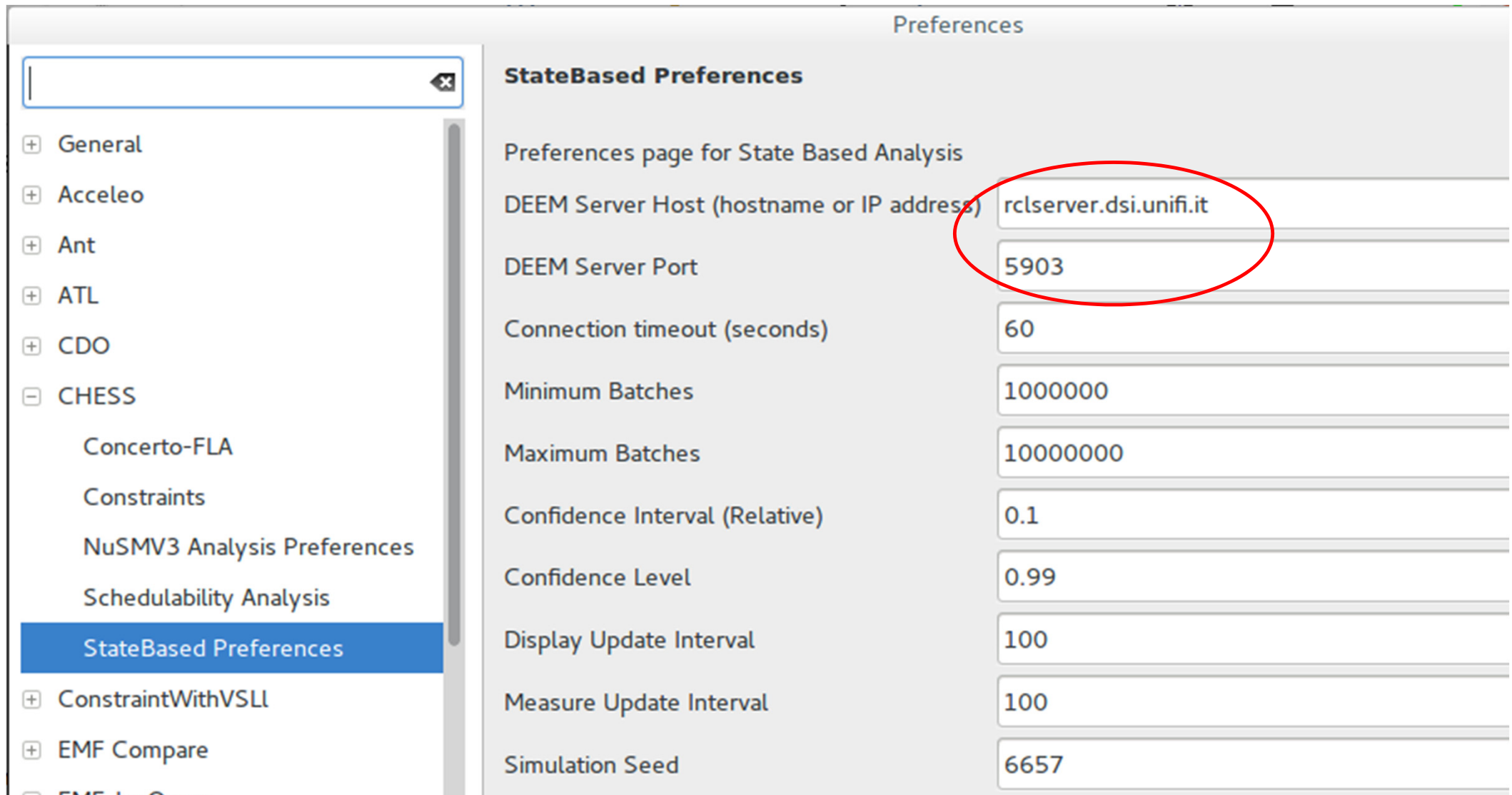
- The user can now run the analysis
 - ◆ The result is back-annotated in the measureEvaluationResult attribute of the «StateBasedAnalysis» stereotype

- The selected analysis tool for state-based analysis is the **DEEM Simulator**
 - ◆ Tool developed by UNIFI and ISTI-CNR
 - ◆ Runs under Linux

- Within CHES: Used as a remote service
 - ◆ Solves licensing issues
 - ◆ Solves compatibility issues
 - ◆ DEEM service running on UNIFI server
 - Host: **rclserver.dsi.unifi.it**
 - Port: **5903**

Plug-in Configuration

Window → Preferences → CHESS → StateBased

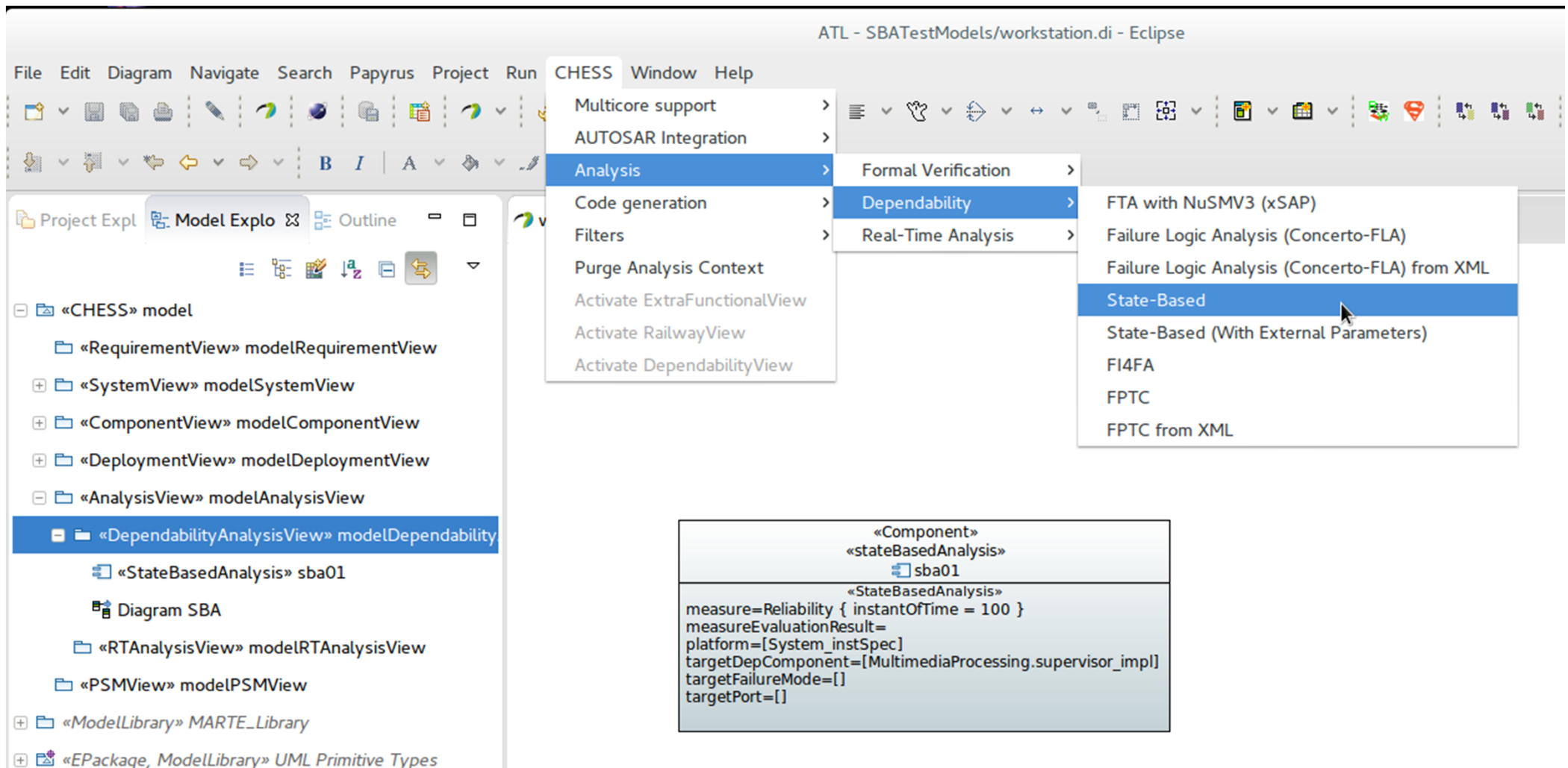


The screenshot shows the 'Preferences' dialog box for the 'StateBased' plugin. The left sidebar lists various categories, with 'StateBased Preferences' selected. The main area displays several configuration parameters:

Parameter	Value
DEEM Server Host (hostname or IP address)	rclserver.dsi.unifi.it
DEEM Server Port	5903
Connection timeout (seconds)	60
Minimum Batches	1000000
Maximum Batches	10000000
Confidence Interval (Relative)	0.1
Confidence Level	0.99
Display Update Interval	100
Measure Update Interval	100
Simulation Seed	6657

Running the analysis

- Enter the “DependabilityAnalysis” view to enable the state-based analysis plug-in
- Select CHES → Analysis → Dependability → State-Based



The screenshot shows the CHES Eclipse IDE interface. The menu path is: CHES -> Analysis -> Dependability -> State-Based. The Project Explorer on the left shows the project structure with the «DependabilityAnalysisView» modelDependability view selected. The Properties view on the right shows the configuration for the «StateBasedAnalysis» component.

```

«Component»
«stateBasedAnalysis»
    sba01
«StateBasedAnalysis»
measure=Reliability { instantOfTime = 100 }
measureEvaluationResult=
platform=[System_instSpec]
targetDepComponent=[MultimediaProcessing.supervisor_impl]
targetFailureMode=[]
targetPort=[]
    
```



CHESS

Simulator Configuration

Simulator Configuration

- The simulator can be configured to tune the accuracy of results
 - ◆ Minimum batches – minimum measure samples that the simulation will collect
 - ◆ Maximum batches – maximum measure samples that the simulation will collect
 - ◆ Confidence interval (relative) – interval (relative to the measure's mean) in which the real value of the measure is likely to fall
 - ◆ Confidence level – requested probability that the real value of the measure will fall in the given confidence interval
 - ◆ Display update interval – number of samples after which the display is updated
 - ◆ Measure update interval – number of samples after which the measure is updated
 - ◆ Simulation seed – seed used to initialize the simulation. Different seeds yield different simulation paths

Simulator Configuration: Example

- The main purpose of configuring those variables is to set the stopping conditions for the execution of the simulator
- Default values are the following
 - ◆ Confidence level: 99%
 - ◆ Confidence interval: relative, 1%.
 - ◆ Minimum number of samples: 1000
 - ◆ Maximum number of samples: 1000000
 - ◆ Measure update interval: 100
- These settings imply that
 - ◆ The simulator will stop and provide a result only when, for every computed measure:
 - 99% samples of the evaluated measure fall within an interval of $\pm 1\%$ from the computed mean, and
 - at least 1000 samples have been collected
 - ◆ Or
 - more than 1000000 samples have been collected
 - ◆ ...and this check is performed every 100 samples

Tuning of Simulator Configuration

- Default values will usually work well
- However, you can change these parameters to improve the result or to speedup the simulation. For example
 - ◆ Increase the confidence level or decrease the confidence interval
 - to get a more accurate result
 - ◆ Increase the minimum number of batches
 - if you get strange results that you did not expect (e.g. a zero value that should be something else)
 - ◆ Decrease the minimum batches
 - for the simulation to end earlier (in some cases it may provide wrong results!)
 - ◆ Decrease the maximum batches
 - to force the simulation to stop after a given amount of time (however, results may not converge with the requested precision anymore!)

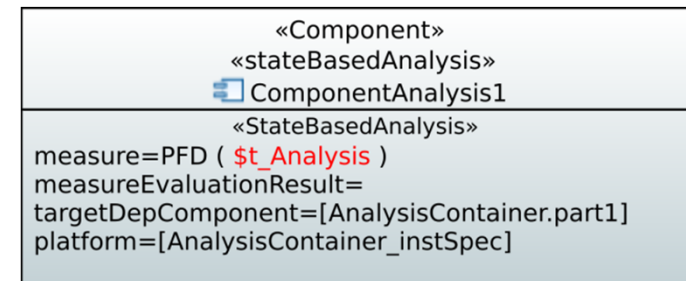
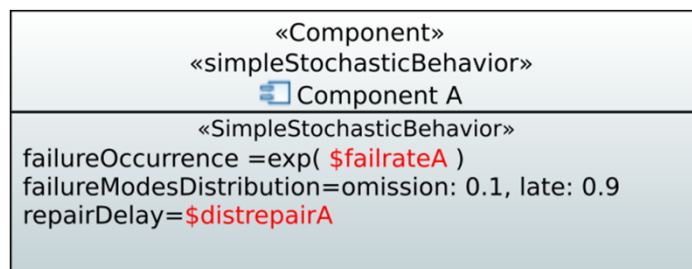


CHESS

Parametric and Periodic Analysis

Parametric Analysis

- Instead of adding values directly in the model, parameters can be used instead
 - ◆ Actual values of parameters are set at run-time
- Specification
 - ◆ In the UML model, parameters are specified with a dollar sign, e.g., \$FailureOccurrenceA



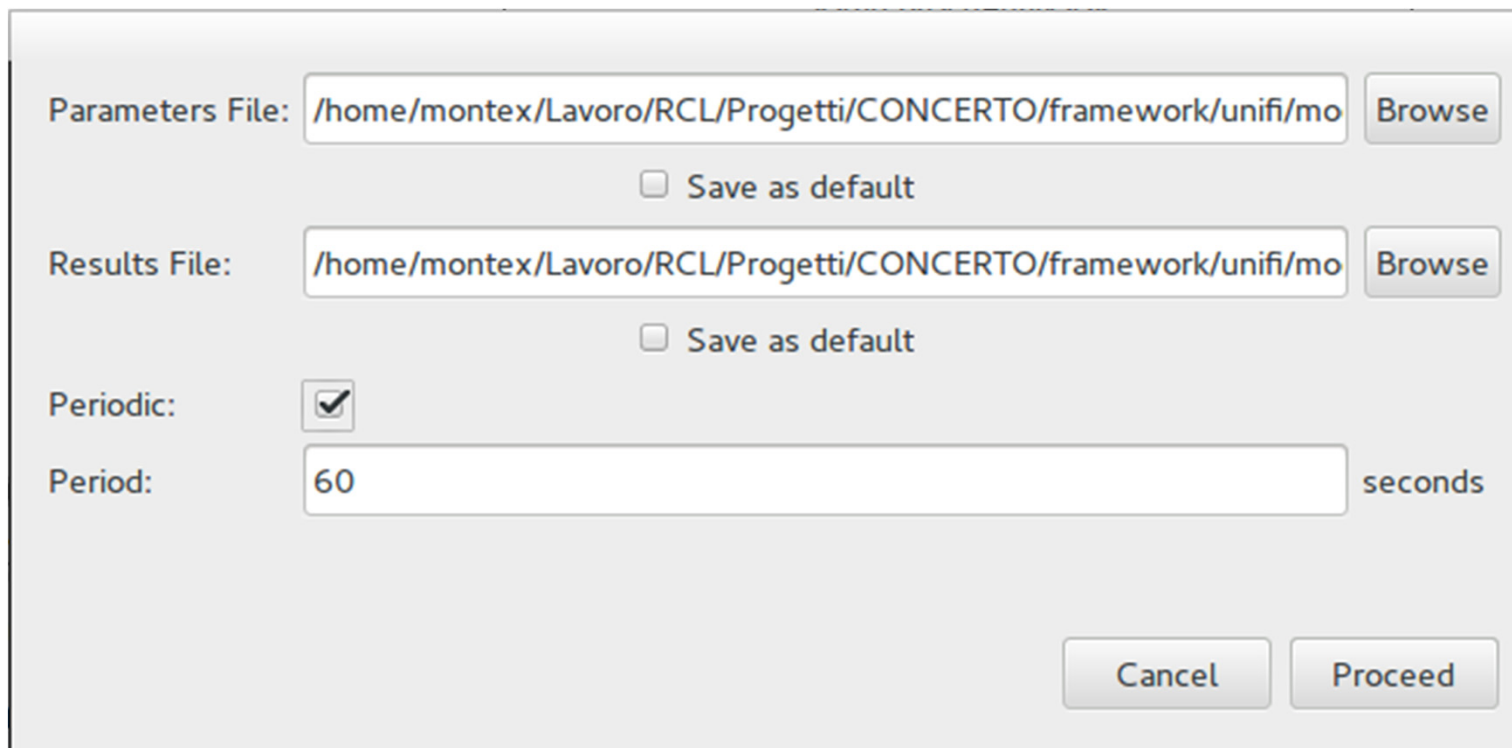
- Values
 - ◆ Are given in a textual file, in the format


```

$Parameter1      value
$Parameter2      value....
                    
```
 - ◆ Values are directly replaced in the model, they can thus be numbers, but also strings, or anything is used as an attribute in the model

Parametric Analysis (2)

- Run with the alternate menu option
 - ◆ State-Based Analysis (With External Parameters)
- A dialog appears allowing you to
 - ◆ Set the parameters file
 - ◆ Set the results file where output should be saved
 - ◆ (Optionally) define a periodic execution of the analysis



The dialog box contains the following fields and controls:

- Parameters File:** A text input field containing the path `/home/montex/Lavoro/RCL/Progetti/CONCERTO/framework/unifi/mo` and a **Browse** button to the right.
- Save as default
- Results File:** A text input field containing the path `/home/montex/Lavoro/RCL/Progetti/CONCERTO/framework/unifi/mo` and a **Browse** button to the right.
- Save as default
- Periodic:** A checkbox that is checked.
- Period:** A text input field containing the value `60`, followed by the text `seconds`.
- Buttons:** **Cancel** and **Proceed** buttons at the bottom right.



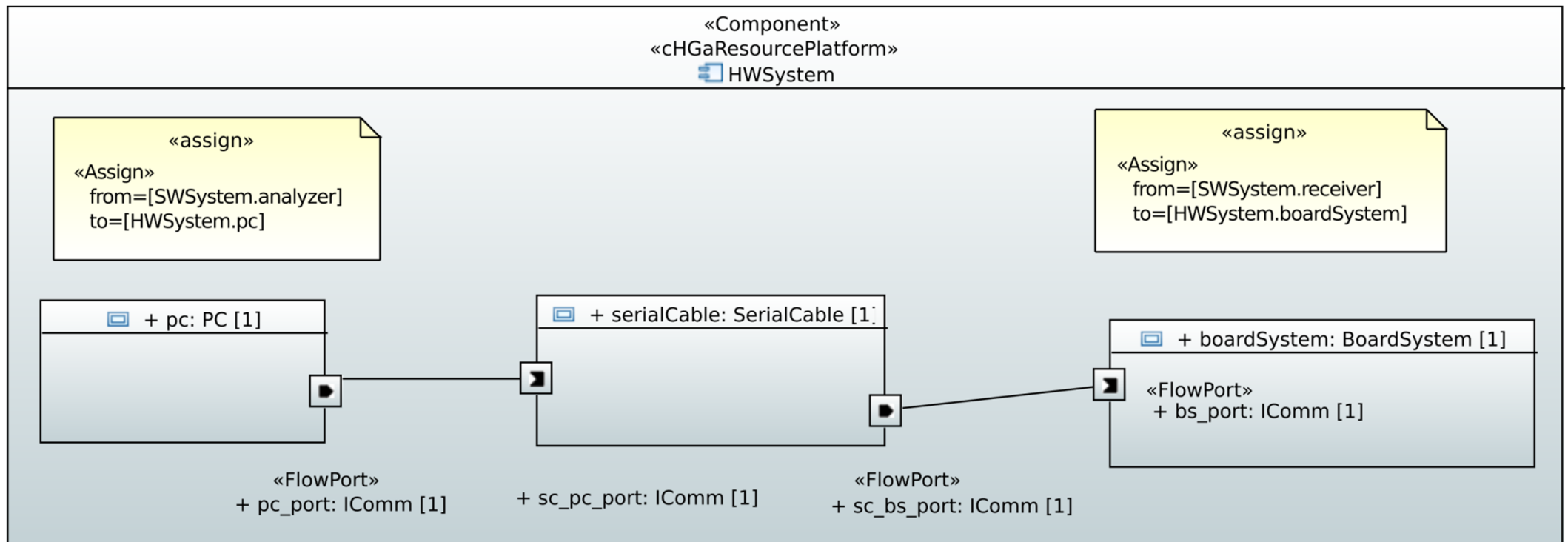
CHESS

Example

Simple Example (from the Railway domain)

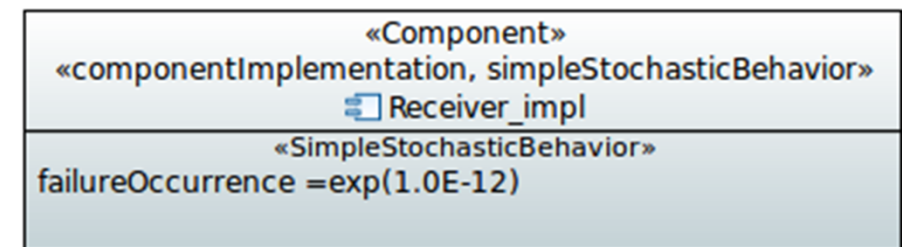
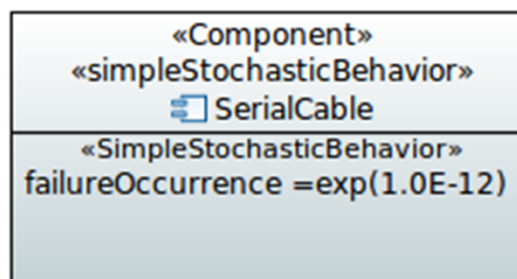
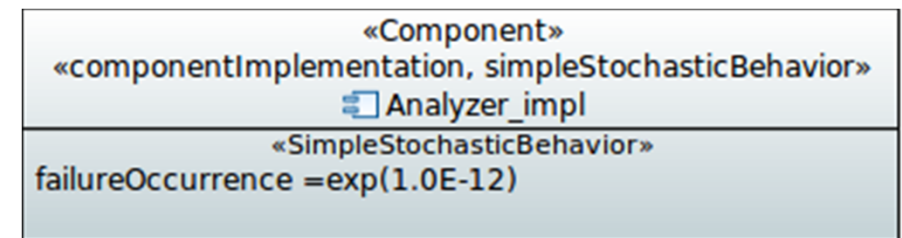
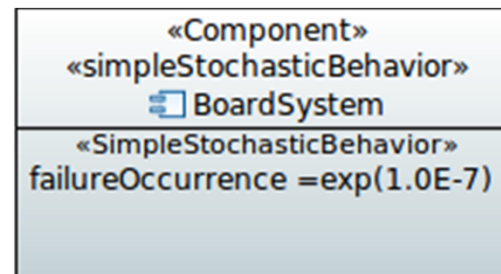
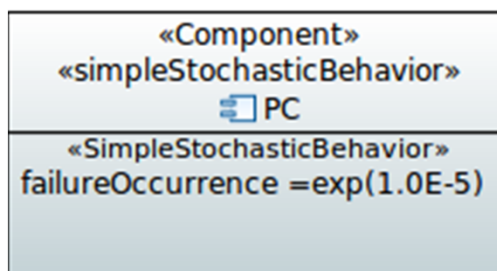
- Two software components
 - ◆ "Analyzer"
 - ◆ "Receiver"
- Three hardware components
 - ◆ "PC"
 - ◆ "SerialBus"
 - ◆ "BoardSystem"
- The "Receiver" component uses a service provided by the "Analyzer" component
- The "Analyzer" software component is allocated on the "PC" hardware component
- The "Receiver" software component is allocated on the "BoardSystem" hardware component
- We are interested in the instant-of-time reliability of the "Receiver" software component

Example – Functional Model

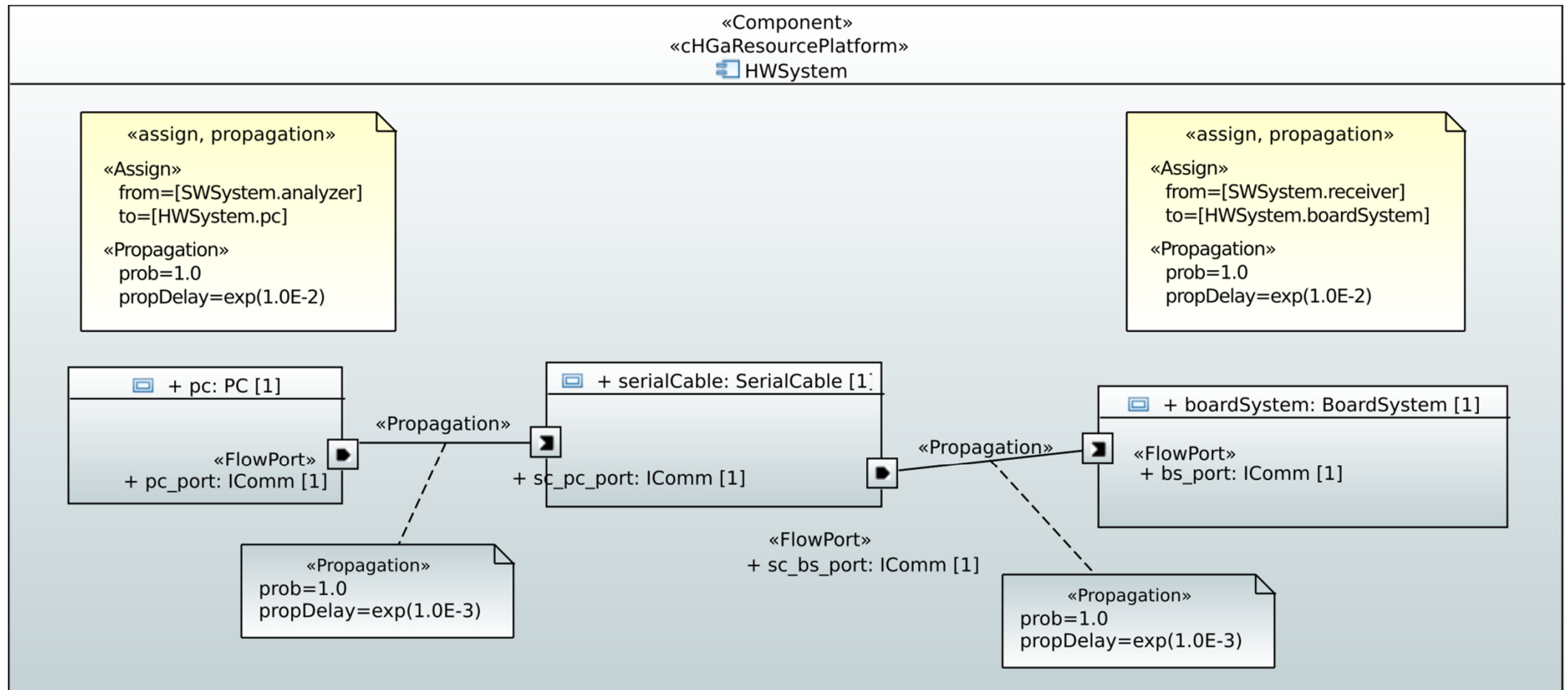


Example – Components enriched with dependability information

- Components are enriched with dependability information
- Note the fault occurrence rate for the different components
 - ◆ 10^{-12} faults/hour for software components ("Analyzer" and "Receiver") and for the "SerialBus"
 - ◆ 10^{-5} faults/hour for the "PC"
 - ◆ 10^{-7} faults/hour for the "BoardSystem"



Example – Enriched functional model



Evaluation and interpretation of the resulting measure(s)

- Measure
 - ◆ Reliability of the “Receiver” component at time $t=10.000=10^4$ hours
- Since the “Receiver” component has a fault occurrence rate of 10^{-12} faults/hour, its reliability at time $t = 10^4$ hours would be expected to be very near to 1
 - ◆ Actually, something like 0.99999...
- We obtained the following result:
 - ◆ **Reliability { instantOfTime = 10000 } = 0.9126**

How should this result be interpreted?

- The reliability of the “Receiver” component *is lower than expected because it is affected by the propagation* coming from both:
 - ◆ The hardware on which it is allocated (having a greater fault occurrence rate: 10^{-7})
 - ◆ The “Analyzer” software component, which is in turn affected by propagation coming from the “PC” hardware where it is allocated (having a lower fault occurrence rate: 10^{-5})



CHESS

Frequently Asked Questions

FAQs

- How do I specify the unit of measurements?
 - ◆ It is not needed, it is assumed that the unit of measurement is implicit, and should be the same across the model
- Can I use multiple «StateBasedAnalysis» stereotypes to evaluate multiple metrics?
 - ◆ Yes, provided that all of them refer to the same resources platform in their “platform” attribute
- How do I know if the result did not reach the specified confidence level?
 - ◆ In case the specified confidence interval and level are not reached, a mark (*) is added next to the result
- What if don't want to differentiate between multiple failure modes?
 - ◆ You can just use a placeholder name for the single failure mode, like “failure”