Modelica code generation from ModelicaML state machines extended by asynchronous communication

***Uwe Pohlmann**, Matthias Tichy*

ENTIME

Project:
http://wwwhni.uni-paderborn.de/en/priority-projects/entime/

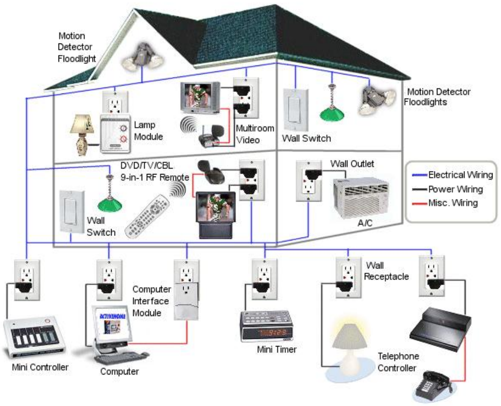
Funded by:

European Union
European Regional Development Fund
Investing in your future

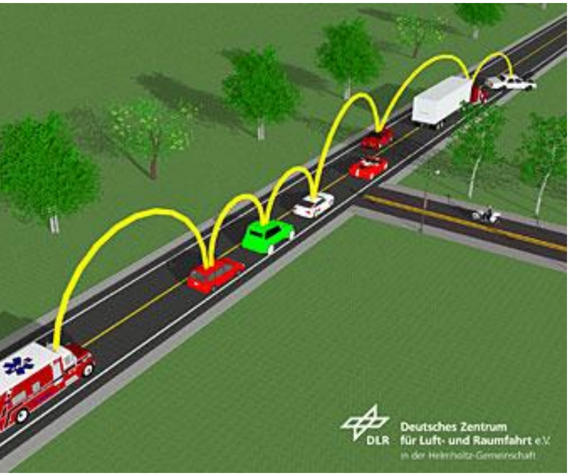
Ministry of Innovation, Science, Research and Technology of the German State of North Rhine-Westphalia

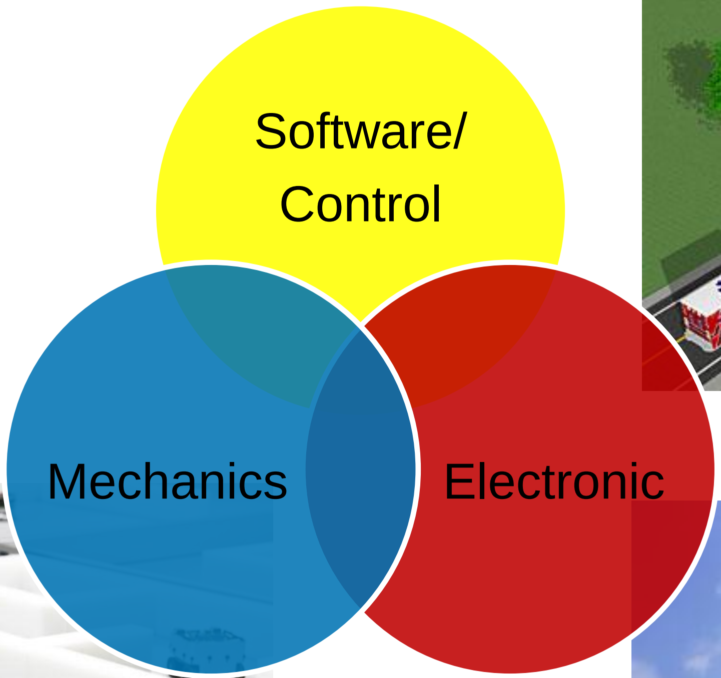
Intelligent Mechatronics Systems

Home network systems

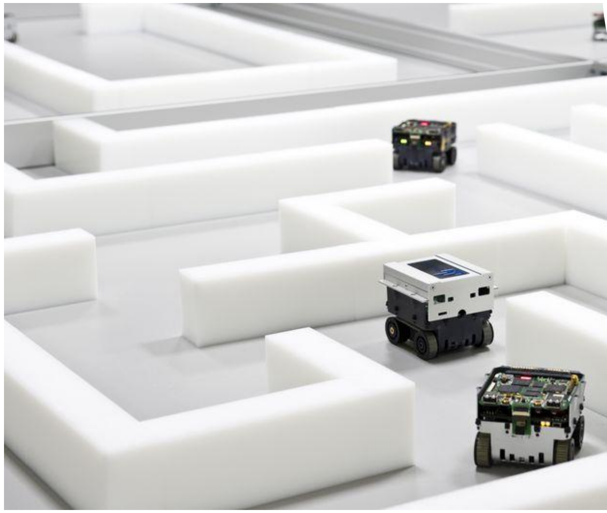


Car-to-car communication


DLR
Deutsches Zentrum für Luft- und Raumfahrt e.V.
in der Helmholtz-Gemeinschaft



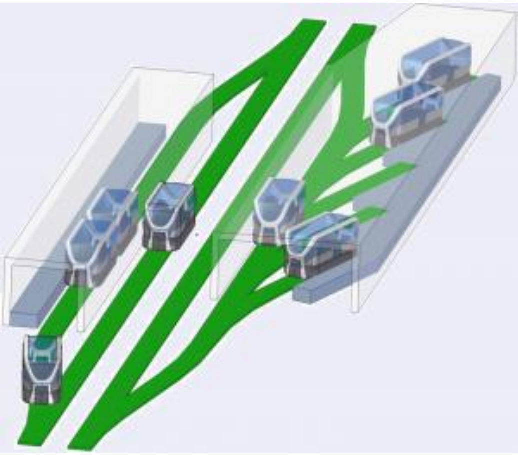
Robot swarms

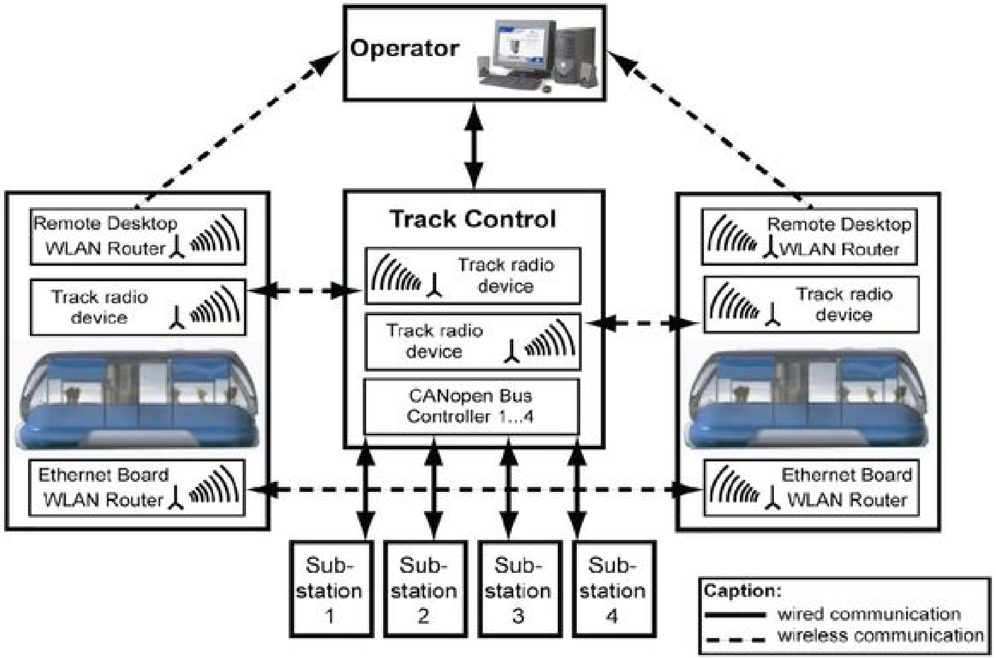


Railcabs

Example for Intelligent Mechatronics Systems 1/2

- Railcab shuttles are autonomous train systems
- Transport goods or people
- Form convoys to save energy
- Safety Critical System
- Hard real-time requirements



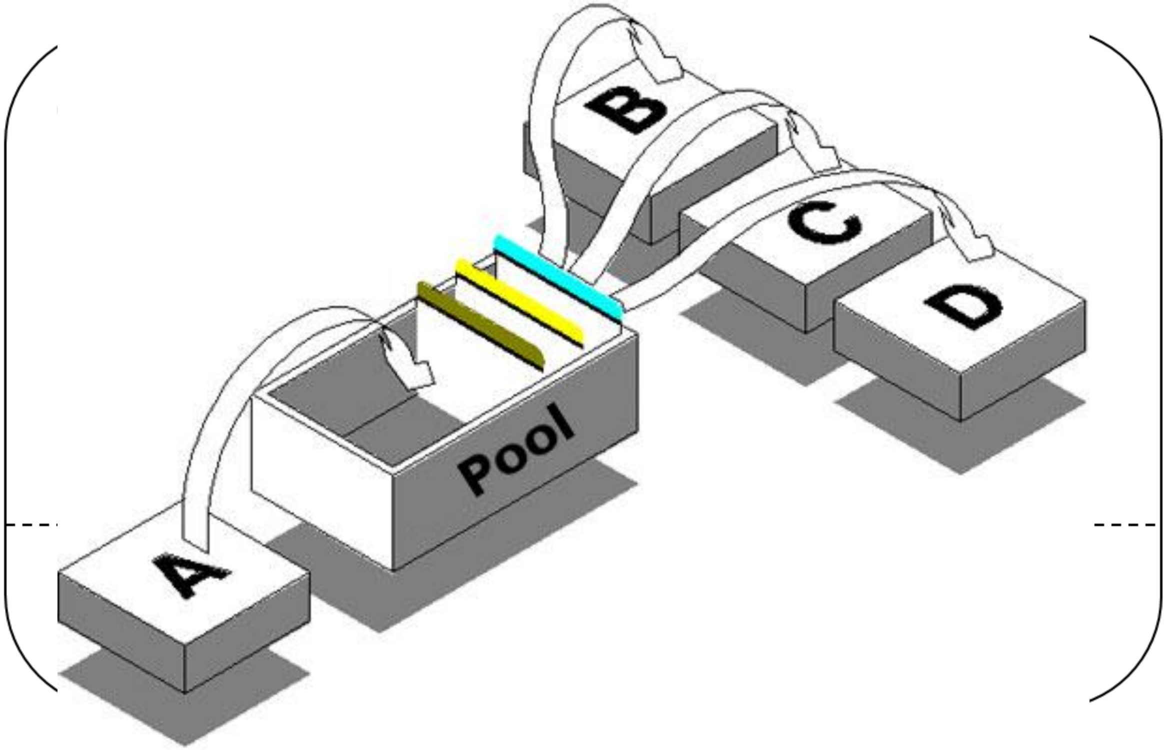


Source:
“Neue Bahntechnik Paderborn”
http://www.railcab.de/

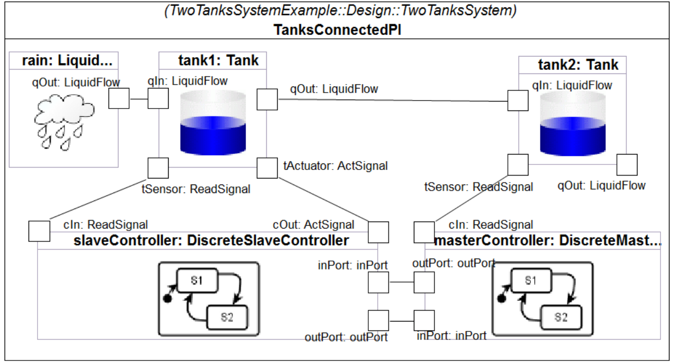
Example for Intelligent Mechatronics Systems 2/2

- Railcabs can dynamically form convoys
- Coordination by wireless connection

- Very complex coordination
- Message passing
  - Looses type of coupling
- State Machines define the communication protocol
- Message pools provide an asynchronous communication
  - Sender has not to wait for the receiver



Design Method for Intelligent Mechatronics

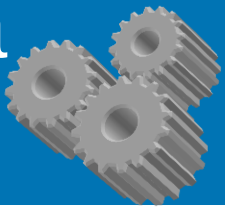


Modeling in ModelicaML

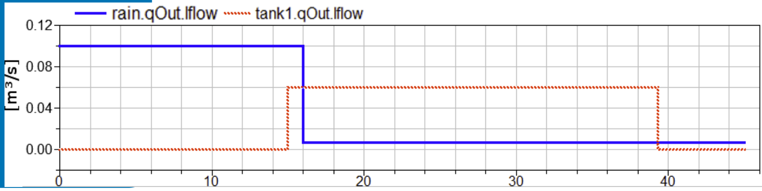
Physics/ Equations

Software/ Communication

Generate Modelica Code



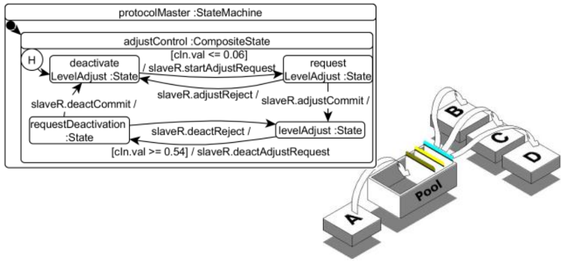
Simulate with Simulation Engine

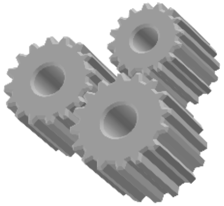


ModelicaML

- ModelicaML
  - Combination of UML and Modelica
  - Tight integration in mechatronic design
  - Close collaboration between different domain experts
  - Graphical Modeling of behavior by state machines
    - Appropriate modeling formalism
    - Describe discrete behavior of a system

- Contribution of this Paper
  - Extension of ModelicaML state machines with messages and message pools
    - Definition of syntax and semantics
  - Automatic transformation of ModelicaML state machines with messages to Modelica


protocolMaster :StateMachine
adjustControl :CompositeState
H
deactivate LevelAdjust :State
[cln.val <= 0.06] / slaveR.startAdjustRequest
request LevelAdjust :State
slaveR.adjustReject /
slaveR.deactCommit /
slaveR.adjustCommit /
requestDeactivation :State
slaveR.deactReject /
levelAdjust :State
[cln.val >= 0.54] / slaveR.deactAdjustRequest
A
Pool
B
C
D



Modelica code generation from ModelicaML state machines extended by asynchronous communication

1 Introduction

2 State Machine with Messages

3 Transformation to Modelica Code

4 Related Work

5 Conclusion

Example

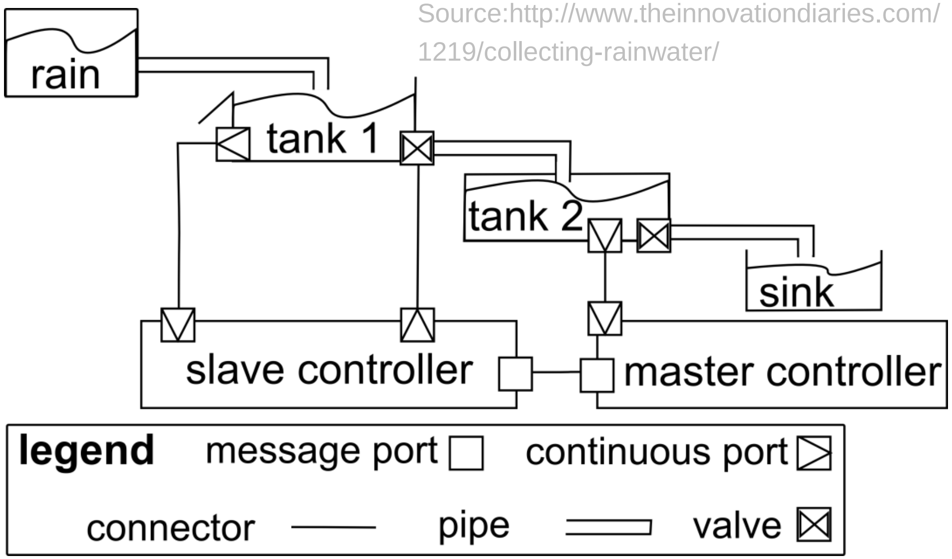
Use Cases

- Tank 1 collects rain
- User takes periodically water from tank 2
- Slave controller controlls valve of the pipe between the tanks
- Master controller asks the slave controller via messages to open the valve

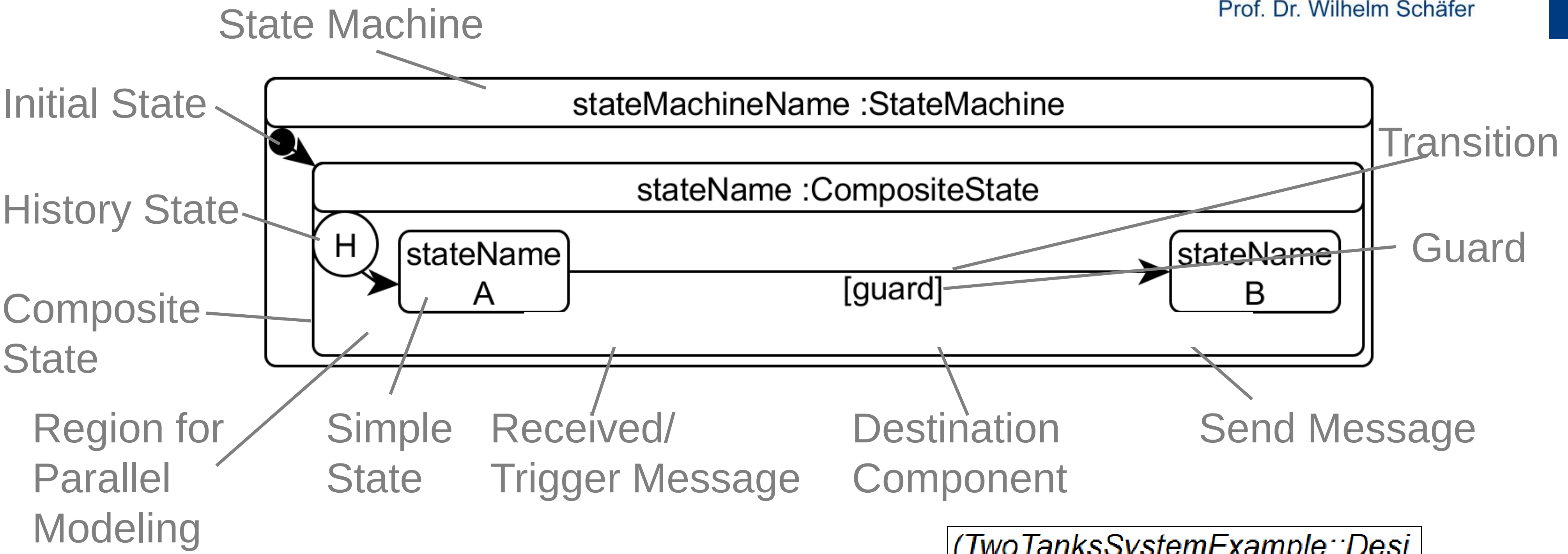
Structure

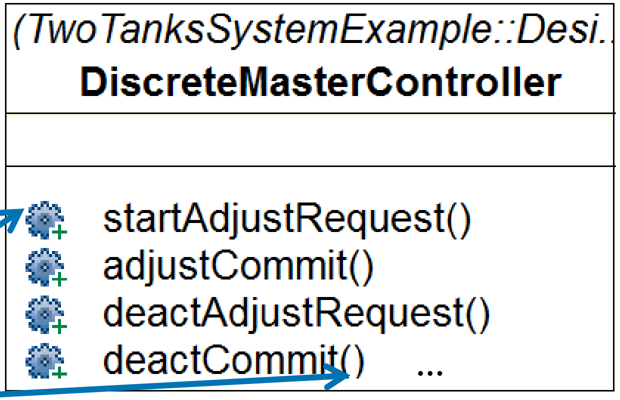
- Two tanks, controllers, sensors
- Pipe between the tanks
- Communication link between controller

Source:http://www.theinnovationdiaries.com/1219/collecting-rainwater/



State Machine and Message Syntax




(TwoTanksSystemExample::Desi...

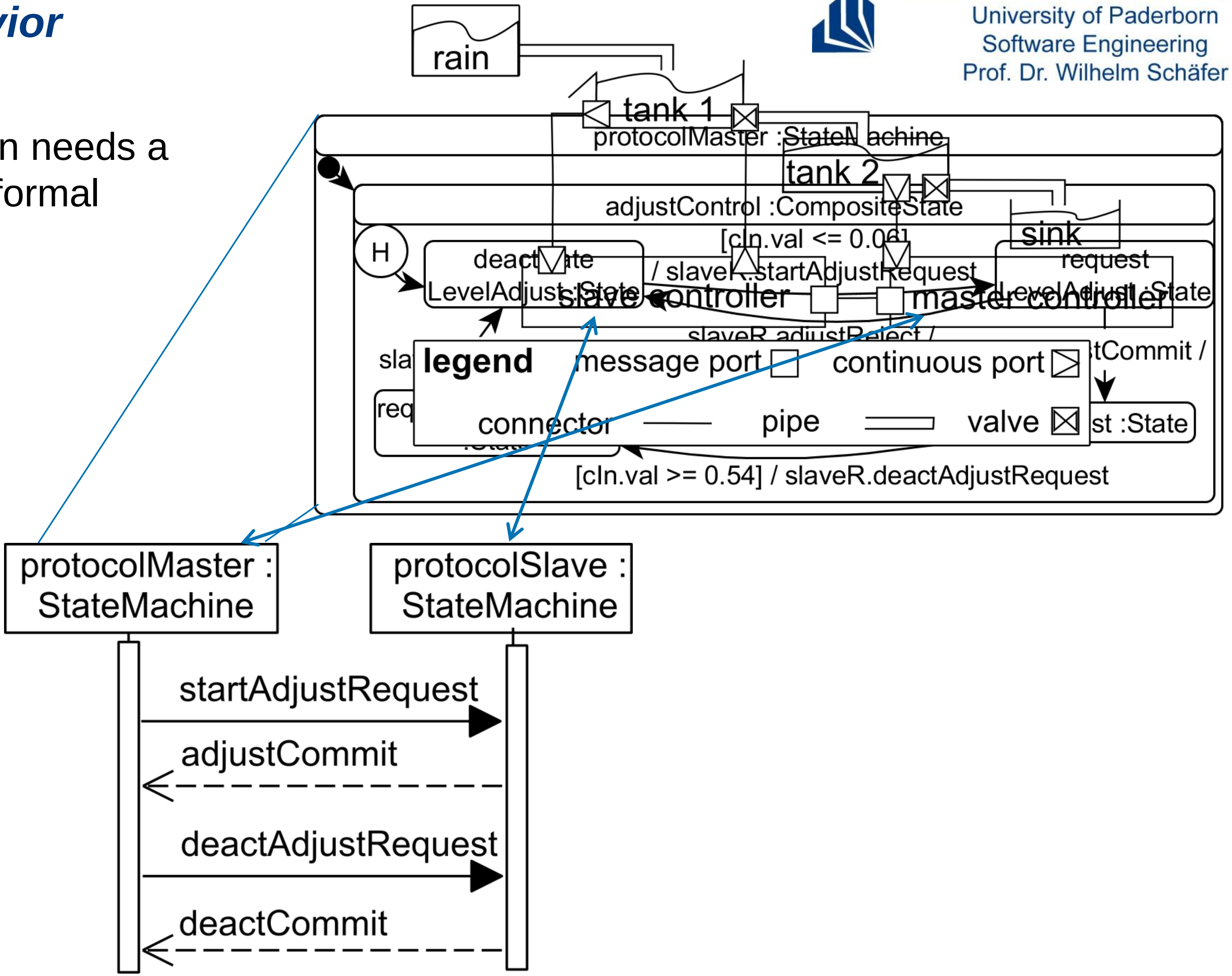
DiscreteMasterController

startAdjustRequest()
adjustCommit()
deactAdjustRequest()
deactCommit() ...

- Define message types as UML operations
- Messages types could have parameters

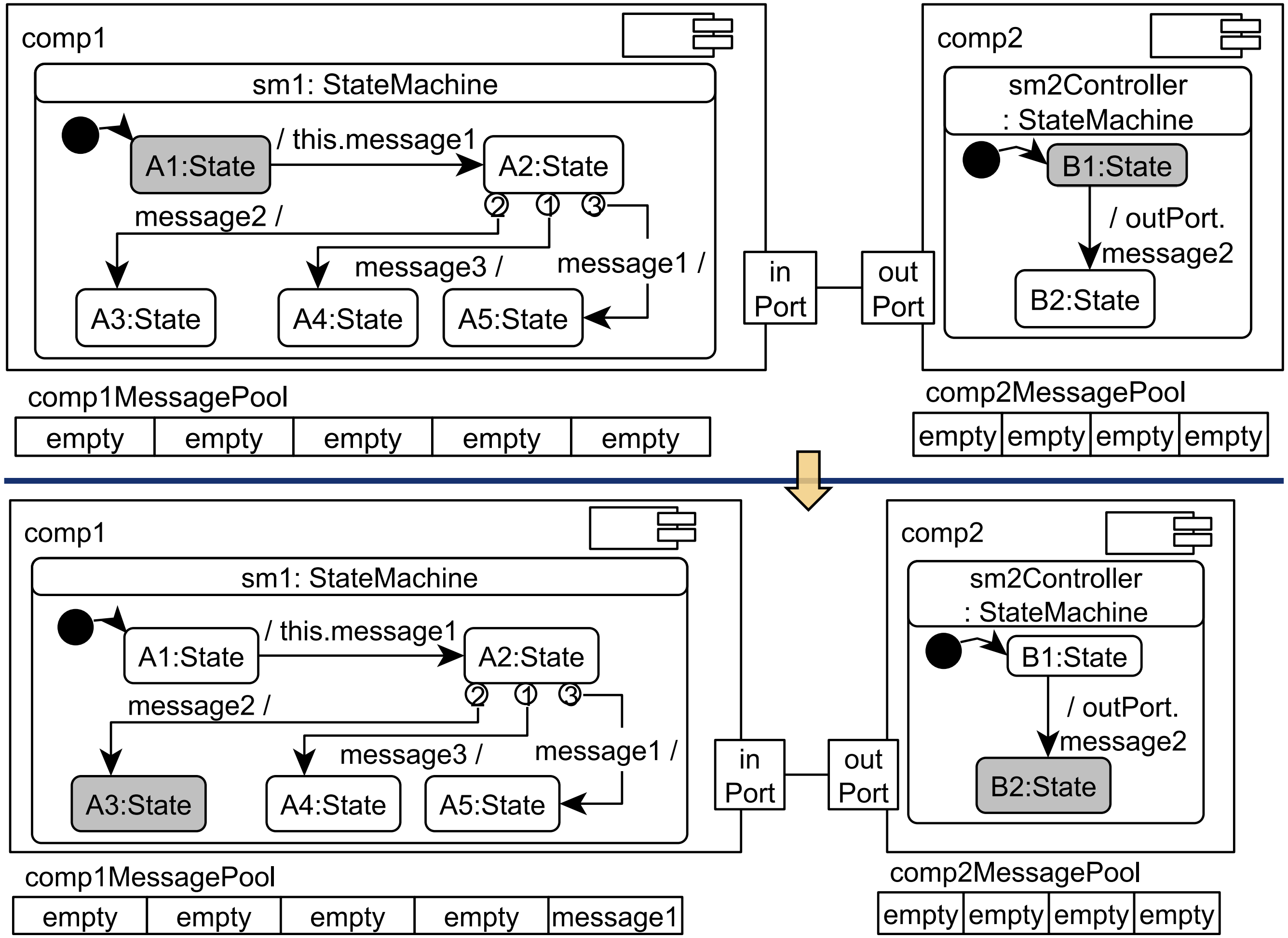
Example Behavior

- communication needs a protocol as a formal description.



Message Syntax and Semantics

- Message Pool
- Unique priorities define the concrete execution order of the state machine



comp1MessagePool

empty	empty	empty	empty	empty

comp2MessagePool

empty	empty	empty	empty

comp1MessagePool

empty	empty	empty	empty	message1

comp2MessagePool

empty	empty	empty	empty

Modelica code generation from ModelicaML state machines extended by asynchronous communication

1 Introduction

2 State Machine with Messages

3 Transformation to Modelica Code

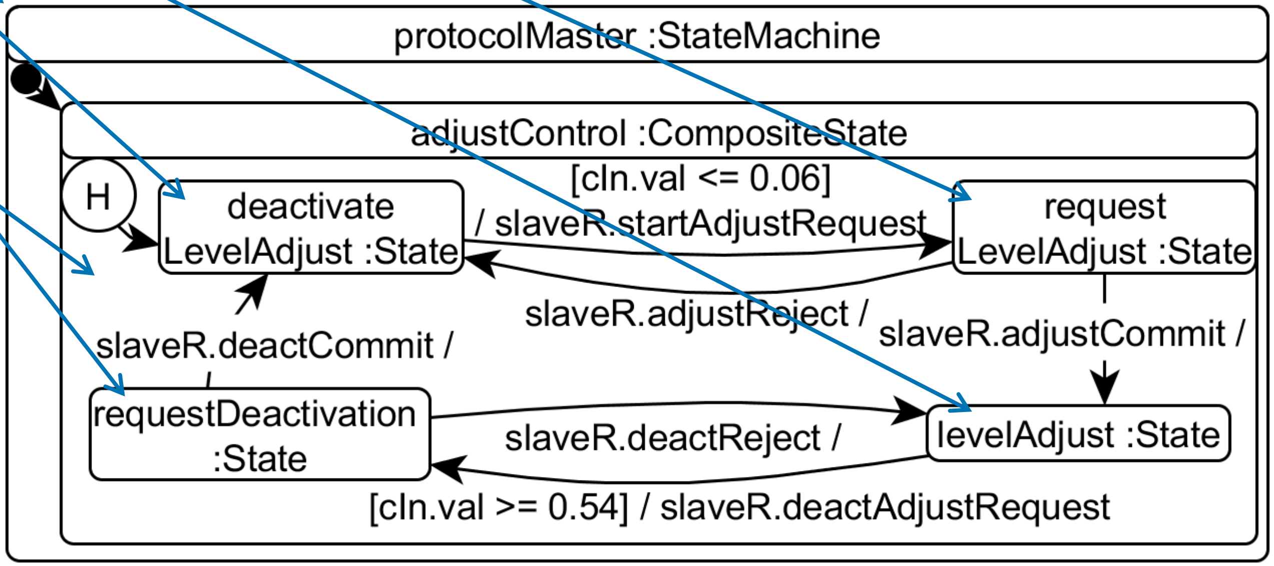
4 Related Work

5 Conclusion

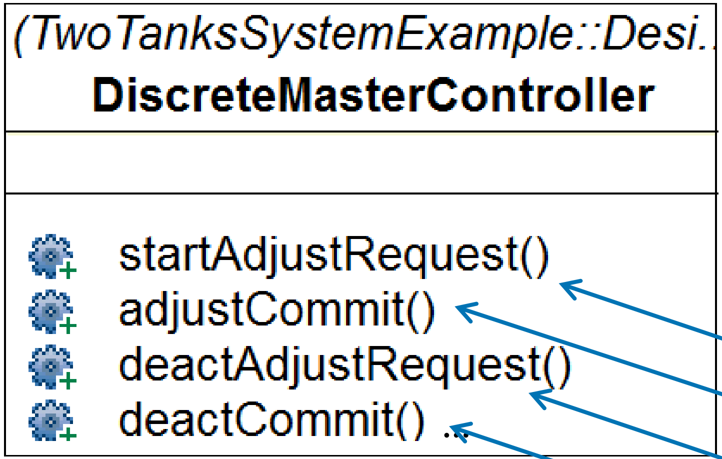
Generated Modelica Code for the Structure of State Machines

```
record masterController_SM_protocolMasterControl
Boolean active;
…
protocolMaster_Region_0_adjustControl_Region0 Region_0;
record protocolMaster_Region_0_adjustControl_Region0
SimpleState requestLevelAdjust;
SimpleState levelAdjust;
SimpleState deactivateLevelAdjust;
SimpleState requestDeactivation;
…
end protocolMaster_Region_0
_adjustControl_Region0;

record SimpleState
Boolean active;
…
end SimpleState;
```



Generated Modelica Code for the Definition of Messages



(TwoTanksSystemExample::Desi...

DiscreteMasterController

- startAdjustRequest()
- adjustCommit()
- deactAdjustRequest()
- deactCommit() ...

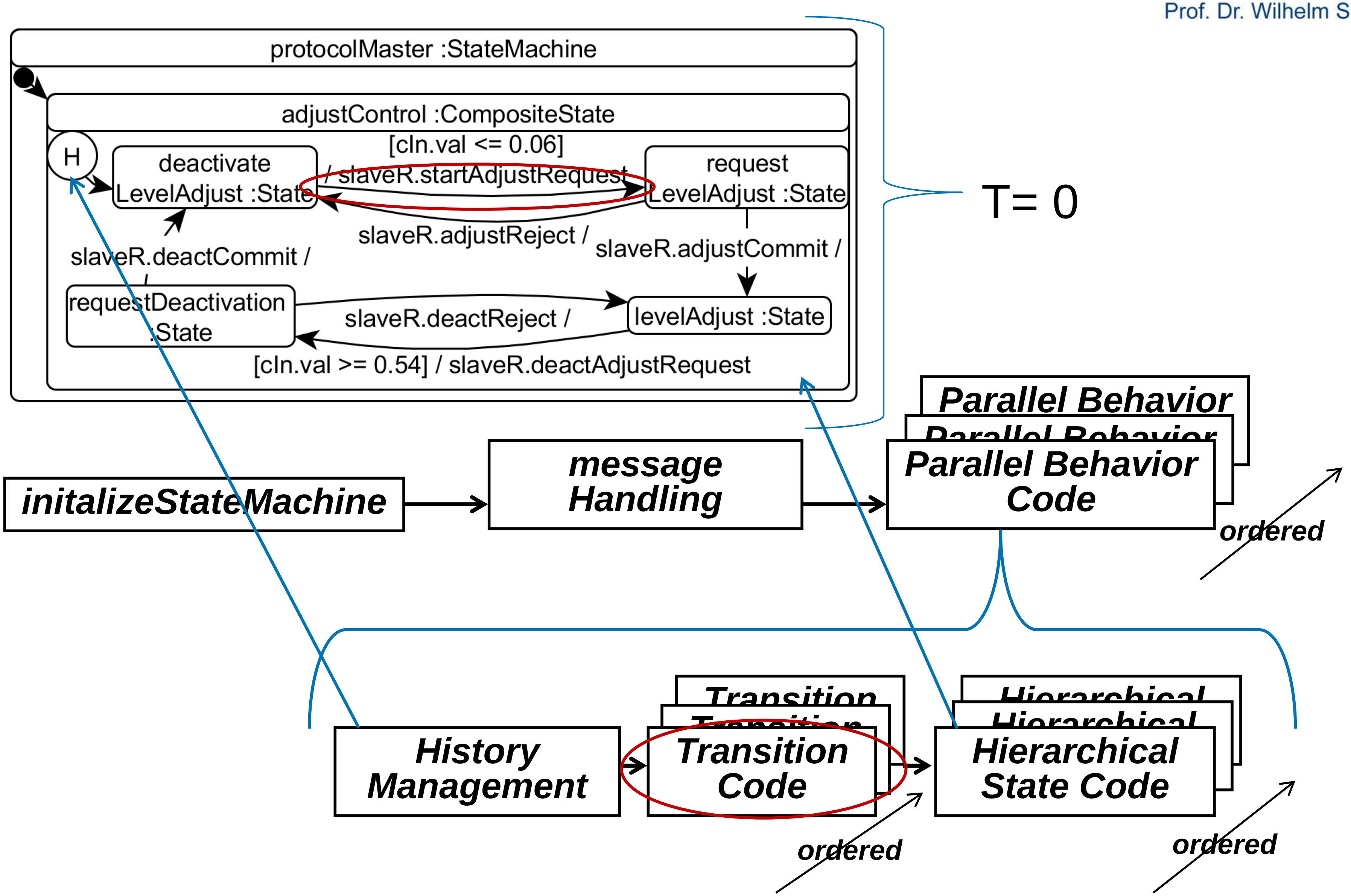
```
record stdMessage
Integer port;
Integer msgType;
end stdMessage;


Integer cmpMsgPoolAdr;


Boolean startAdjustRequest;
Boolean adjustCommit;
Boolean deactAdjustRequest;
Boolean deactCommit;
…
```

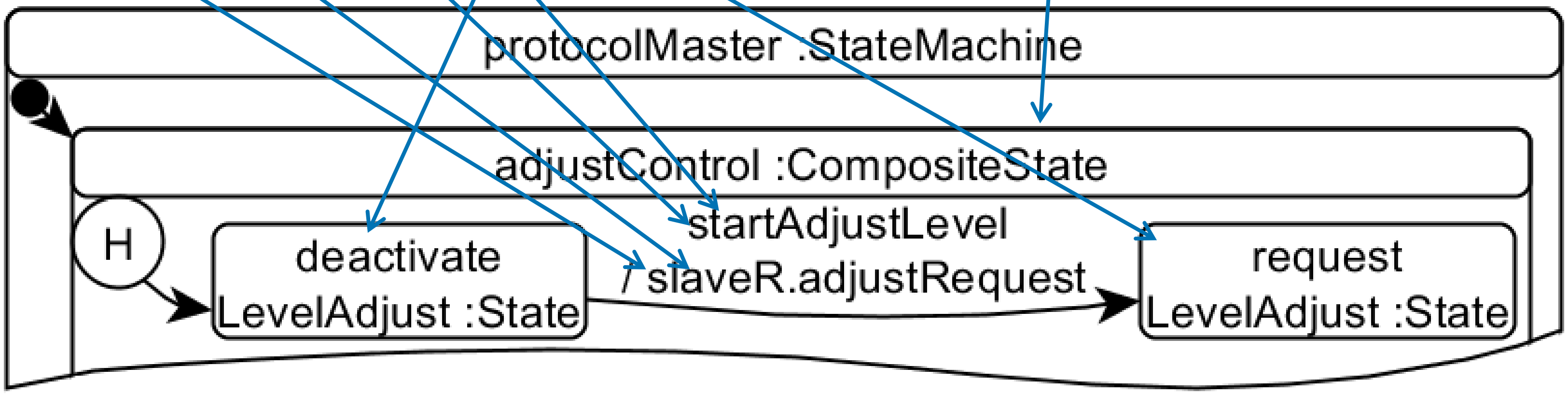
Memory address of the message pool

Structure of the Modelica Algorithmic Behavior Code

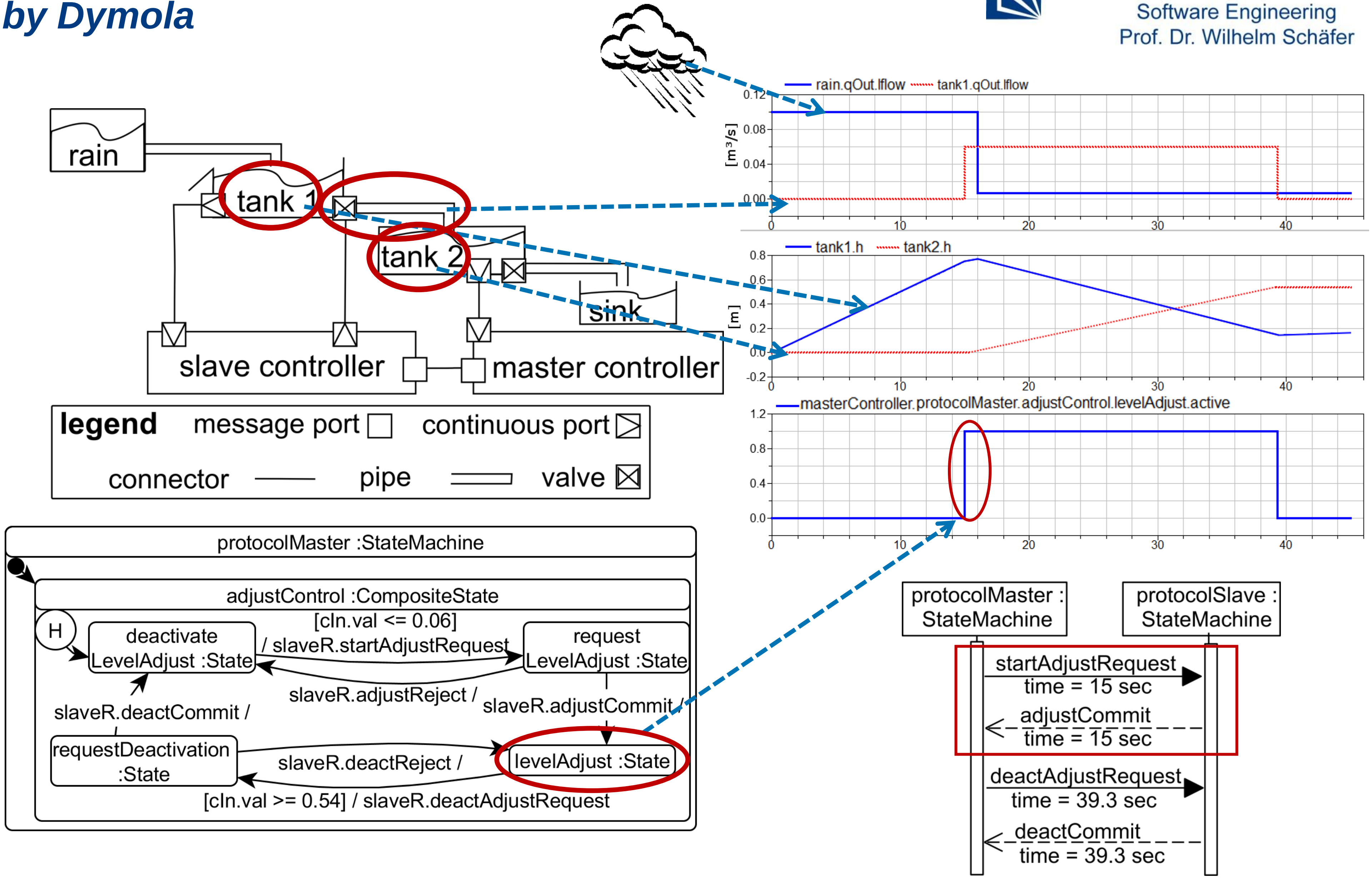


Generated Code for a Transition

```
algorithm
...
if pre(protocolMaster.Region_0.adjustControl.active) then
  if pre(... .adjustControl.deactivateLevelAdjust.active) then
    if startAdjustLevel then
      ....deactivateLevelAdjust.active := false ;
      startAdjustLevel := false ;
      message.msgType:=10; //adjustRequest
      sendMessage(pre(cmpMsgPoolAdr,message);
      ... .requestLevelAdjust.active := true;
    end if;
  end if;
end if;
```


protocolMaster :StateMachine
adjustControl :CompositeState
H
deactivate LevelAdjust :State
startAdjustLevel / slaveR.adjustRequest
request LevelAdjust :State

Simulation of the Rainwater Two Tanks System by Dymola


rain
tank 1
tank 2
sink
slave controller
master controller
legend
message port □
continuous port
connector
pipe
valve ☒
protocolMaster :StateMachine
adjustControl :CompositeState
H
deactivate LevelAdjust :State
[cIn.val <= 0.06] / slaveR.startAdjustRequest
request LevelAdjust :State
slaveR.adjustReject /
slaveR.deactCommit /
slaveR.adjustCommit /
requestDeactivation :State
slaveR.deactReject /
levelAdjust :State
[cIn.val >= 0.54] / slaveR.deactAdjustRequest
rain.qOut.lflow
tank1.qOut.lflow
[m³/s]
0.12
0.08
0.04
0.00
0
10
20
30
40
tank1.h
tank2.h
[m]
0.8
0.6
0.4
0.2
0.0
-0.2
masterController.protocolMaster.adjustControl.levelAdjust.active
1.2
0.8
0.4
0.0
protocolMaster : StateMachine
protocolSlave : StateMachine
startAdjustRequest time = 15 sec
adjustCommit time = 15 sec
deactAdjustRequest time = 39.3 sec
deactCommit time = 39.3 sec

Modelica code generation from ModelicaML state machines extended by asynchronous communication

1 Introduction

2 State Machine with Messages

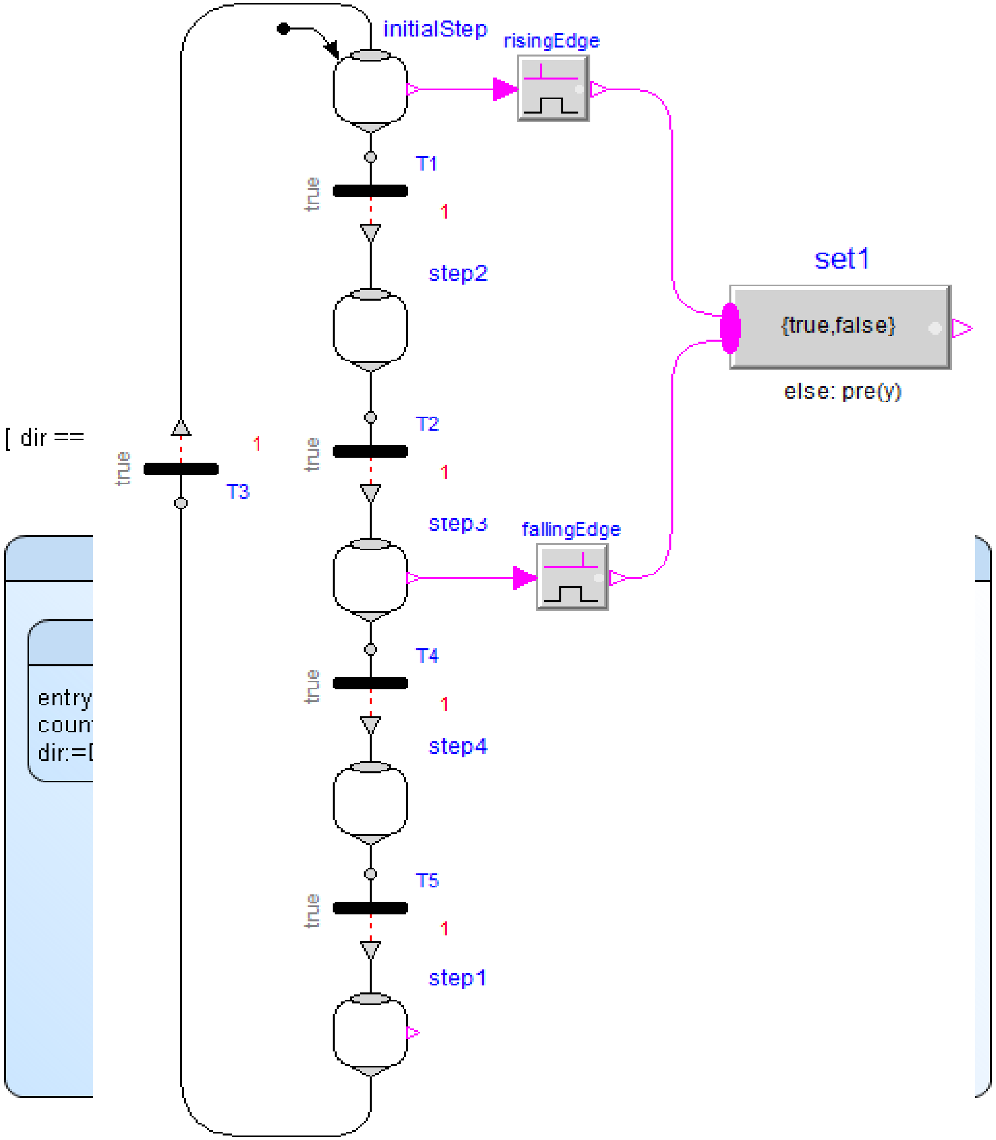
3 Transformation to Modelica Code

4 Related Work

5 Conclusion

Related Work

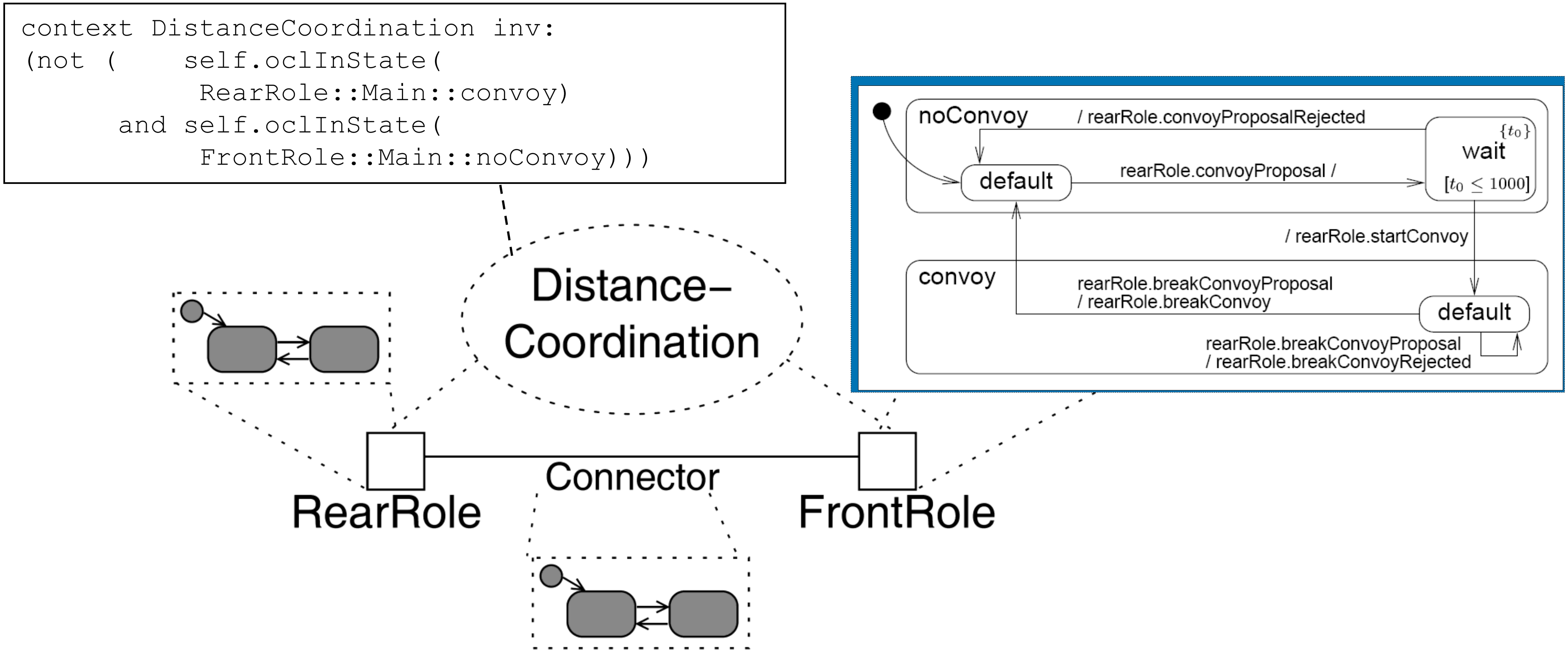
- State Graph2
  - Equation based
  - Implement as Modelica library
  - Not UML conform
  - No algorithm support
  - Higher visual complexity
  - No message exchange support
- SimulationX
  - Modelica code generation
  - No parallel regions, submachines
  - UML like state machine
  - No message exchange support



Related Work

- MechatronicUML
  - Model-driven software development and verification of mechatronic real-time systems
- The presented message exchange is based on the MechatronicUML

- Further constructs
  - Capabilities of timed automata
  - Real-Time Coordination Pattern
  - Hybrid Reconfiguration Charts



```
context DistanceCoordination inv:
(not (    self.oclInState(
           RearRole::Main::convoy)
      and self.oclInState(
           FrontRole::Main::noConvoy)))
```

Modelica code generation from ModelicaML state machines extended by asynchronous communication

1 Introduction

2 State Machine with Messages

3 Transformation to Modelica Code

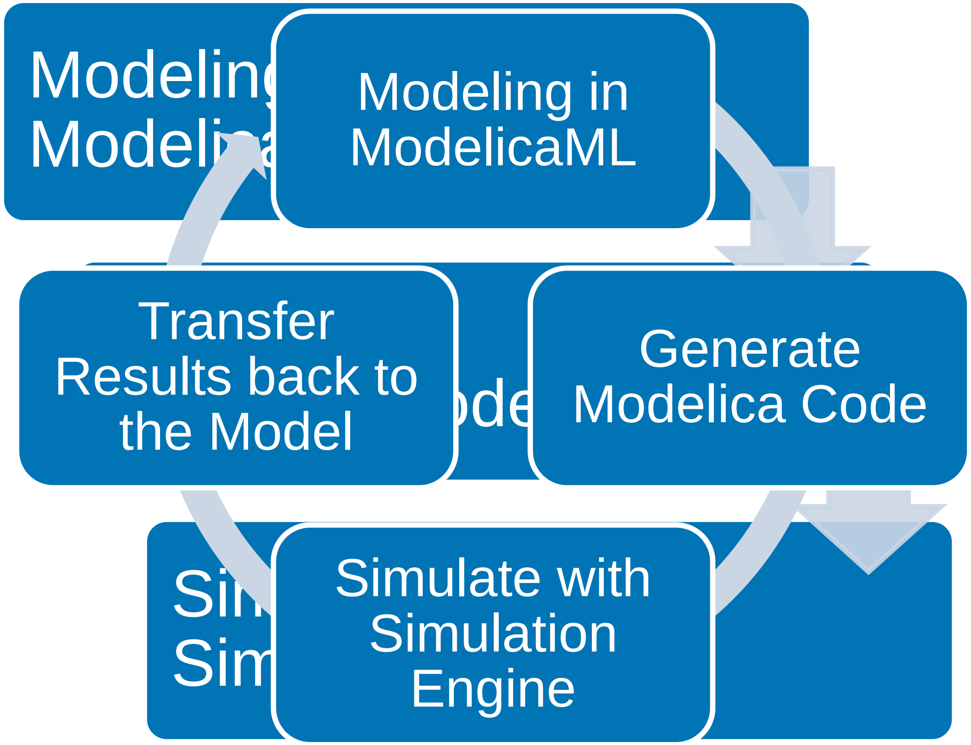
4 Related Work

5 Conclusion

Summary and Future Work

Summary

- Extended ModelicaML state machines with message passing
- Define syntax and semantics of state machines and messages
- Translation of ModelicaML state machines to Modelica


Modeling in ModelicaML
Generate Modelica Code
Simulate with Simulation Engine
Transfer Results back to the Model

Future Work

- Transformation of state machines to StateGraph2
- Add elements to specify temporal real-time behavior, such as clocks, time guards, invariants from timed automata
- Transfer simulation results back into the model
- Visualize simulation results of the state machine behavior
- Visualize simulation results of message passing as sequence diagrams

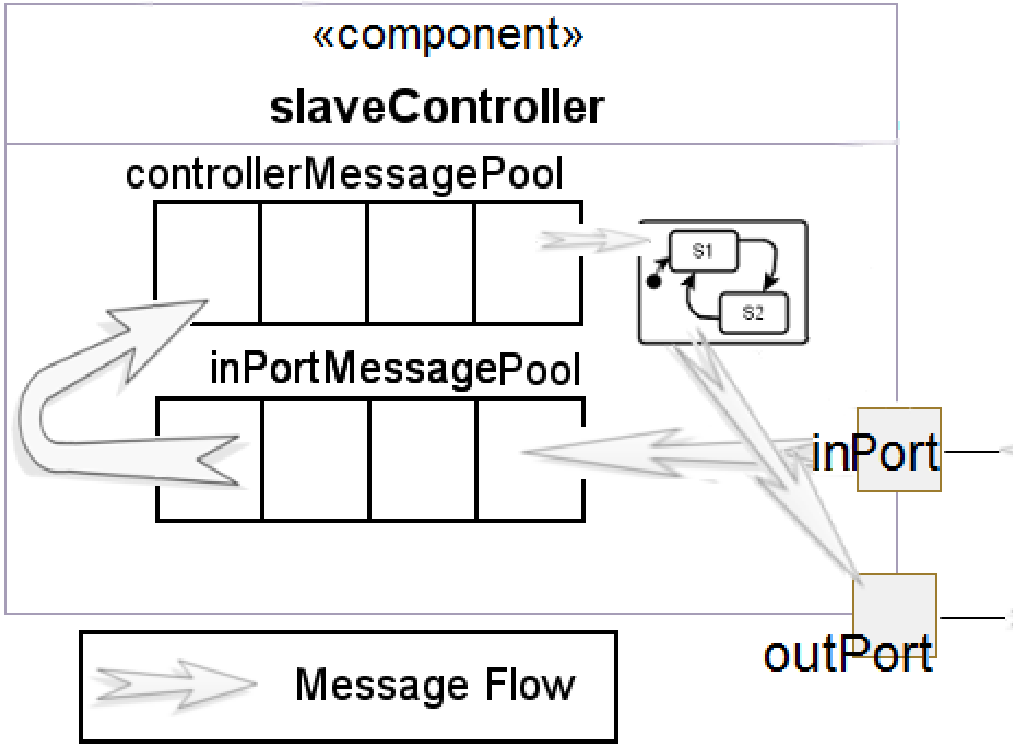
Thank you for your attention

Heinz Nixdorf Institute
University of Paderborn
Software Engineering
Warburger Str. 100
33098 Paderborn

Phone: +49 5251 603323
E-mail: upohl@uni-paderborn.de
http://www.cs.uni-paderborn.de/en/fachgebiete/software-engineering/

Generated Code for the Message Handling of the Message Queues

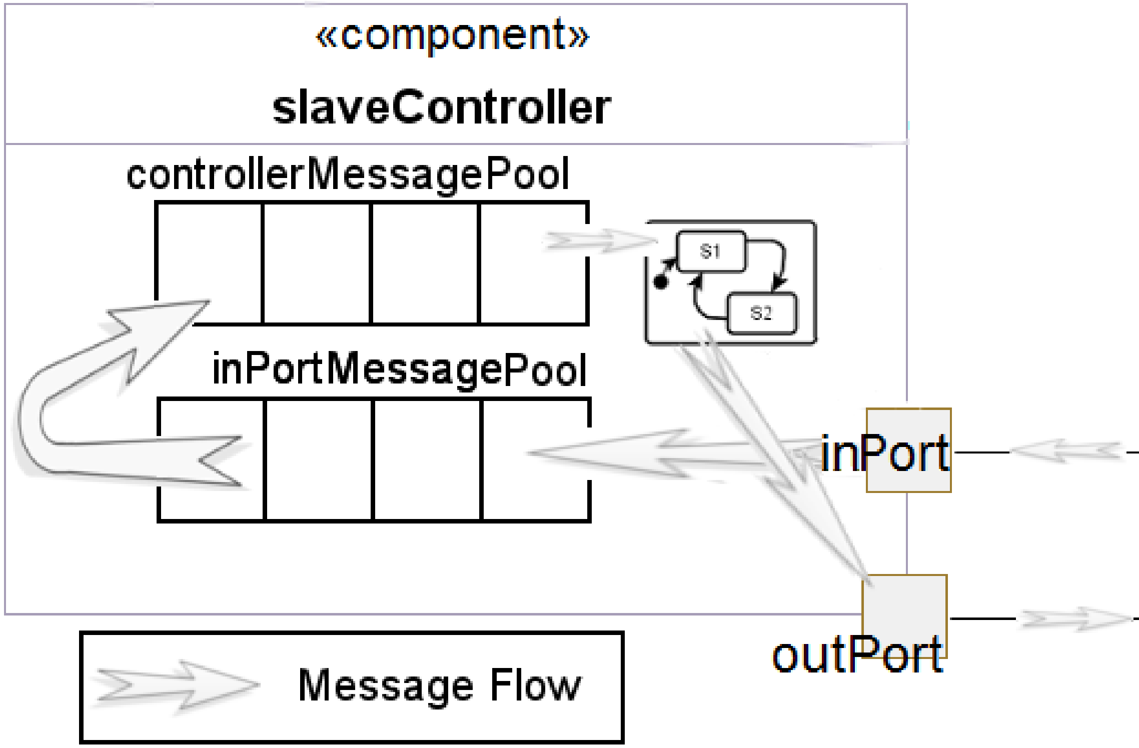
```
for i in 1:numIn loop
  numreceived :=
   numMessages(inputMsgPoolAdr[i]);
  for j in 1:numreceived loop
    message :=
   getMessage(inputMsgPoolAdr[i]);
    message.port := i;
    sendMessage(cmpMsgPoolAdr,event);
  end for;
end for;
numreceived :=
   numMessages(cmpMsgPoolAdr);
for j in 1:numreceived loop
  message :=
   getMessage(cmpMsgPoolAdr);
if message.msgType == 10
and adjustRequest == false) then
  adjustRequest := true;
...
else
sendMessage(cmpMsgPoolAdr,message);
end if;
```



Generated Modelica Helper Functions for Message Handling

```
function CreatePool
output Integer q;
external "C" q = QCreate();
annotation (Include="#include
   <events.c>");
end CreatePool;
```

```
function sendMessage
input Integer poolAdr;
input stdMessage e;
output Integer out;
external "C" out =
QAdd(poolAdr,e.port,e.msgType,
   e.value,0);
annotation (Include="#include
   <events.c>");
end sendMessage;
```


«component»
slaveController
controllerMessagePool
S1
S2
inPortMessagePool
inPort
outPort
Message Flow