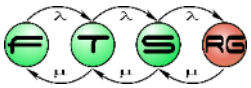
The Modeling and Analysis of an Industrial Protocol in the Gamma Framework

Bence Graics

Budapest University of Technology and Economics

Fault Tolerant Systems Research Group



Overview

Communication protocol - Orion

- To be used in the railway industry
- Master – slave protocol

Correct behavior is important

- Modeling and V&V are necessary
  - Modeling and analysis languages
  - Modeling and analysis tools

*Tool support for high-level modeling **and** analysis?*

Overview of the V&V process

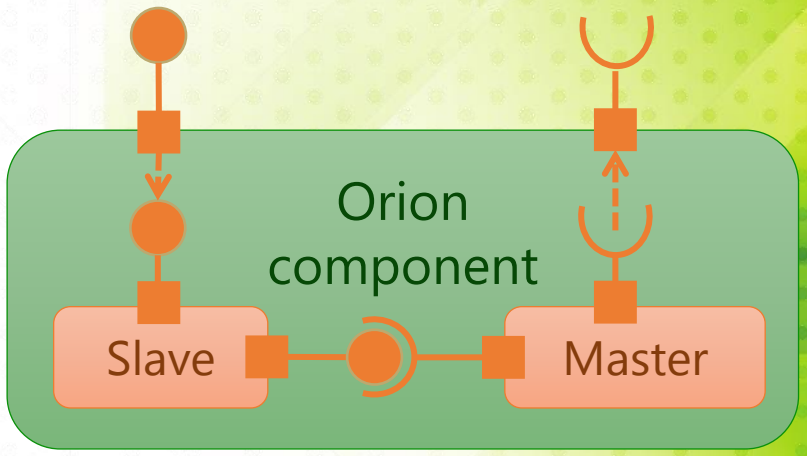
High-level modeling language

Support for component-based, hierarchical design

State-based (statecharts)

Formal semantics

- Enable formal analysis



Modeling and analysis tool

Validation during modeling

Formal verification of the finished model (model checking)

- Automatic model transformations
- Model checking based on CTL expressions (supported by templates)

(Automatic derivation and deployment of implementation)

Overview of the V&V process

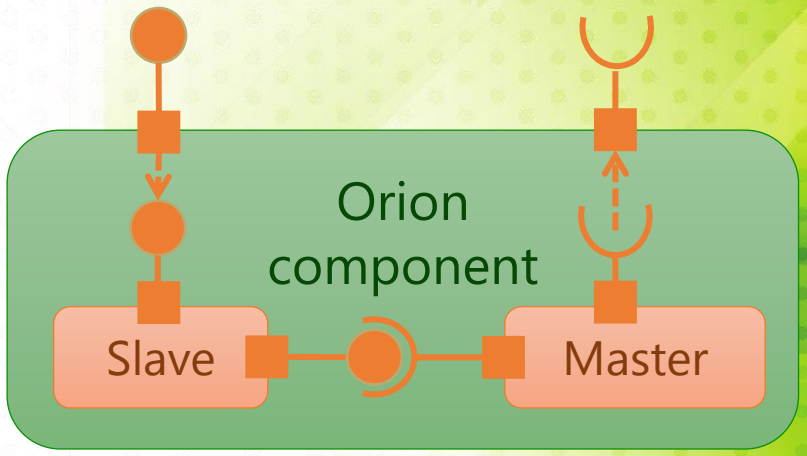
High-level modeling language

Support for component-based, hierarchical design

State-based (statecharts)

Formal semantics

- Enable formal analysis



Modeling and analysis tool

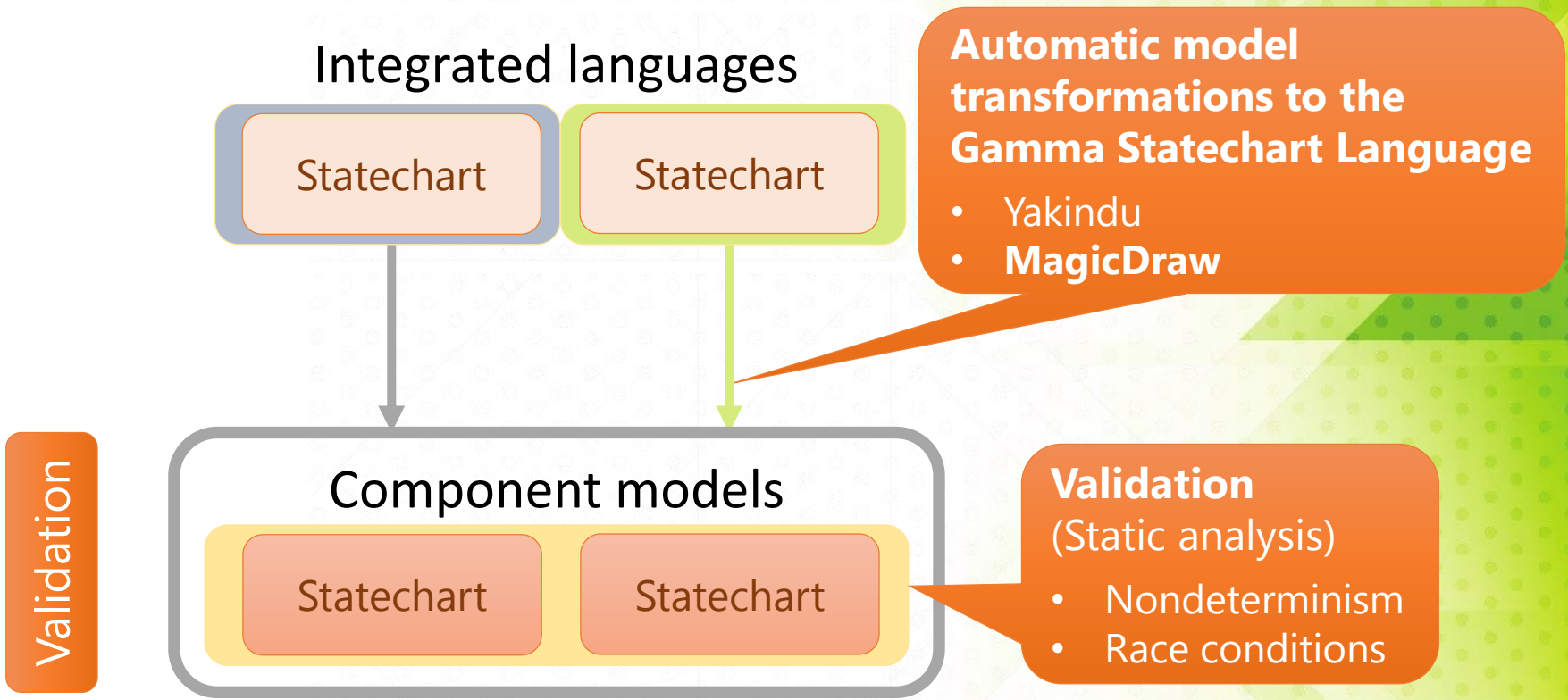
Validation during modeling

Formal verification of the finished model (model checking)

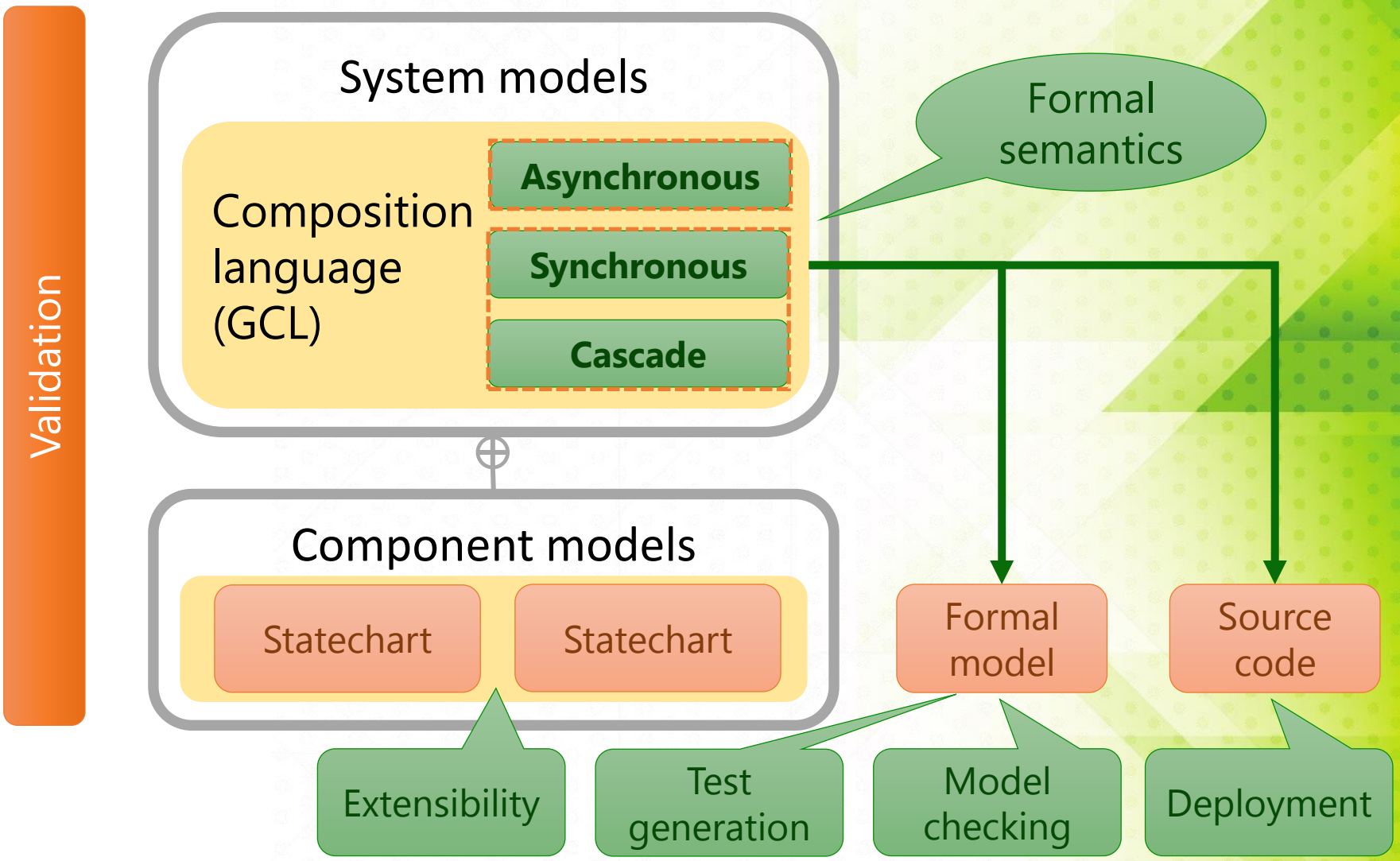
- Automatic model transformations
- Model checking based on CTL expressions (supported by templates)

Gamma Statechart Composition Framework

Gamma framework – components



Gamma framework – composition



Synchronous systems

Coherent unit with a well-defined function

Communication based on signals via ports

- No message queues

Execution is initiated by a clock signal

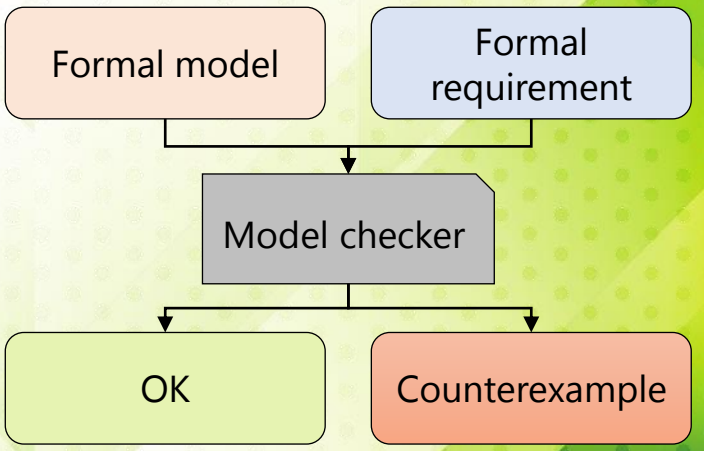
Synchronous

- Concurrent execution
- Components sample incoming signals at the same time (global clock)
  - Execution order is not relevant

Cascade

- Sequential execution
- Components sample incoming signals right before execution
  - Execution order is relevant

Functionalities



Verification

Model checking

– Systematic traversal of the state space (UPPAAL)

Test generation

– Applicable for testing the model implementation

Code generation

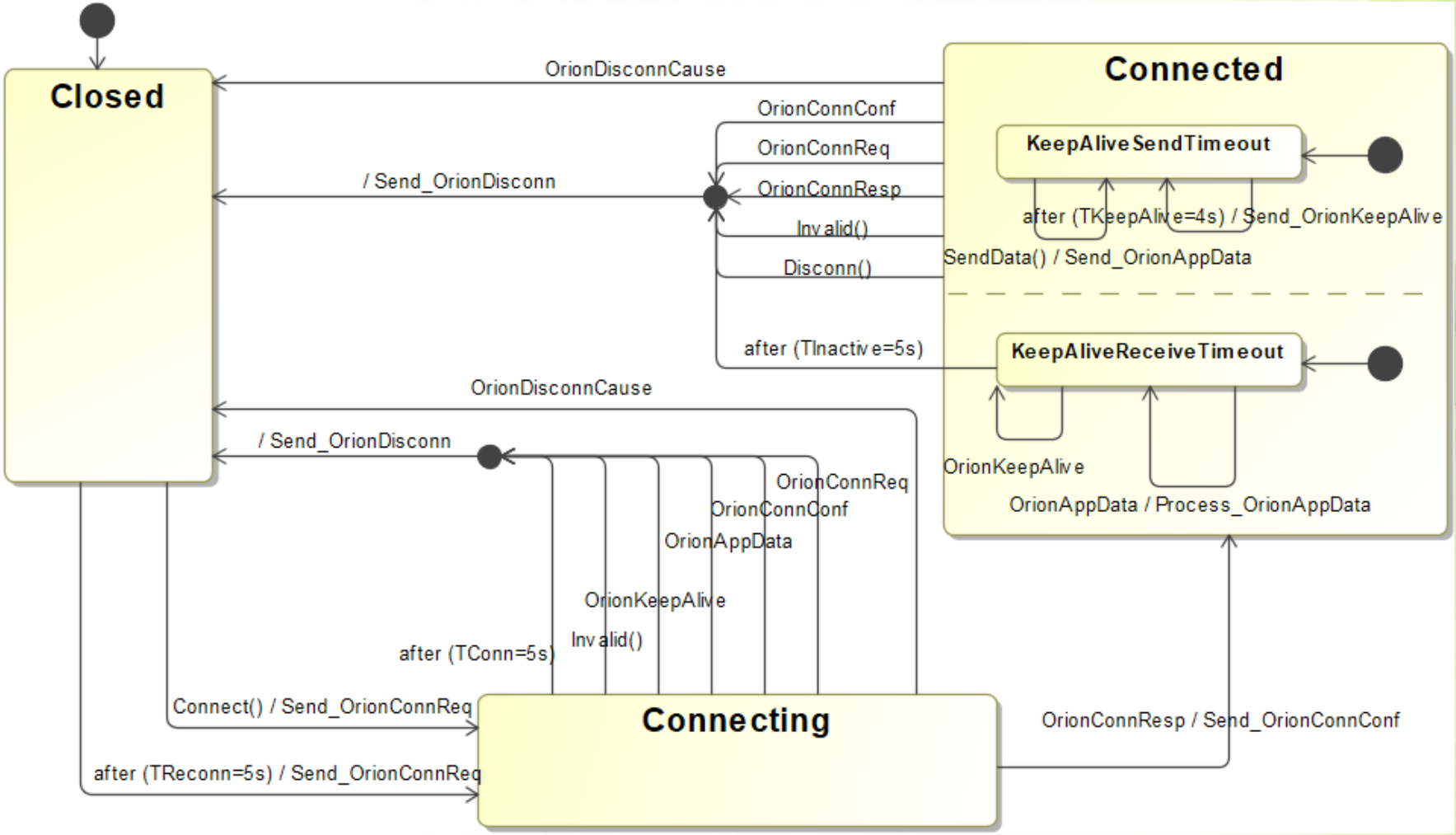
With threads or over DDS

– Automatic deployment

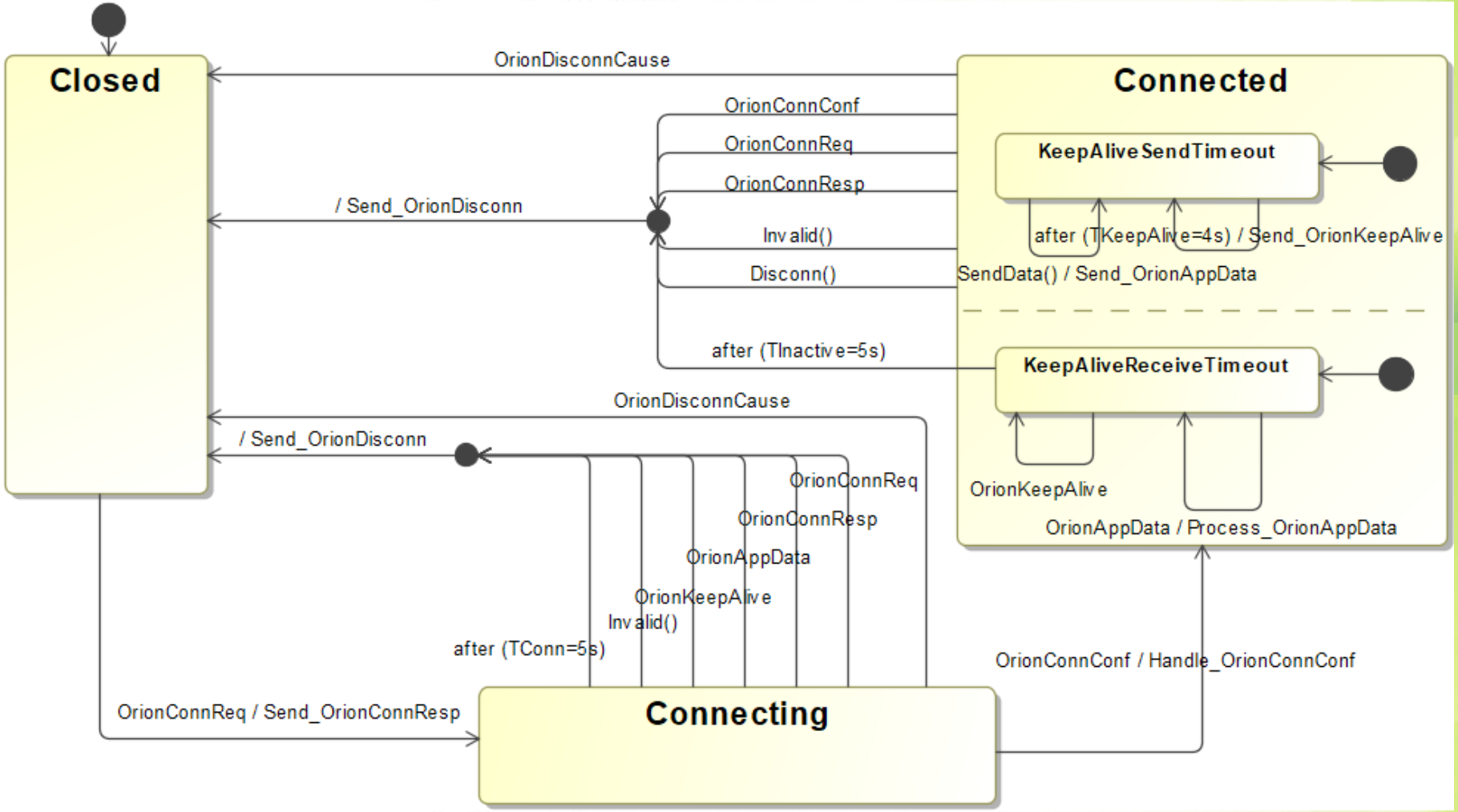
Validated case-by-case with generated test set

Case Study

Orion protocol - master



Orion protocol - slave



Creating the Gamma models

The master and slave are modeled in MagicDraw

They have to be transformed to Gamma statecharts (automatic)

Gamma synchronous models have to be created

Protocol documentation – *1)* components are *executed one after another, 2)* execution is *triggered by a clock* → cascade mode

Communication between components

1. Ideal channel – events are **not** lost and arrive immediately
2. Event losing channel – a finite number of events can be lost (modeled in Yakindu and transformed to Gamma)
3. Delay channel – the transmission of events is delayed with (at most) a constant time (modeled in Yakindu and …)

Requirements to be checked

1. *The system can reach a state in which both the master and the slave are in state Connected*

   EF master.Connected && slave.Connected

2. *The system must eventually reach a state in which both the master and the slave are in state Connected*

   AF master.Connected && slave.Connected

System model – ideal channel

```
cascade OrionSystemIdealChannel [
    // Ports visible from the envrionment
    port masterConnection : requires Connect
    port slaveConnection : requires Connect
] {
    // Definition of the master and slave components
    component master : OrionMaster
    component slave : OrionSlave
    // Binding Connect, Disconnect signals to component ports
    bind masterConnection -> master.Connection
    bind slaveConnection -> slave.Connection
    // Connecting OrionConnReq, OrionConnResp, ... signals
    channel [master.SendOrion] -o)- [slave.ReceiveOrion]
    channel [slave.SendOrion] -o)- [master.ReceiveOrion]
}
```

Analysis of ideal channel model

1. *The system can reach a state in which both the master and the slave are in state Connected*

   Requirement is met!

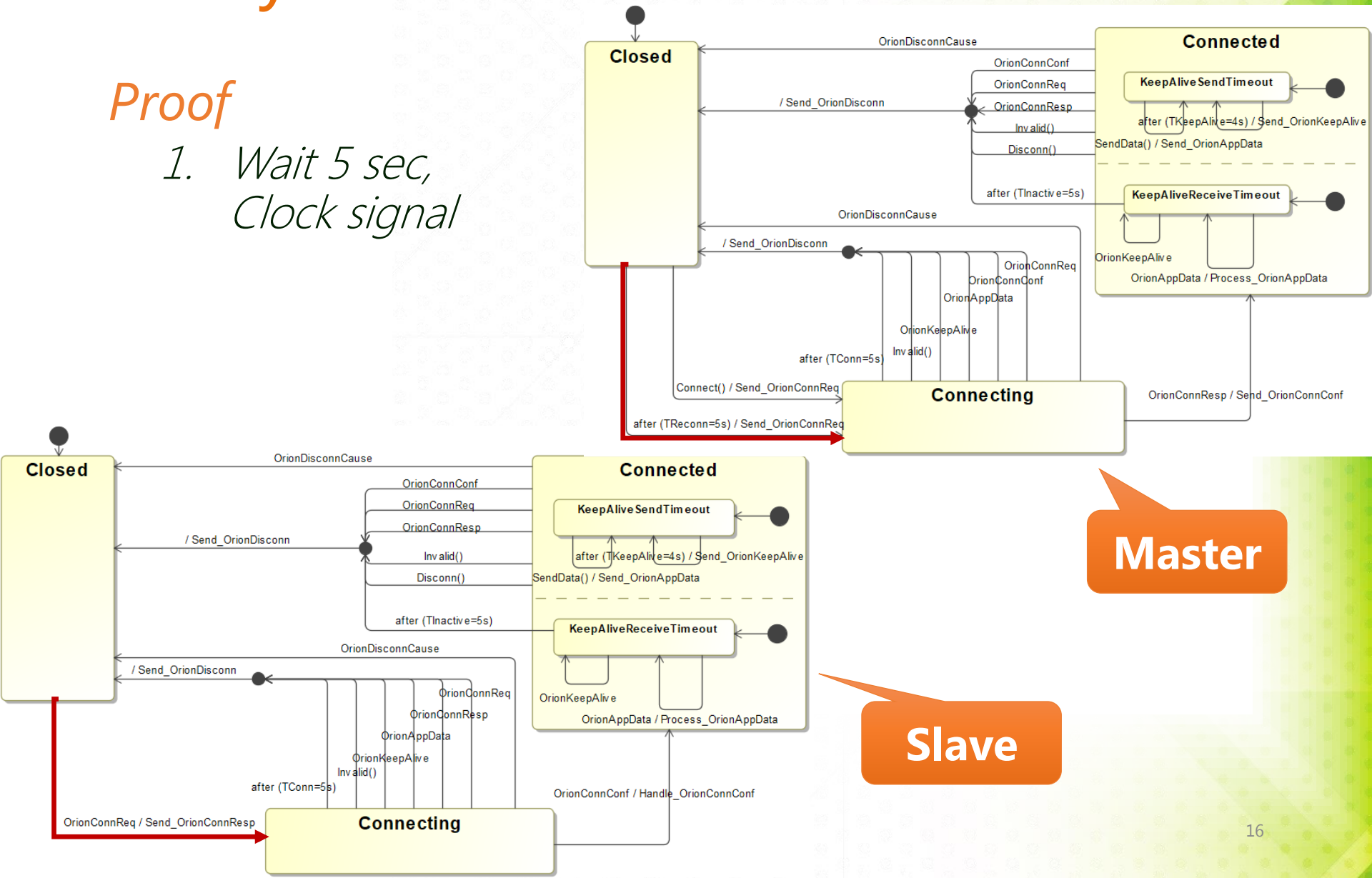
Proof

1. *Wait 5 sec,*
   Clock signal
2. *Clock signal*

Analysis of ideal channel model

Proof

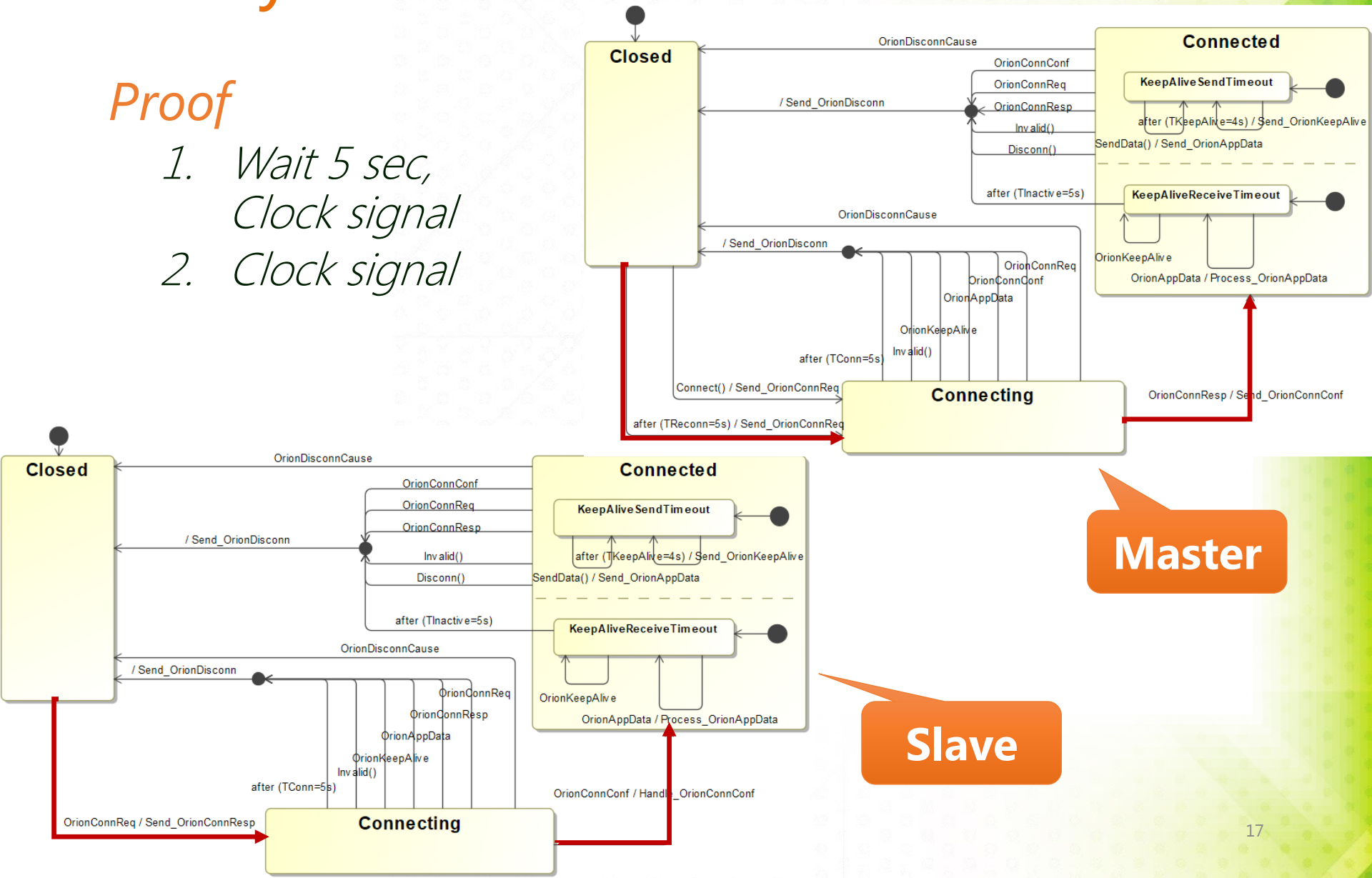
1. *Wait 5 sec, Clock signal*



Analysis of ideal channel model

Proof

1. *Wait 5 sec, Clock signal*
2. *Clock signal*



Analysis of ideal channel model

2. *The system must eventually reach a state in which both the master and the slave are in state Connected*

*Requirement is **not** met!*

Counterexample

Loop:

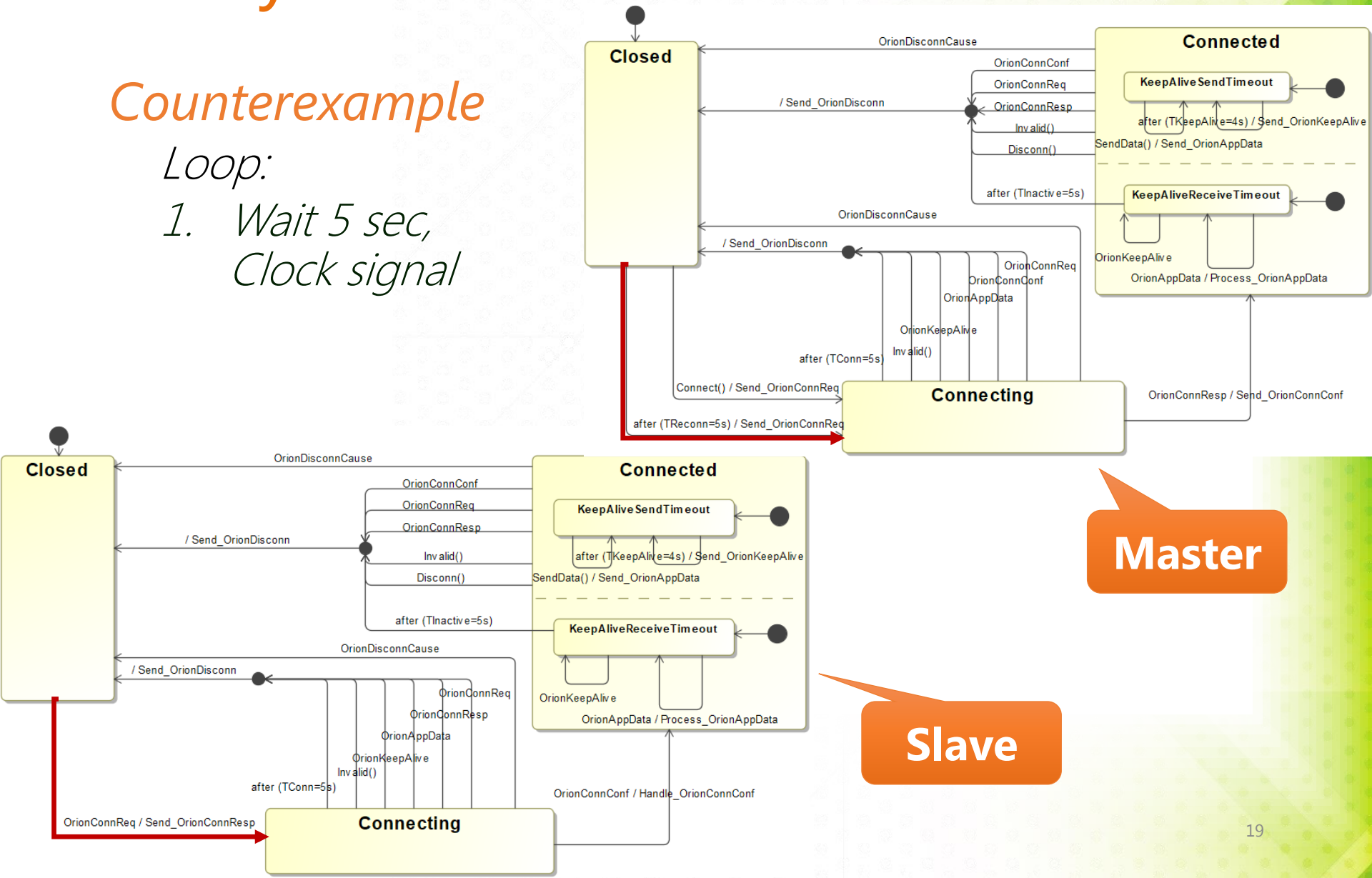
1. *Wait 5 sec,*
   Clock signal
2. *Wait 5 sec*
   Clock signal

Analysis of ideal channel model

Counterexample

Loop:

1. *Wait 5 sec,*
   Clock signal

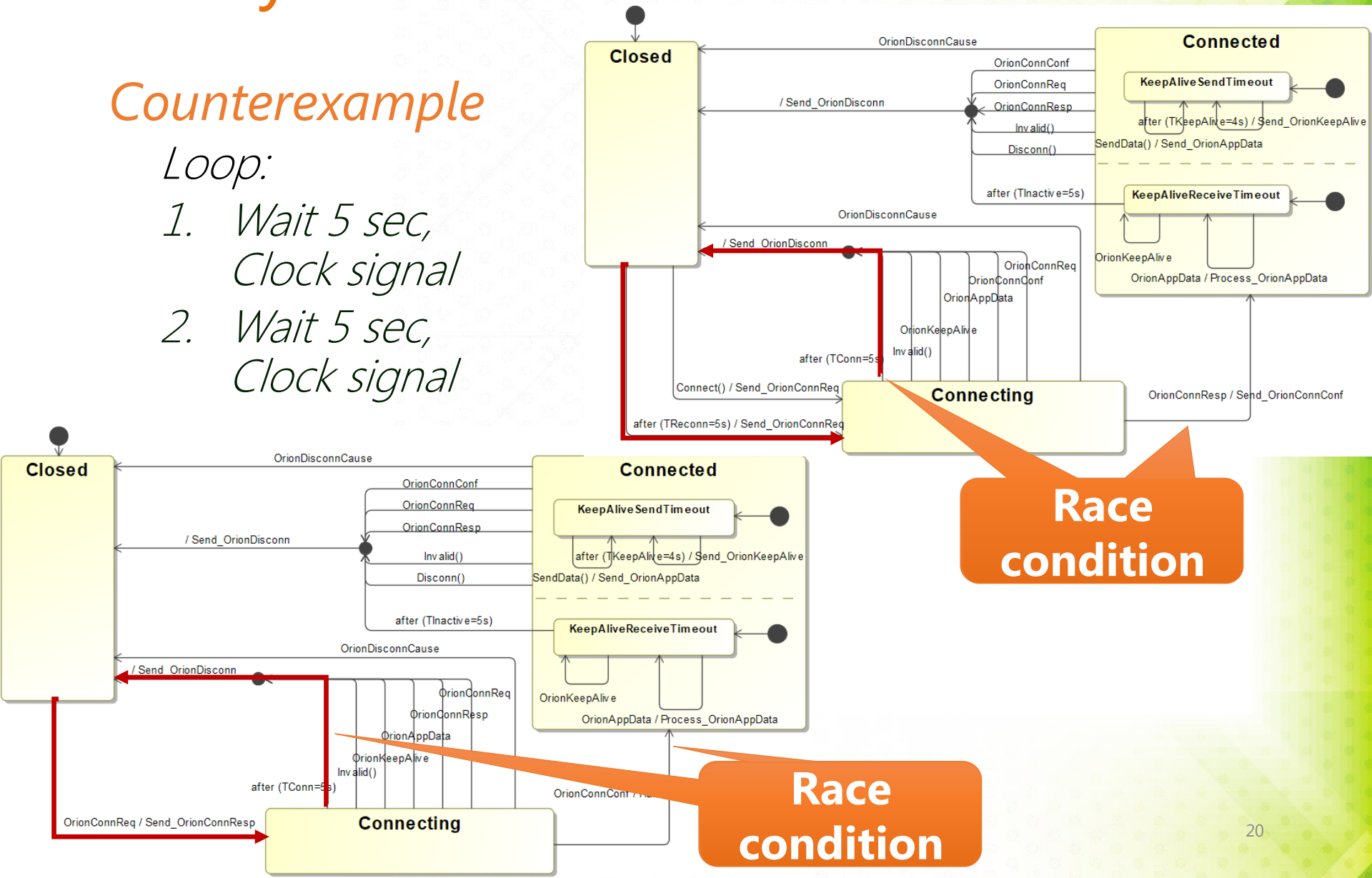


Analysis of ideal channel model

Counterexample

Loop:

1. *Wait 5 sec, Clock signal*
2. *Wait 5 sec, Clock signal*

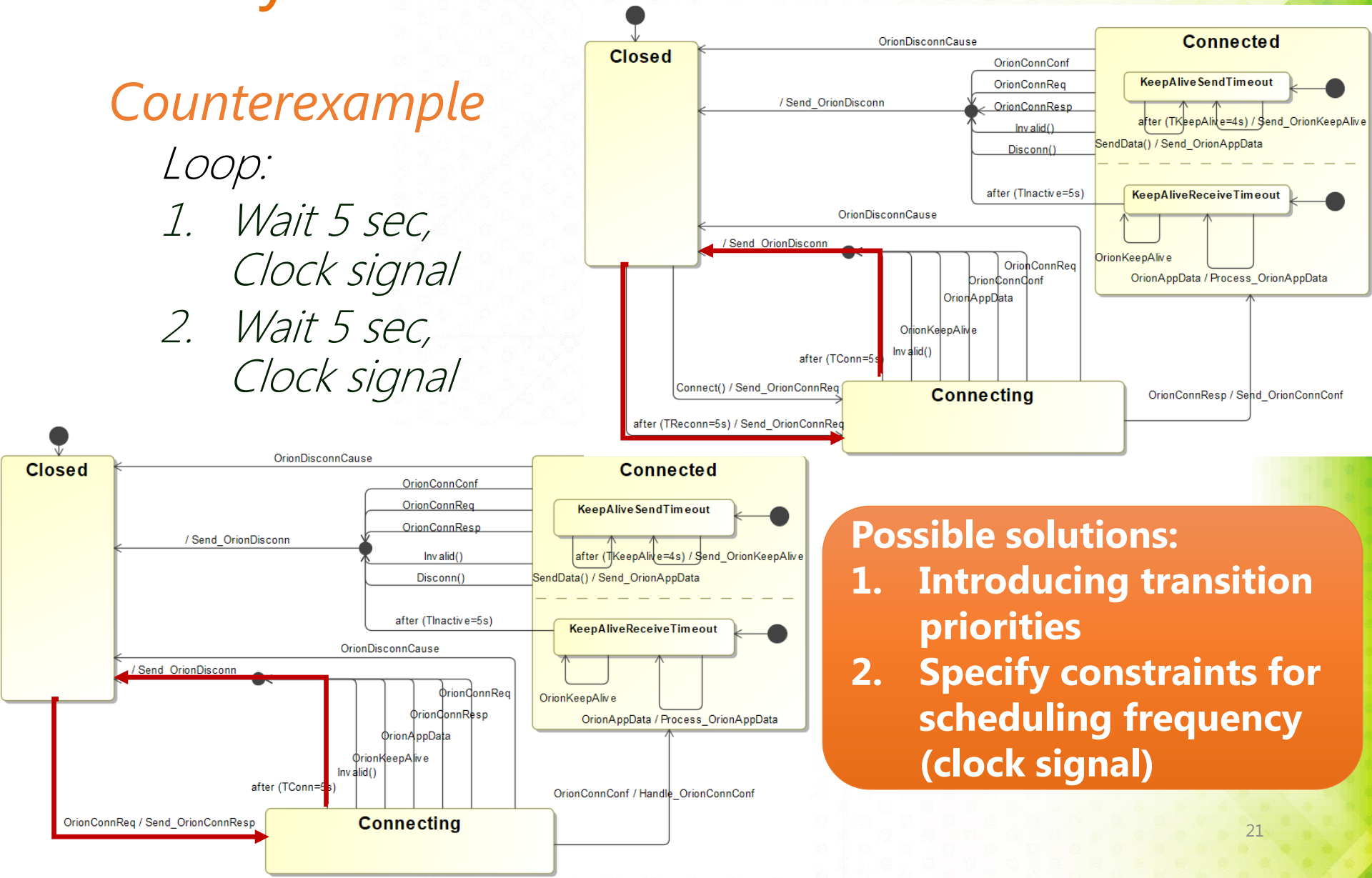


Analysis of ideal channel model

Counterexample

Loop:

1. *Wait 5 sec, Clock signal*
2. *Wait 5 sec, Clock signal*



Possible solutions:

1. **Introducing transition priorities**
2. **Specify constraints for scheduling frequency (clock signal)**

System model – event losing ch.

```
cascade OrionSystemEventLosingChannel [
    // Ports visible from the envrionment
    port masterConnection : requires Connect
    port slaveConnection : requires Connect
] {
    // Definition of the master, slave and channel components
    component master : OrionMaster
    component masterToSlaveChannel : Channel
    component slave : OrionSlave
    component slaveToMasterChannel : Channel
    ...
    // Connecting OrionConnReq, OrionConnResp, ... signals via channels
    channel [master.SendOrion] -o)- [masterToSlaveChannel.Input]
    channel [masterToSlaveChannel.Output] -o)- [slave.ReceiveOrion]
    channel [slave.SendOrion] -o)- [slaveToMasterChannel.Input]
    channel [slaveToMasterChannel.Output] -o)- [master.ReceiveOrion]
}
```

Gamma channel model: Maximum number of losable events - *constant*

System model – event losing ch.

```
cascade OrionSystemEventLosingChannel [
    // Ports visible from the envrionment
    port masterConnection : requires Connect
    port slaveConnection : requires Connect
] {
    // Definition of the master, slave and channel components
    component master : OrionMaster
    component masterToSlaveChannel : Channel
    component slave : OrionSlave
    component slaveToMasterChannel : Channel
    ...
    // Connecting OrionConnReq, OrionConnResp, ... signals via channels
    channel [master.SendOrion] -o)- [masterToSlaveChannel.Input]
    channel [masterToSlaveChannel.Output] -o)- [slave.ReceiveOrion]
    channel [slave.SendOrion] -o)- [slaveToMasterChannel.Input]
    channel [slaveToMasterChannel.Output] -o)- [master.ReceiveOrion]
}
```

Scheduling frequency: 1 / 450 ms

Analysis of event losing channel

1. *The system can reach a state in which both the master and the slave are in state Connected*

   Requirement is met!

Proof

1. *Wait 5 sec,*
   Clock signal
2. *Clock signal*

2. *The system must eventually reach a state in which both the master and the slave are in state Connected*

   Requirement is met (1, 2, …, 7 lost events)!

System model – event delay ch.

```
cascade OrionSystemEventDelayChannel [
    // Ports visible from the envrionment
    port masterConnection : requires Connect
    port slaveConnection : requires Connect
] {
    // Definition of the components
    component master : OrionMaster
    component masterToSlaveChannel : Channel
    component slave : OrionSlave
    component slaveToMasterChannel : Channel
    ...
    // Connecting OrionConnReq, OrionConnResp, ... signals via channels
    channel [master.SendOrion] -o)- [masterToSlaveChannel.Input]
    channel [masterToSlaveChannel.Output] -o)- [slave.ReceiveOrion]
    channel [slave.SendOrion] -o)- [slaveToMasterChannel.Input]
    channel [slaveToMasterChannel.Output] -o)- [master.ReceiveOrion]
}
```

Gamma channel model:
An event does *not* get transmitted until

- **an event with the same type is sent to the same channel, or**
- **a certain (constant) time elapses**

Analysis of event delay channel

If the event transmission delay is lesser than 2.5 s:

1. *The system can reach a state in which both the master and the slave are in state Connected*

   Requirement is met!

   Proof is similar to the previous ones

2. *The system must eventually reach a state in which both the master and the slave are in state Connected*

   Requirement is met!

Analysis of event delay channel

If the event transmission delay is greater than or equals to 2.5 s:

1. *The system can reach a state in which both the master and the slave are in state Connected*

   *Requirement is **not** met!*

   Specified by the 5 s timeout in state Connecting

2. *The system must eventually reach a state in which both the master and the slave are in state Connected*

   Requirement is not *met!*

Lessons learned

The definition of the execution semantics of the statechart is crucial

- Asynchronous statechart (UML, SysML)
- Synchronous statechart (Yakindu, Gamma)
  - Do transitions have priorities?
  - Is there a constraint on scheduling frequency?
    - It greatly reduces the size of the state space

Hard to verify the analysis tool

- Model checkers can reveal nondeterministic behavior ←→ implementation is deterministic
- Minor (now fixed) bugs in Gamma