Tutorial:
Embedded and Reactive Systems Development with UML-RT and Papyrus-RT

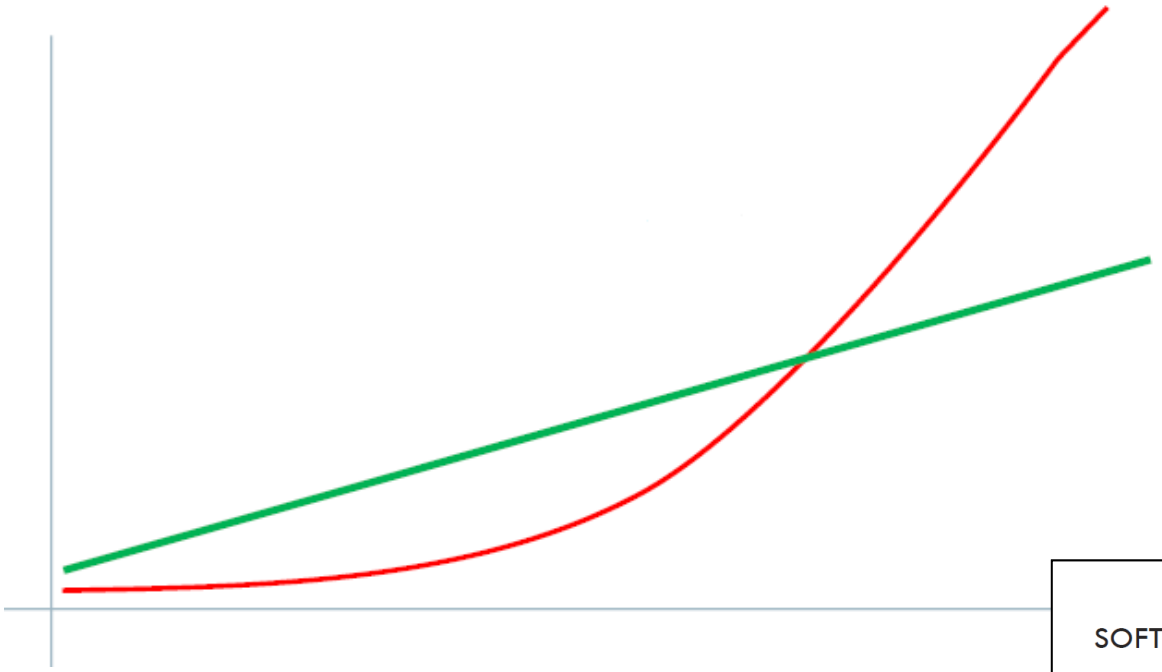
Juergen Dingel (Queen's)
Nicolas Hili (Queen's)
Ernesto Posse (Zeligsoft)

MODELS'17

Sept 18, 2017

All material available at
http://flux.cs.queensu.ca/mase/papyrus-rt-resources/supporting-material-for-the-models17-tutorial/

49 Years Ago at 1st NATO SW Eng Conference



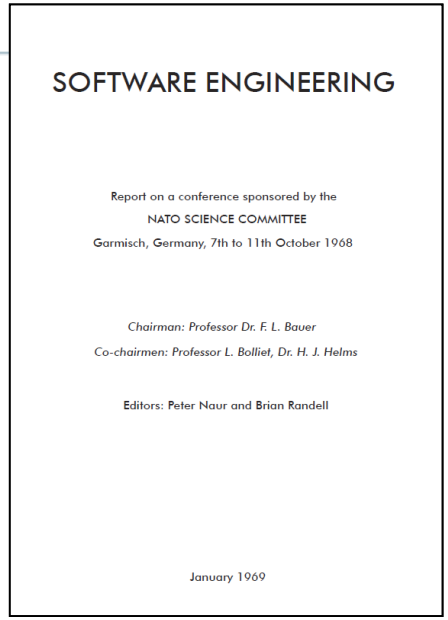
HW computing power ⇑

⇒ Complexity of tasks SW asked to do ⇑

⇒ Complexity of SW ⇑

⇒ Existing SW development capabilities strained

⇒ **"Software crisis"**



SOFTWARE ENGINEERING

Report on a conference sponsored by the
NATO SCIENCE COMMITTEE
Garmisch, Germany, 7th to 11th October 1968

Chairman: Professor Dr. F. L. Bauer
Co-chairmen: Professor L. Bolliet, Dr. H. J. Helms

Editors: Peter Naur and Brian Randell

January 1969

Since Then: LOTS of Progress

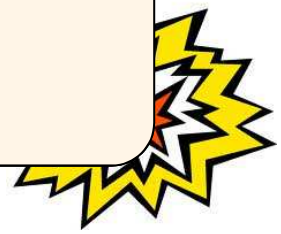
- **Hardware**
  - **Computing power** (2016 vs 1969) [Paul Ledak on quora.com]:
    - Number of transistors:
      - iPhone 6 = Apollo 11 GC x 180,000
    - Clock frequency:
      - iPhone 6 = Apollo 11 GC x 32,000
    - Instructions per second:
      - iPhone 6 = Apollo 11 GC x 80 million
    - Overall:
      - iPhone 6 = Apollo 11 GC x 120 million
  - **Cost of 1 MB of memory** in US$ [www.jcmit.com]:
    - Dec 2015 = 1957 / 100 billion

Since Then: LOTS of Progress

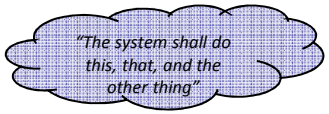
- **Software engineering**
  - Information hiding via modularization, encapsulation, interfaces, MDE, …
- **Pro**
  - (
- **Dat**
  - Relational model, …
- **Operating systems**
  - Virtual memory, …

Key general techniques:

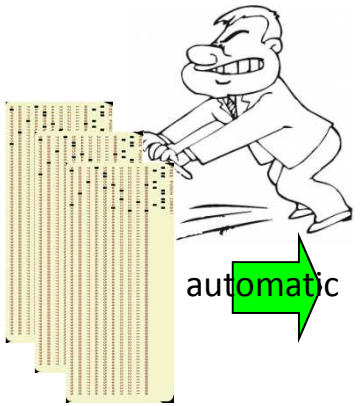
Abstraction, automation, and analysis



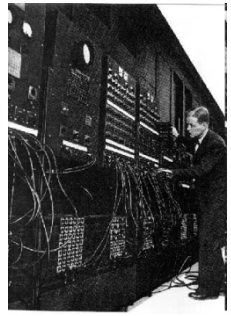
40 years ago



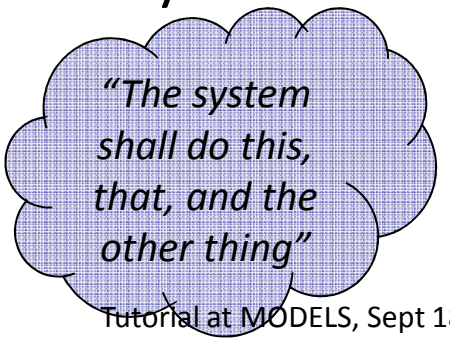
manual



automatic



Today



manual

automatic

Model-Driven Engineering (MDE)

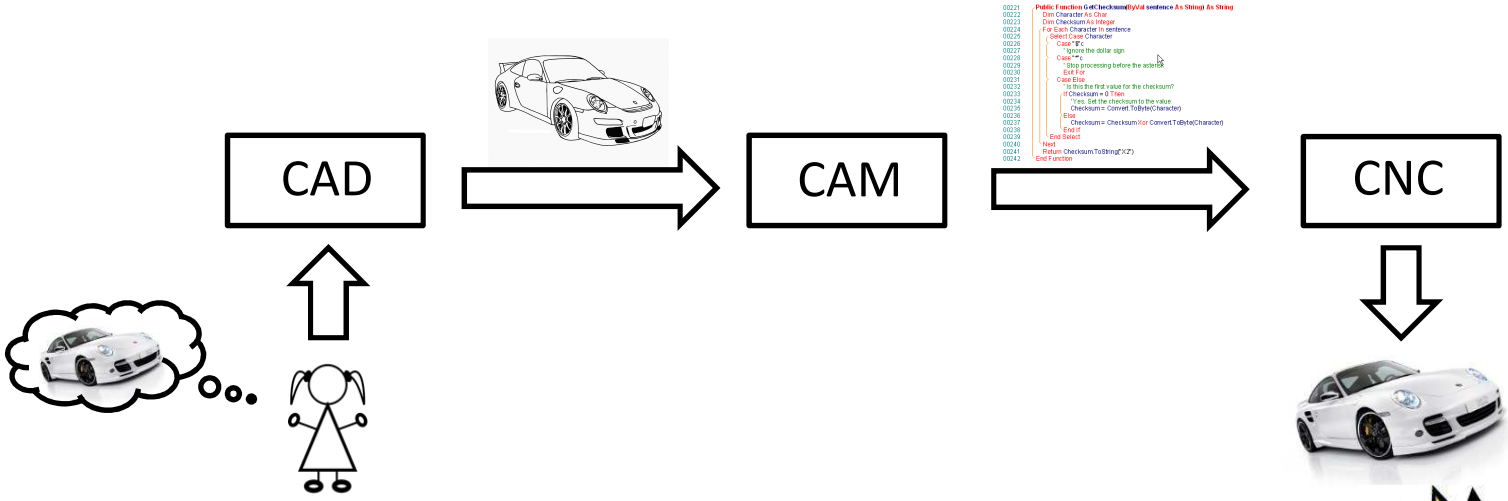
- Improve productivity, quality, and ability to handle complexity by
  - increasing level of **abstraction**
    - through use of 'models'
  - leveraging **automation**
    - e.g., via code generation from models, model transformation, …
  - improving **analysis** capabilities
    - e.g., through constraint solving, simulation, state space exploration, …

MDE = Abstraction + Automation + Analysis

- Inspired by use of models in engineering and science

Abstraction, Automation, and Analysis in Manufacturing

- Mechanical design till early 1970ties: paper drawings, …
- Mechanical design from about 1972: CAD/CAM
  1. Create drawings w/ computer (CAD)
  2. Computer automatically generates milling/CNC programs (CAM)



⇒ much better analysis capabilities and productivity

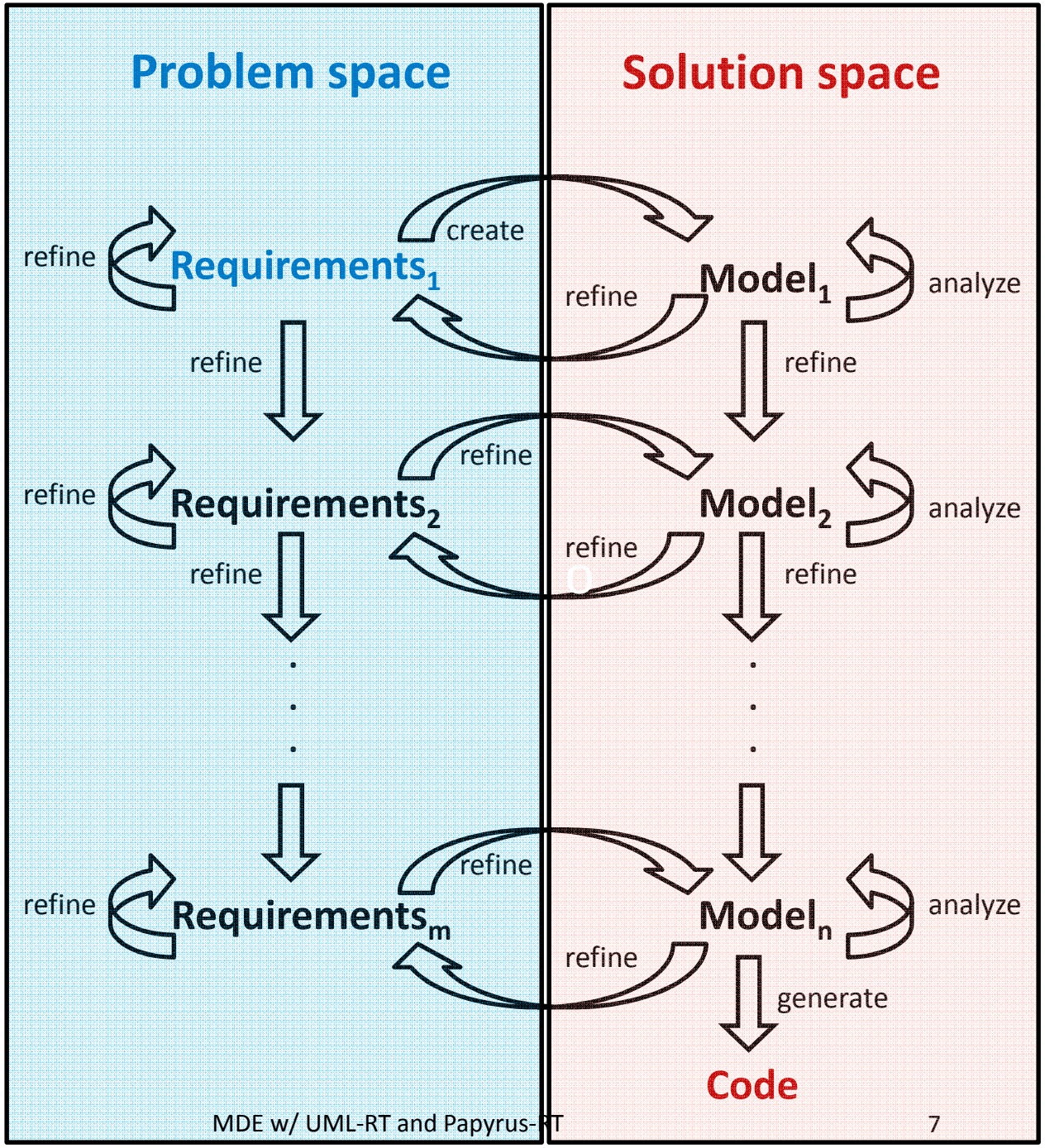
⇒ abstraction, automation, and analysis have revolutionized manufacturing

MDE Process

Elements in solution space exist in **same medium:** the computer

⇒ Model can evolve into system it is modeling!

⇒ fewer discontinuities



MDE for Embedded Systems: Context

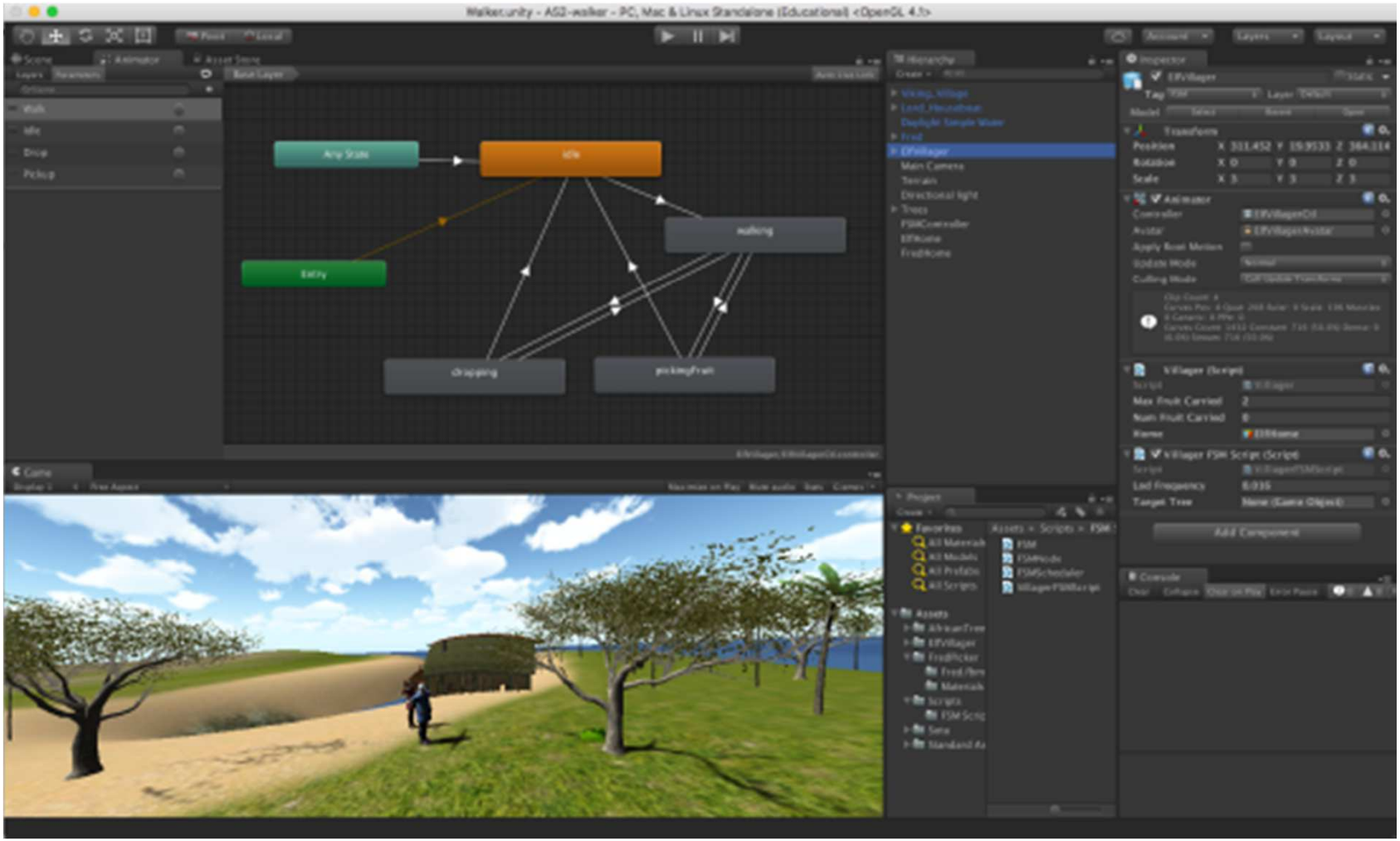
- Some vendors
  - Mathworks: Stateflow/Simulink
  - IBM: Rational RoseRT, Rational Rhapsody, RSA-RTE
  - National Instruments: LabVIEW
  - Esterel Technologies: SCADE
  - IAR Systems: IAR Visual State
- Some standards
  - DO-178C, DO-331

[Radio Technical Commission for Aeronautics (RTCA). DO-178C: Software Considerations in Airborne Systems and Equipment Certification. Jan 2012] [https://en.wikipedia.org/wiki/DO-178C]

[Radio Technical Commission for Aeronautics (RTCA). DO-331 "Model-Based Development and Verification Supplement to DO-178C and DO-278A]

Even the Game Industry is Using MDE Now

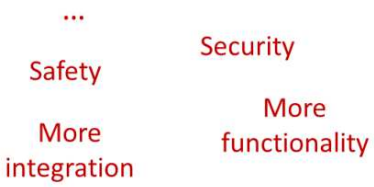
[http://docs.unity3d.com/Manual/Animator.html]



Screenshot courtesy Nick Graham

MDE: Challenges, Opportunities

- Challenges [1],[2]
  - Technical: user experience, model analysis, …
  - Social: education/training, …
- Opportunities
  - Emerging eco-system: open source, standards, forums, repositories, …
  - Abstraction, automation, and analysis will continue to be key [3,4]



[1] Selic. What will it take? A view on adoption of model-based methods in practice. Software and Systems Modeling (SoSyM) 11(4):513-526. October 2012.

[2] Whittle, Hutchinson, Rouncefield. The state of practice in model-driven engineering. IEEE Software 31 (3), 79-85. 2014.

[3] SPARC. Robotics 2020 Multi-Annual Roadmap: For Robotics in Europe, Horizon 2020 Call ICT-2017 (ICT-25, ICT-27 & ICT-28). Dec 2016.

[4] Dingel. Complexity is the Only Constant: Trends in Computing and their Relevance to Model Driven Engineering. Proceedings ICGT'16. LNCS 9761:79-85. 2016.

Goal of Tutorial

- Inform
  - Intro to MD with UML-RT and Papyrus-RT
  - Pointers to resources, related work, etc
- Inspire
  - We need more abstraction, automation, analysis!
  - UML-RT
    - small, cohesive set of concepts

      "UML-RT has features which appeal to the formalist, but some are severely underused by practioners. The primary reason is undoubtedly that there is nothing within the Rose RealTime toolset that can take advantage of the extra information, relegating it instead to a documentary role" [5]
    - successful track record, but work needed on, e.g.,
      - static analysis, user experience, deployment, interpretation, testing, verification, simulation, …

[5] Whittaker, Goldsmith, Macolini, Teitelbaum, "Model Checking UML-RT Protocols", *Proc. Workshop Formal Design Techniques for Real-Time UML*, 2000-Nov.

Overview

2 – 3:30pm:

1.	Intro		
2.	MDE	(10 mins)	(10 slides)
3.	Overview	(1 min)	(1 slide)
4.	Papyrus-RT	(10 mins)	(3 slides)
	• Hands on (installation)		
5.	UML-RT: Part I	(60 mins)	(24 slides)
	• Core concepts		
	• Demo and hands on		

3:30 – 4pm: Coffee break

4 – 5:30pm:

6.	UML-RT: Part II	(60 mins)	(14 slides)
	• More advanced concepts		
	• Demo and hands on		
7.	Ongoing and future work	(10 mins)	(5 slides)
8.	Conclusion	(10 mins)	(3 slides)

All material available at http://flux.cs.queensu.ca/mase/papyrus-rt-resources/supporting-material-for-the-models17-tutorial/

Papyrus-RT: Overview

- Papyrus for Real-Time industrial-grade, complete modeling environment for the development of complex, software intensive, real-time, embedded, cyber-physical systems.
- Part of PolarSys
  - Eclipse Working Group
  - Open source for embedded systems

- Building on
  - Eclipse Modeling Framework (EMF), Xtext, Papyrus

- History
  - 2015: V0.7.0
  - March 2017: v0.9
  - Fall 2017: v1.0

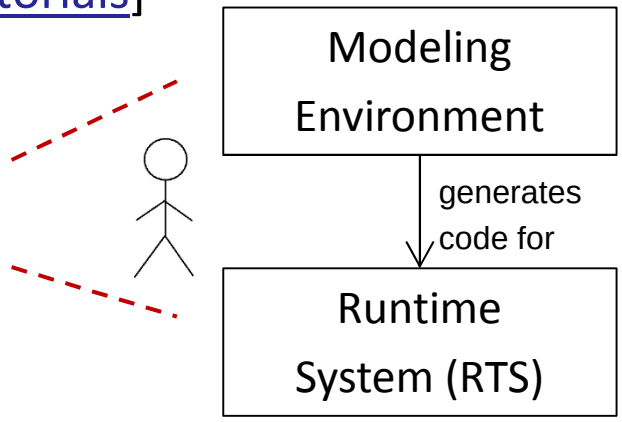
[https://wiki.eclipse.org/Papyrus-RT]

Papyrus-RT: Installation

- Easiest: as RCP
- From web:
  - [https://eclipse.org/papyrus-rt/content/download.php]
  - Download RCP for your platform
  - Extract downloaded file into a folder of your choice
- From USB stick:
  - In ‘Papyrus-RT’ folder:
    - Archive: Copy/paste, unpack
  - In ‘Models’ folder:
    - Models: Import in Papyrus-RT
  - In ‘Doc’ folder:
    - Installation instructions

Papyrus-RT: Use

- Tutorials
  - [https://wiki.eclipse.org/Papyrus-RT/User#Tutorials]
- 2 parts
  1. Editing, building the model, generate code
  2. Compiling and running generated code
     - Linux: easy
       - [https://wiki.eclipse.org/Papyrus-RT/User/User_Guide/Getting_Started#Execute_the_model]
     - macOS: use VirtualBox/Vagrant
     - Windows: use Cygwin, or VirtualBox/Vagrant
       - [https://wiki.eclipse.org/Papyrus-RT/User_Guide/Vagrant_Setup]



Overview

2 – 3:30pm:

1.	Intro		
2.	MDE	(10 mins)	(10 slides)
3.	Overview	(1 min)	(1 slide)
4.	Papyrus-RT	(10 mins)	(3 slides)
	• Hands on (installation)		
5.	UML-RT: Part I	(60 mins)	(24 slides)
	• Core concepts		
	• Demo and hands on		

3:30 – 4pm: Coffee break

4 – 5:30pm:

6.	UML-RT: Part II	(60 mins)	(14 slides)
	• More advanced concepts		
	• Demo and hands on		
7.	Ongoing and future work	(10 mins)	(5 slides)
8.	Conclusion	(10 mins)	(3 slides)

All material available at http://flux.cs.queensu.ca/mase/papyrus-rt-resources/supporting-material-for-the-models17-tutorial/

Overview

2 – 3:30pm:

1.	Intro		
2.	MDE	(10 mins)	(10 slides)
3.	Overview	(1 min)	(1 slide)
4.	Papyrus-RT	(10 mins)	(3 slides)
	• Hands on (installation)		
5.	UML-RT: Part I	(60 mins)	(24 slides)
	• Core concepts		
	• Demo and hands on		

3:30 – 4pm: Coffee break

4 – 5:30pm:

6.	UML-RT: Part II	(60 mins)	(14 slides)
	• More advanced concepts		
	• Demo and hands on		
7.	Ongoing and future work	(10 mins)	(5 slides)
8.	Conclusion	(10 mins)	(3 slides)

Modeling Languages

Modelica
- Physical systems
- Equation-based

Simulink
- Continuous control, DSP
- time-triggered dataflow

Stateflow
- Reactive systems
- Discrete control
- State-machine-based

AADL
- Embedded, real-time

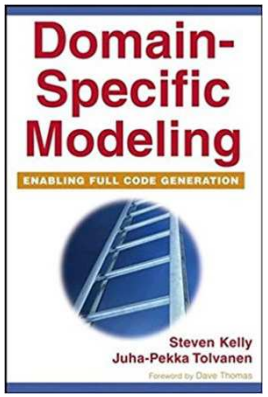
UML

UML MARTE
- Embedded, real-time

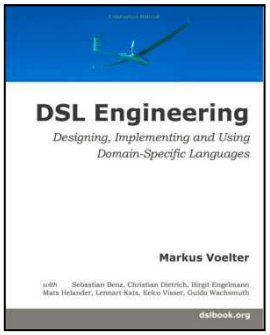
Lustre/SCADE
- Embedded real-time
- Synchronous dataflow

UML-RT
- Embedded, real-time
- State-machine-based

Examples in



[Kelly, Tolvanen 2008]

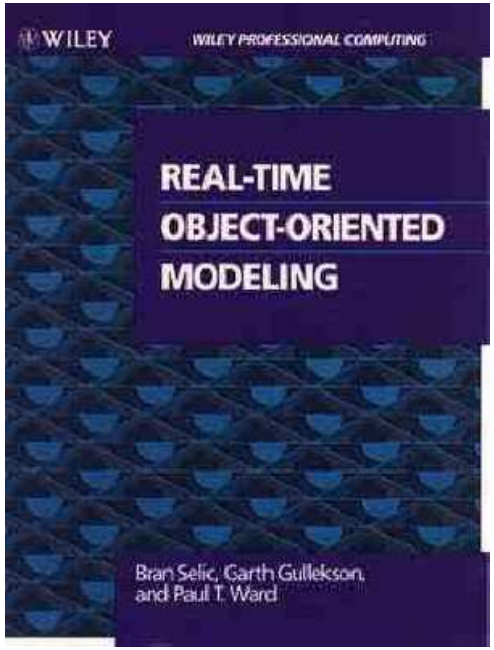


[Voelter 2013]

increasing generality ← → **increasing domain-specifity**

UML-RT: History

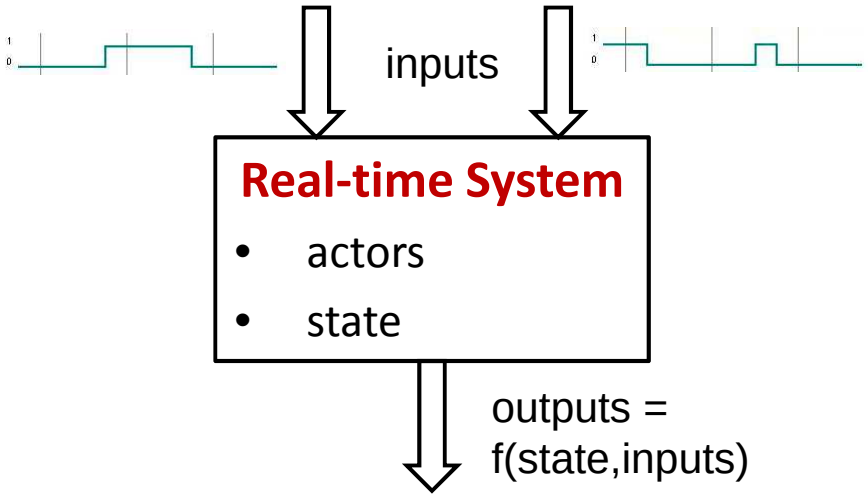
- Real-time OO Modeling (ROOM)
  - ObjecTime, early 1990 ties
- Major influence on UML 2
  - E.g., StructuredClassifier
- "RT subset of UML"
- Tools
  - ObjecTime Developer
  - IBM Rational RoseRT
  - IBM RSA-RTE
  - Eclipse Papyrus-RT

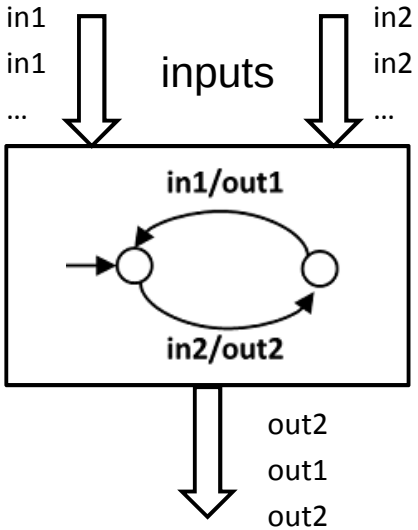


[Selic, Gullekson, Ward. *Real-Time Object-Oriented Modelling*. Wiley. 1994]

UML-RT: Characteristics

- Domain-specific
  - Embedded systems with soft real-time constraints
- Graphical, but textual syntax exists
- Small, cohesive set of concepts
- Strong encapsulation
  - Actors (active objects)
  - Explicit interfaces
  - Message-based communication
- Event-driven execution
  - State machines





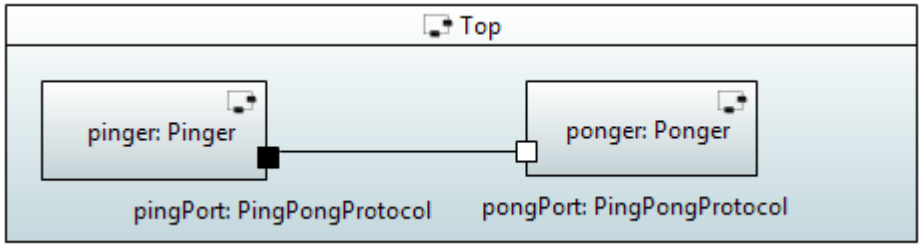
UML-RT Part I

- Core concepts
  - Structural modeling
  - Behavioural modeling

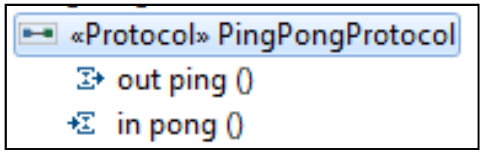
UML-RT: Core Concepts (1)

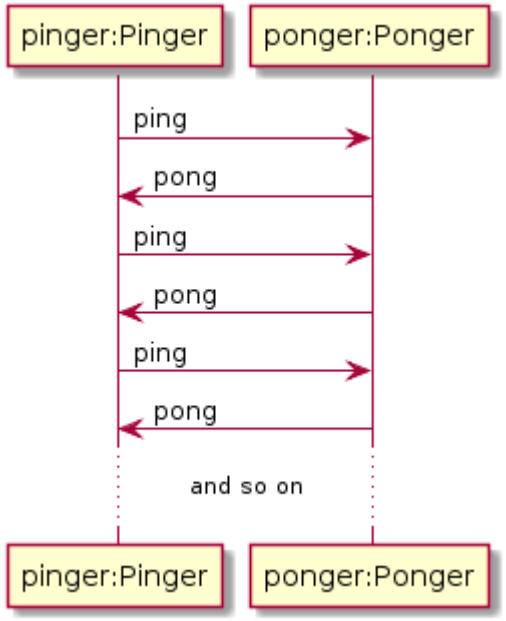
- Types
  - Capsules (active classes)
    - Capsule instances (parts)
  - Passive classes (data classes)
    - Objects
  - Protocols
  - Enumerations
- Structure
  - Attributes
  - Ports
  - Connectors
- Behaviour
  - Messages (events)
  - State machines
- Grouping
  - Package
- Relationship
  - Generalization
  - Associations

UML-RT:
Core Concepts (2)



- Model
  - Collection of capsule definitions
  - 'Top' capsule containing collection of capsule instances (parts)
- Capsules
  - May contain
    - Attributes, ports, or other capsule instances (parts)
  - Behaviour defined by state machine
- Ports
  - Typed over protocol defining input and output messages
- State machine
  - Transition triggered by incoming messages
  - Action code can contain send statements that send messages over certain ports

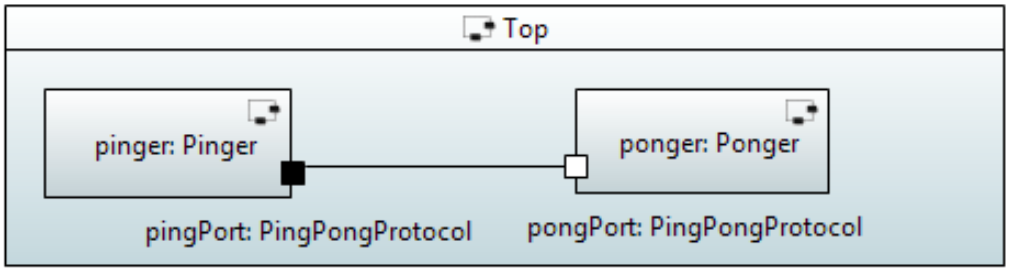


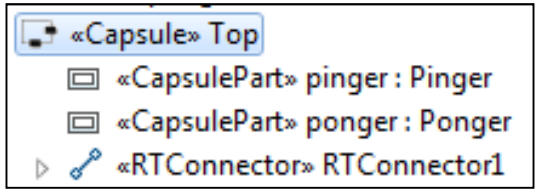


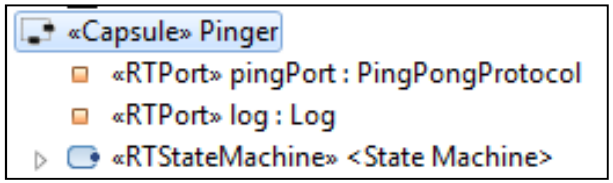
Capsules (1)

- Kind of active class
  - Attributes, operations
  - Own, independent flow of control (logical thread)
- May also contain
  - Ports over which messages can be sent and received
  - Parts (instances of other capsules) and connectors
- Creation, use of instances tightly controlled
  - Created by runtime system (RTS)
  - Cannot be passed around
  - Stored in attribute of another capsule (part)
  - Information flow only via messages sent to ports

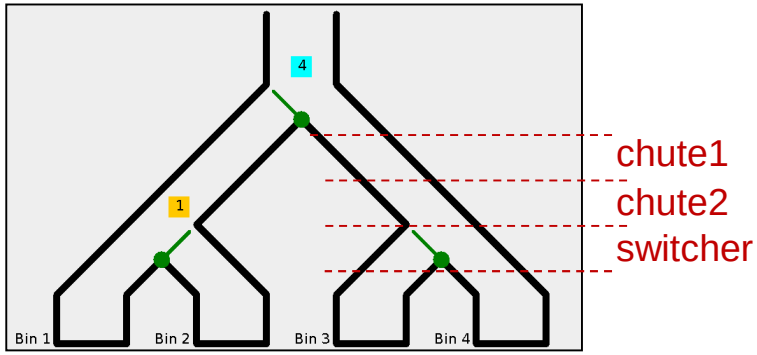
  ⇒ better concurrency control and encapsulation
- Behaviour defined by state machine

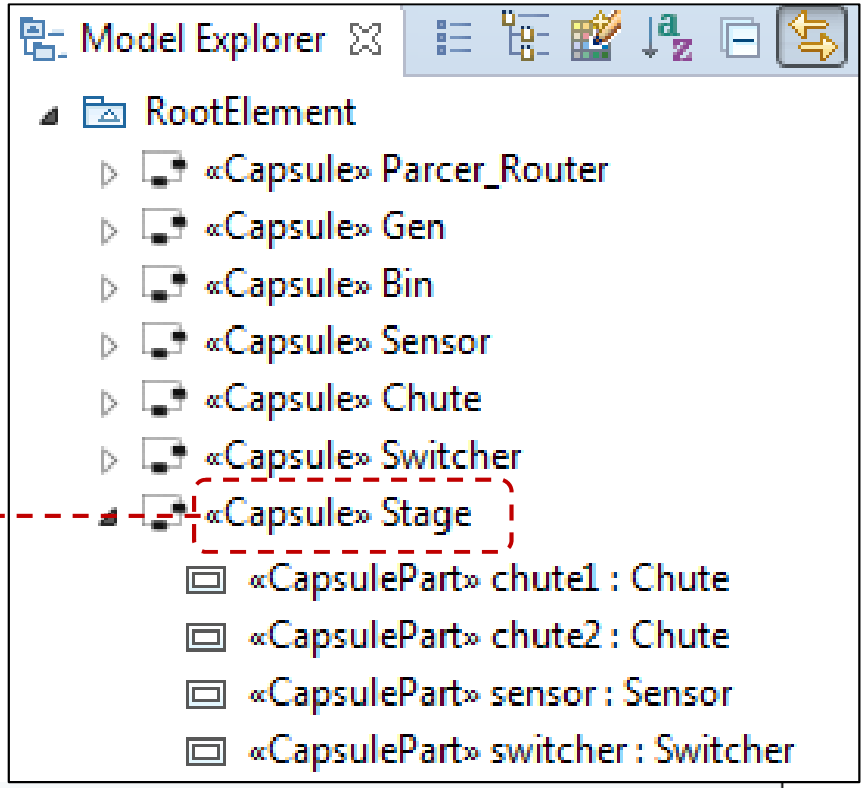


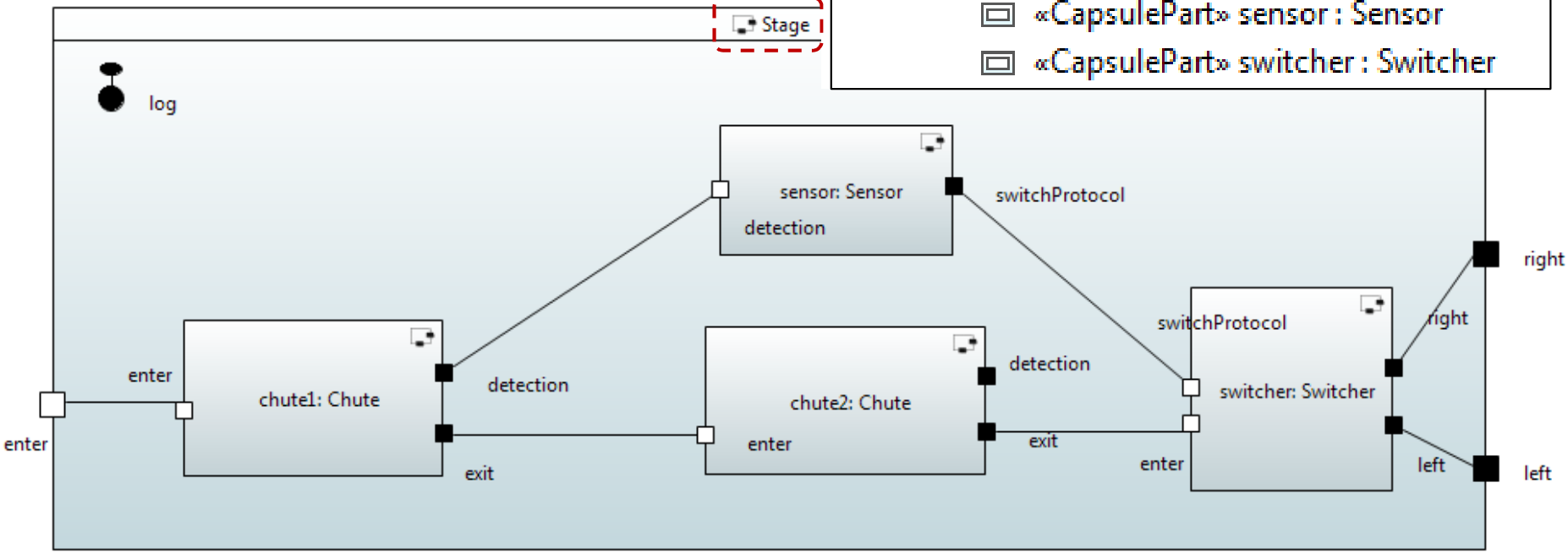




Example: Capsules and Capsule Parts

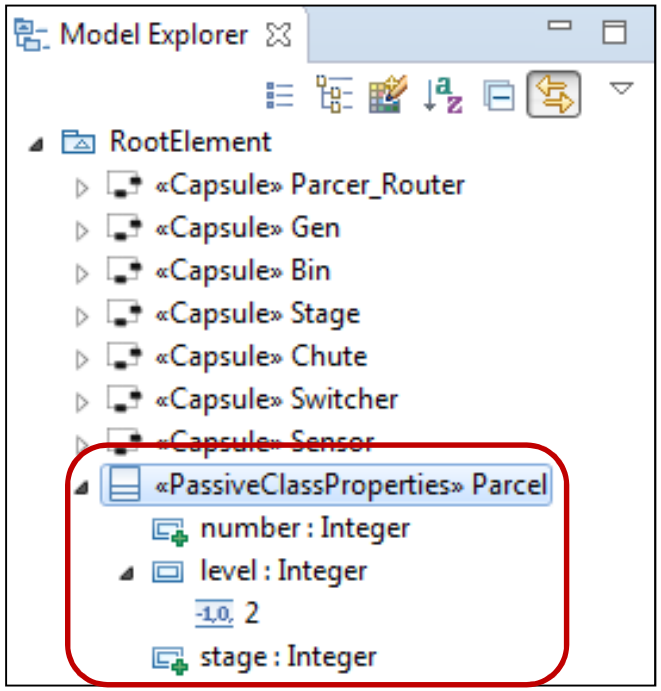






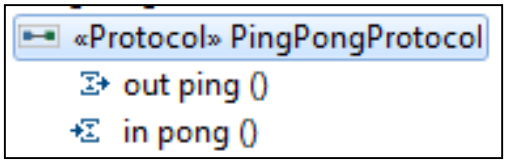
Passive Classes/Data Classes

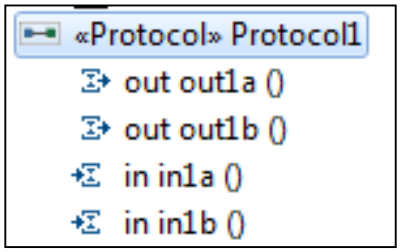
- Similar to regular classes
- Do not have independent flow of control
- Behaviour defined through operations
- Used to define data structures and operations on them

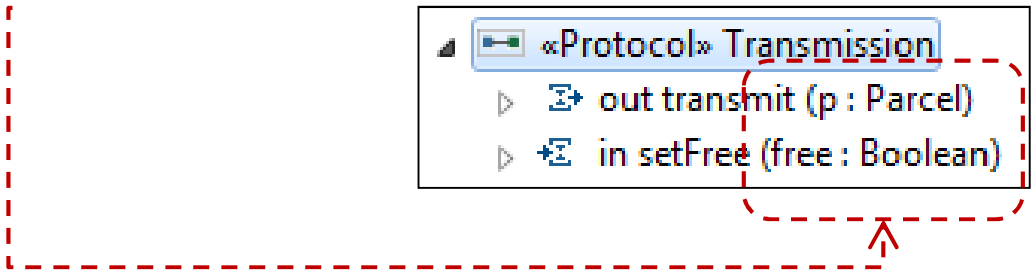


Protocols

- Provide types for ports
- Define
  - Input messages
    - Services provided by capsule owning port
  - Output messages
    - Services required by capsule owning port
  - Input/output messages
- Messages can carry data

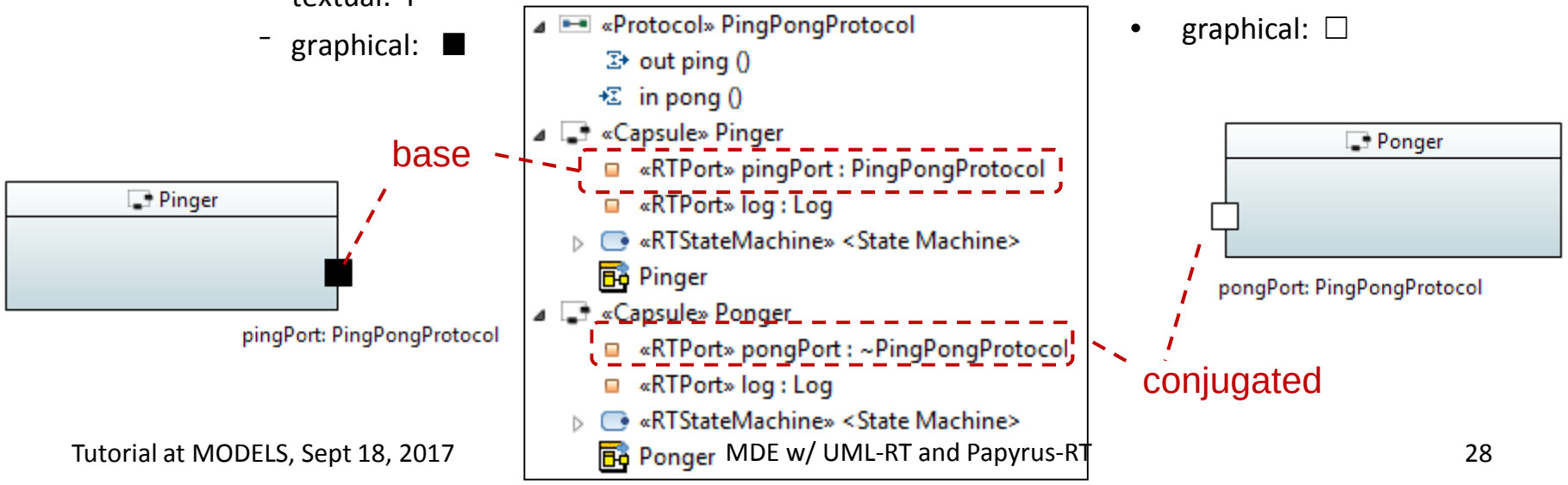






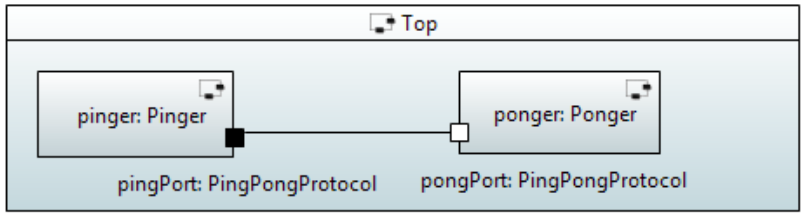
Ports

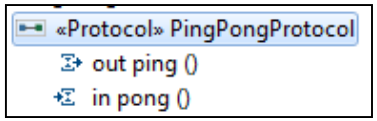
- "Boundary objects" owned by capsule
- Typed over a protocol P
- Have '**send**' operation
  - `portName.msg(arg1,...,argn).send()`
- Can be
  - base (not conjugated)
    - Direction of messages is as declared in protocol
    - Notation:
      - textual: P
      - graphical: ■
  - conjugated
    - Direction of messages declared in protocol is reversed
    - Notation
      - textual: ~P
      - graphical: □

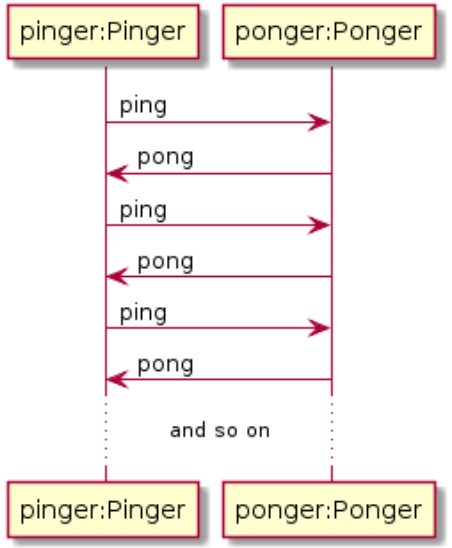


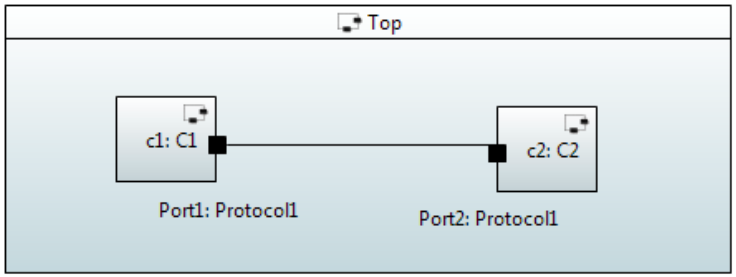
Connectors

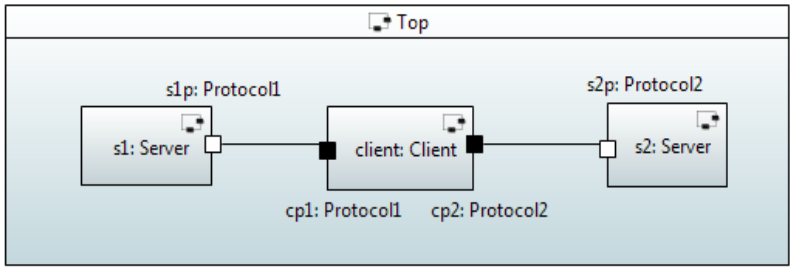
- Connect two ports
- Ports must be compatible
  - Both are instances of same protocol
  - Either (asymmetric)
    - one is 'base' (i.e., not 'conjugated')
      - typically owned by 'client'
    - and the other is 'conjugated'
      - typically owned by 'server'
  - Or (symmetric)
    - only InOut messages





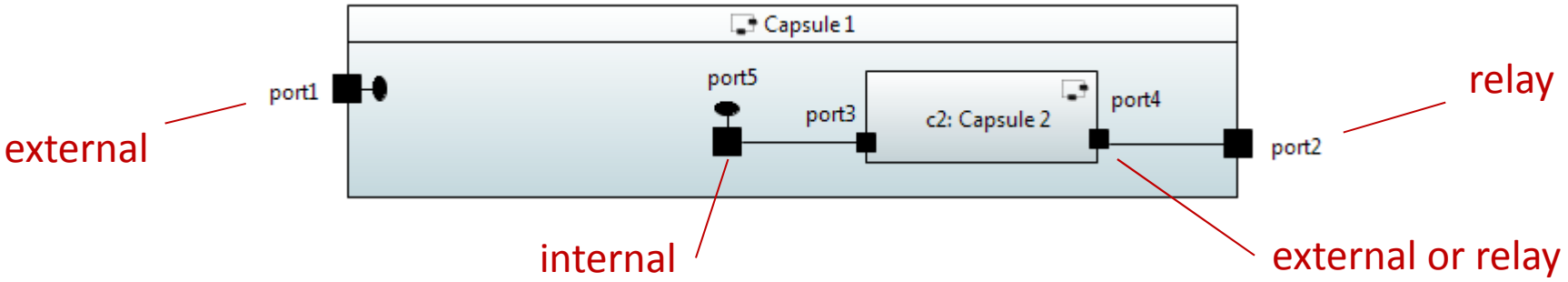




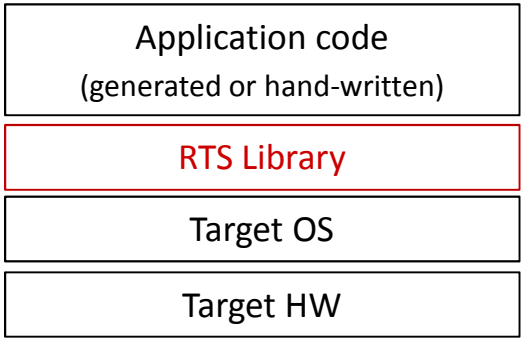


Ports: External, Internal, Relay

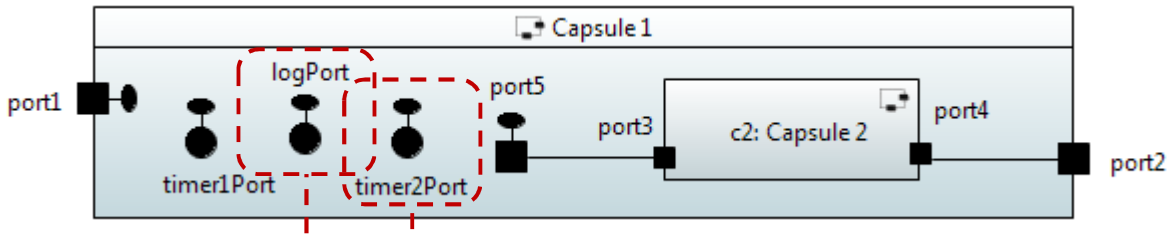
- External behaviour
  - Provides (part of) externally visible functionality (isService=true)
  - Incoming messages passed on to state machine (isBehaviour=true)
  - Must be connected (isWired=true)
- Internal behaviour
  - As above, but not externally visible (isService=false)
  - Connect state machine with a capsule part
- Relay
  - Pass external messages to and from capsule parts



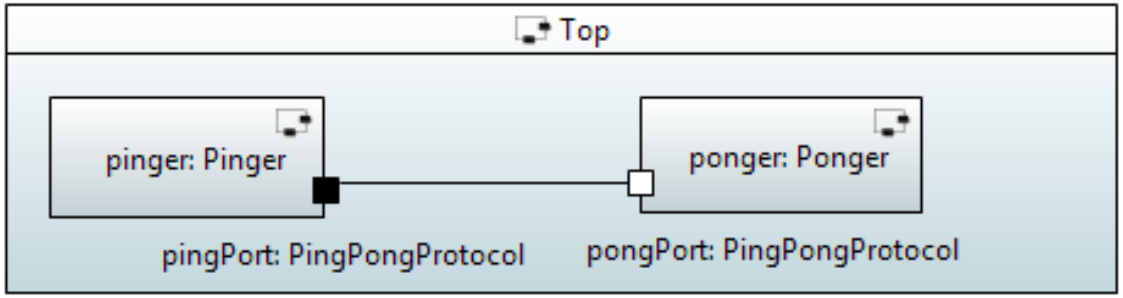
Ports: System

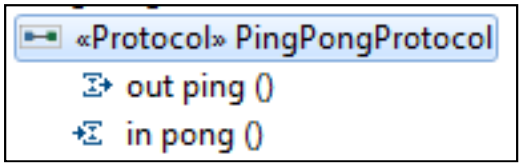


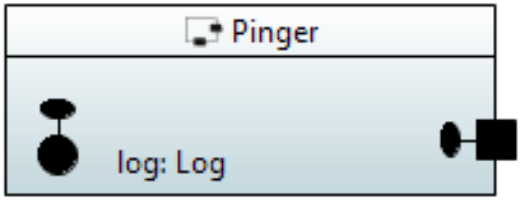
- Connects capsule to Runtime System (RTS) library via corresponding system protocol
- Provides access to RTS services such as
  - Timing: setting timers, time out message
    - `timer2Port.informIn(UMLRTTimespec(10, 0));`
      `// set timer that will expire in 10 secs and 0 nanosecs`
    - When timer expires, 'timeout' message will be sent over `timer2Port`
  - Log: sending text to console
    - `logPort.log("Ready to self-destruct")`
  - Frame: incarnate, destroy capsule instances



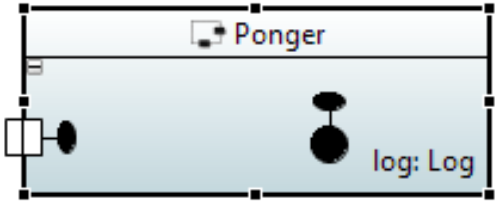
Example: PingPong





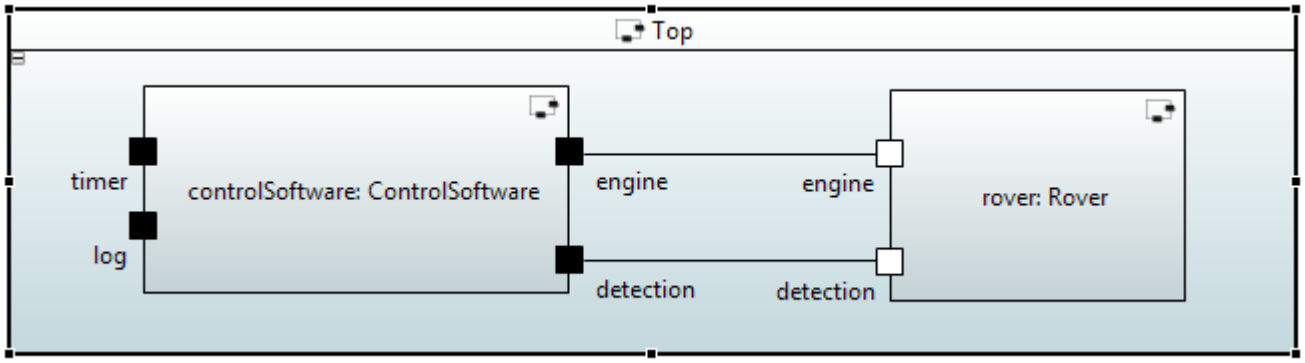


pingPort: PingPongProtocol



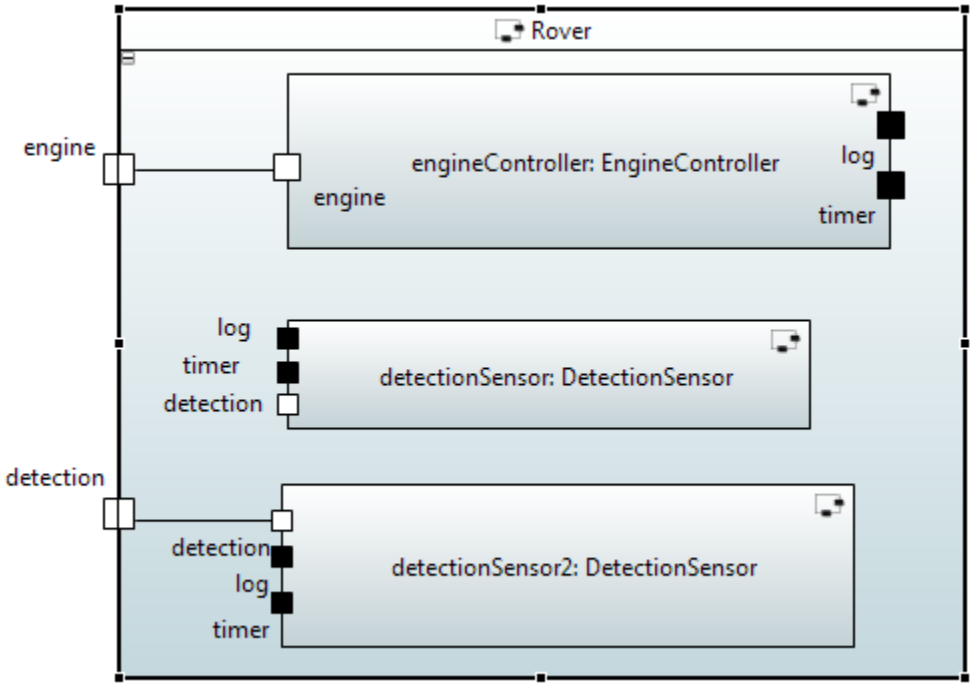
pongPort: PingPongProtocol

Example: Rover

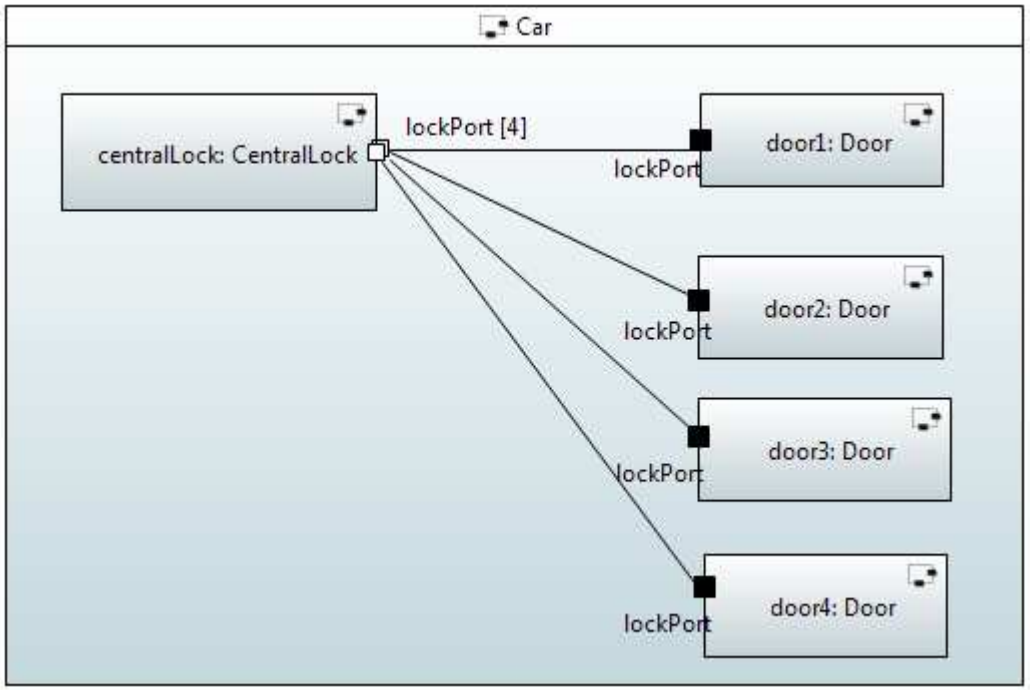


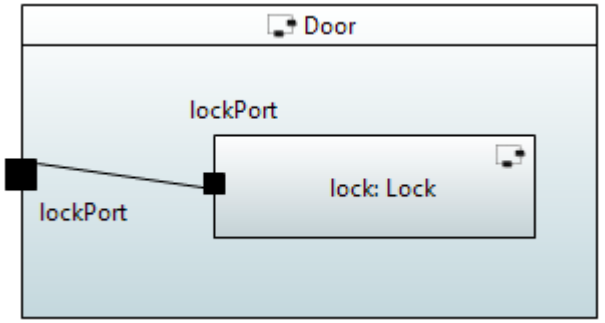
«Protocol» Engine
- out moveForward ()
- out moveBackwards ()
- out turnLeft (angle : Integer)
- out turnRight (angle : Integer)
- out stop ()
- in turnedLeft ()
- in turnedRight ()
- in stopped ()

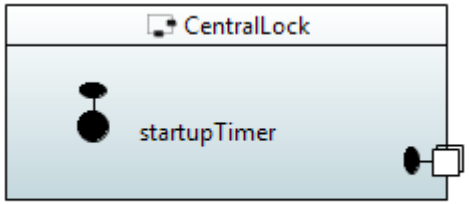
«Protocol» Detection
- out startDetection ()
- out stopDetection ()
- in obstacleDetected (distance : Real)

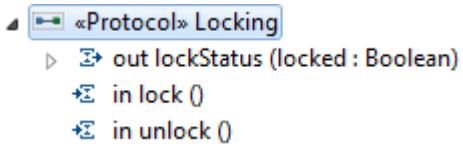


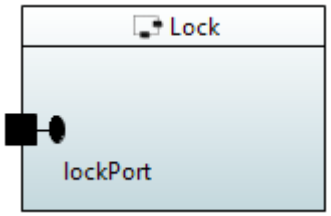
Example: Door Lock System

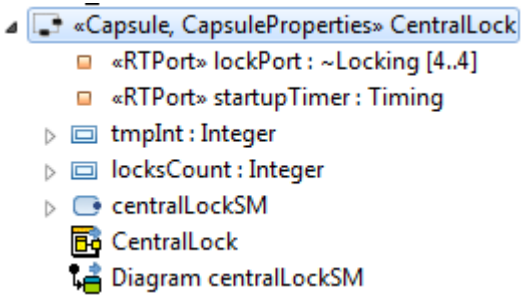












UML-RT Part I

- Core concepts
  - Structural modeling
  - Behavioural modeling

State Machines

States

- Capture relevant aspects of history of object
- Determine how object can respond to incoming messages
- May have invariants associated with them

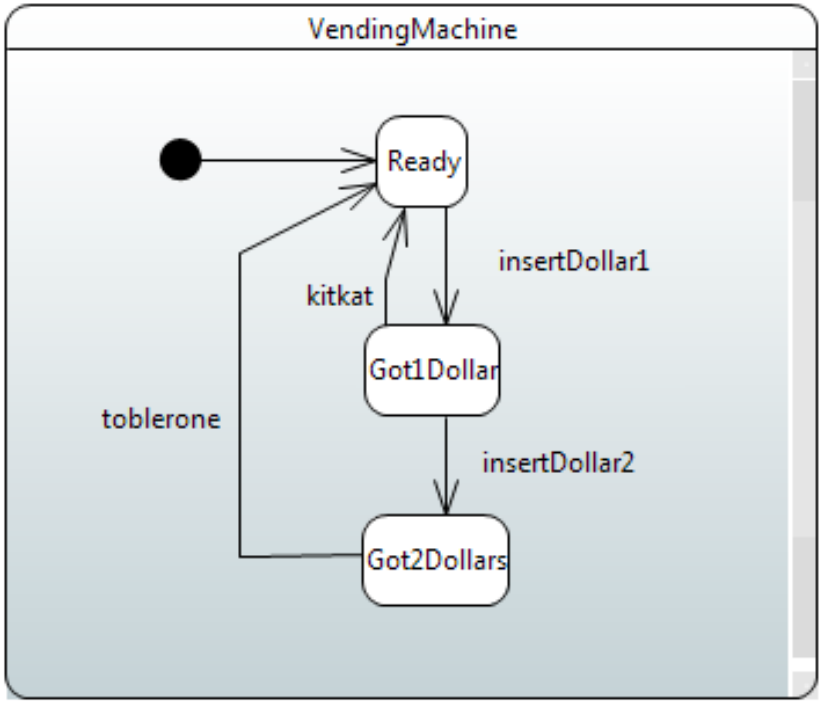
Pseudo states

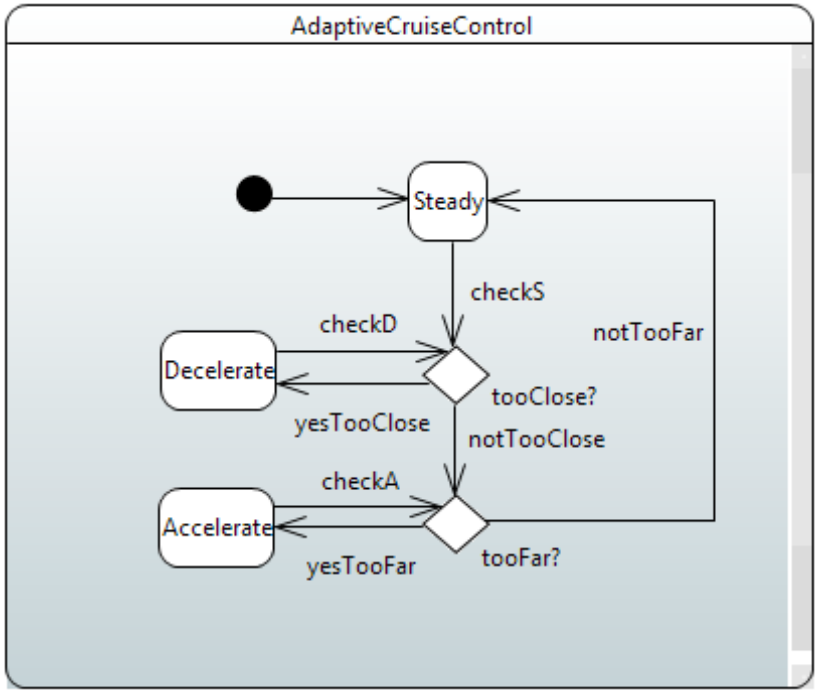
- Don't belong to description of lifetime of object

  $\Rightarrow$ object cannot be 'in' a pseudo state
- Helper constructs to define complex state changes

Transitions

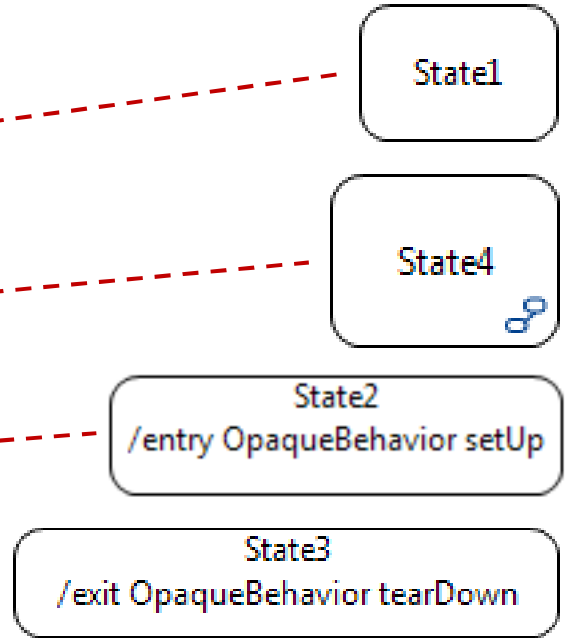
- Describe how object can move from one state to next in response to message input





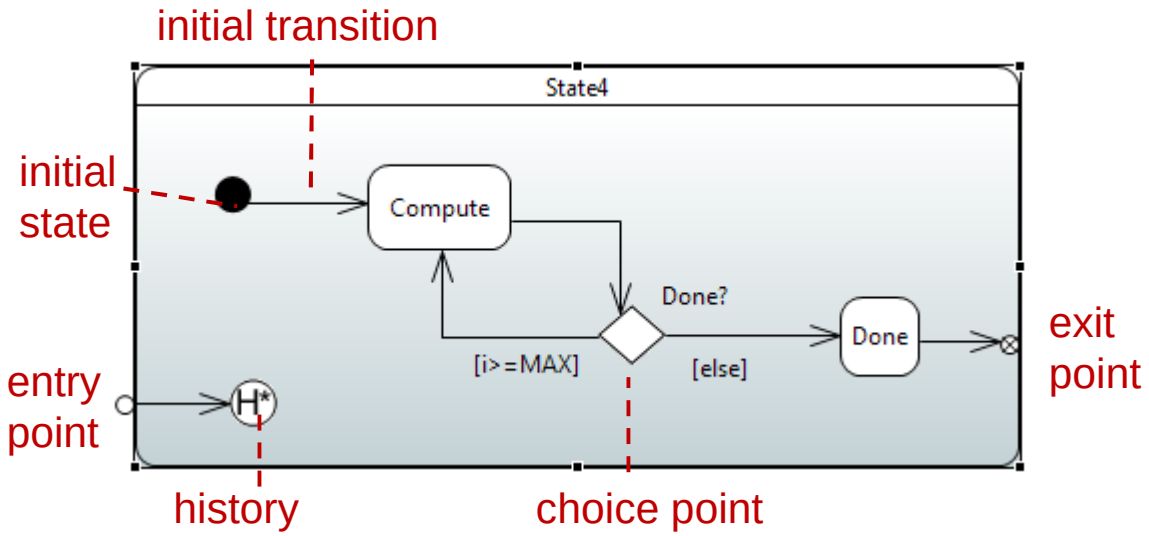
States and Pseudo States

- States
  - Kinds:
    - Basic
    - Composite (in hierarchical state machines)
  - May contain
    - Entry action (written in action language)
    - Exit action (written in action language)



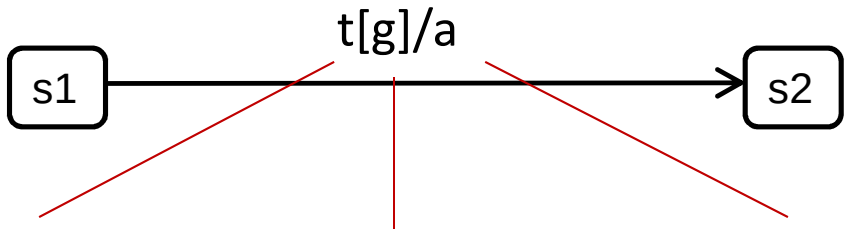
- Pseudo states
  - Initial
  - choice point
  - history
  - entry points
  - exit point

in composite states only



Transitions

- Kinds:
  - Basic
  - Group (in hierarchical state machines)
- Consists of
  - Triggers
    - Transitions out of pseudo states (initial, choice) don't have triggers
    - Transitions out of non-pseudo state should have at least one trigger
  - Guards (optional, written in action language)
  - Effect/Actions (optional, written in action language)



Trigger

Specifies port and message

Guard

Boolean condition that must hold

Effect/Actions

Code that is executed when transition is taken

Action Language

- Language used in
  - guards to express Boolean expressions
  - entry action, exit action, transition effects to read and update attribute values, send messages
- Typically: C/C++, Java

⇒ State machines are a hybrid notation combining

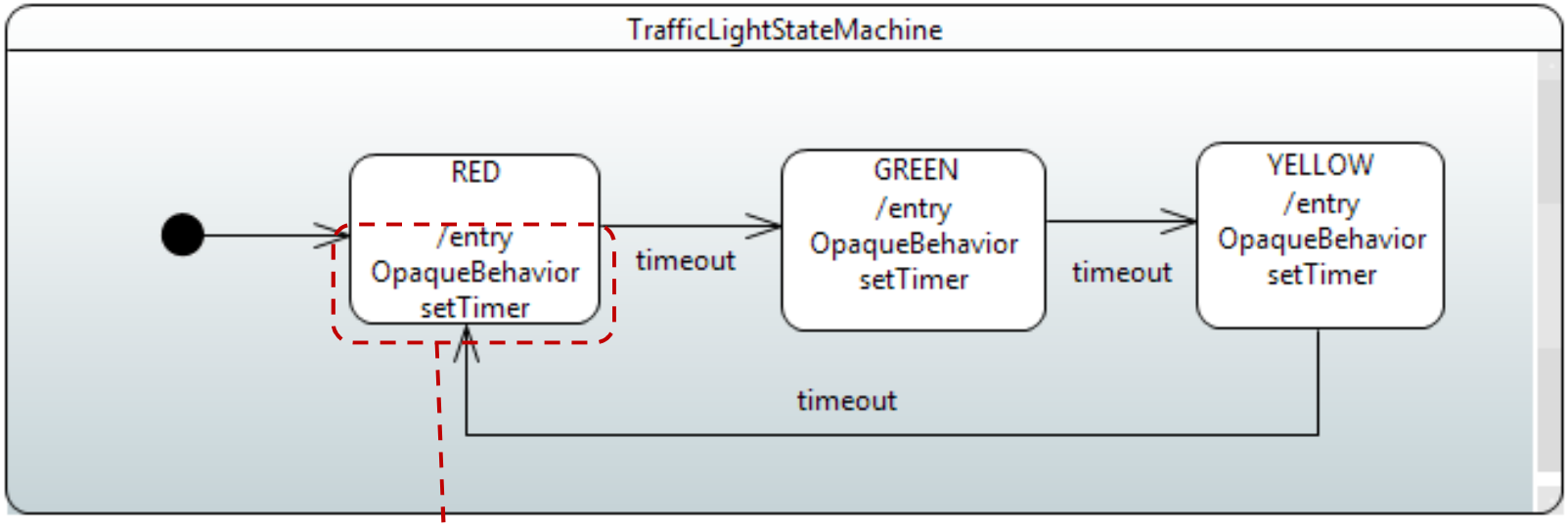
- graphical notation for state machines and
- textual notation for source code in actions

⇒ UML and UML-RT State Machines

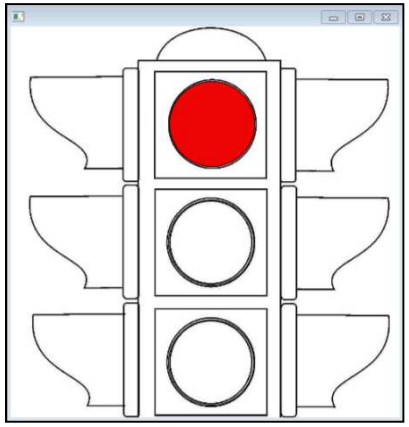
- different from, e.g., Finite Automata
- closer to 'extended hierarchical communicating state machines' [6]

[6] R. Alur. Formal Analysis of Hierarchical State Machines. Verification: Theory and Practice. 2003.

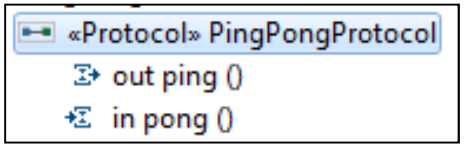
Example: Action Code, Timers, Logging

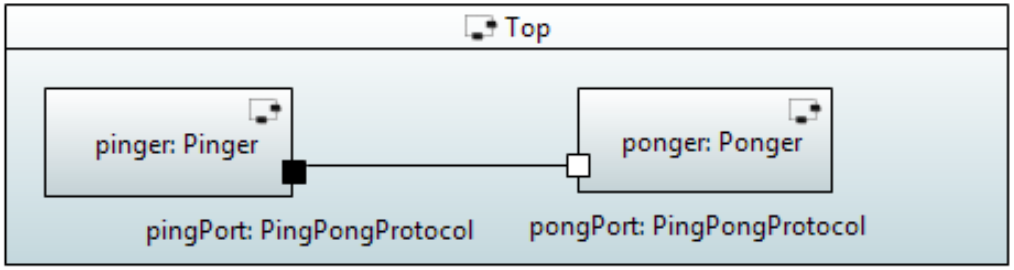



```
timer.informIn(UMLRTTimespec(5,0));
log.log("Switched to red");
```

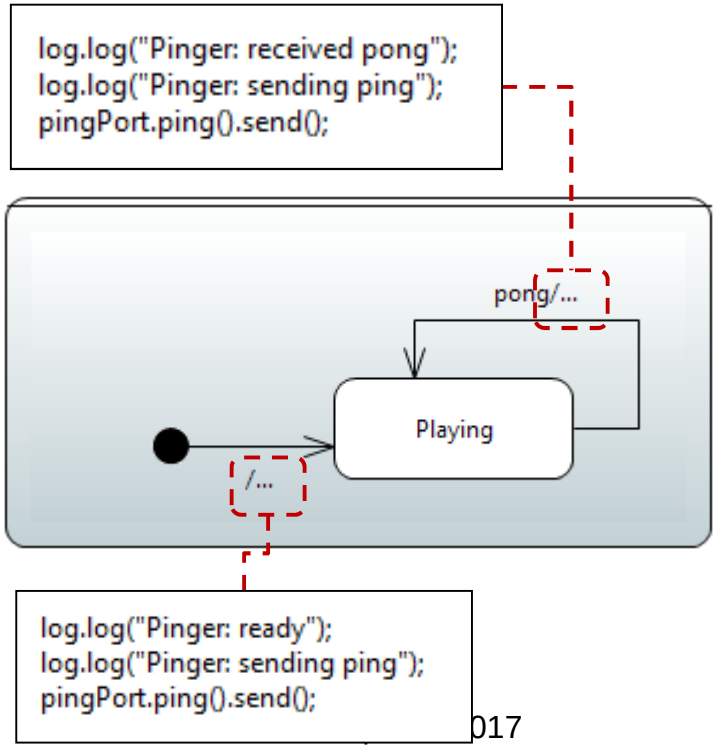


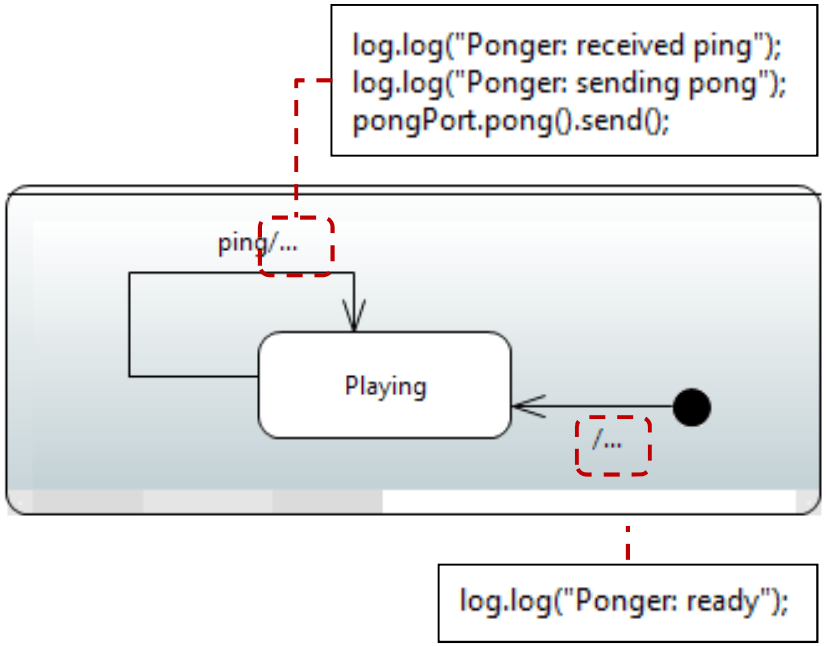
Example: Ping Pong



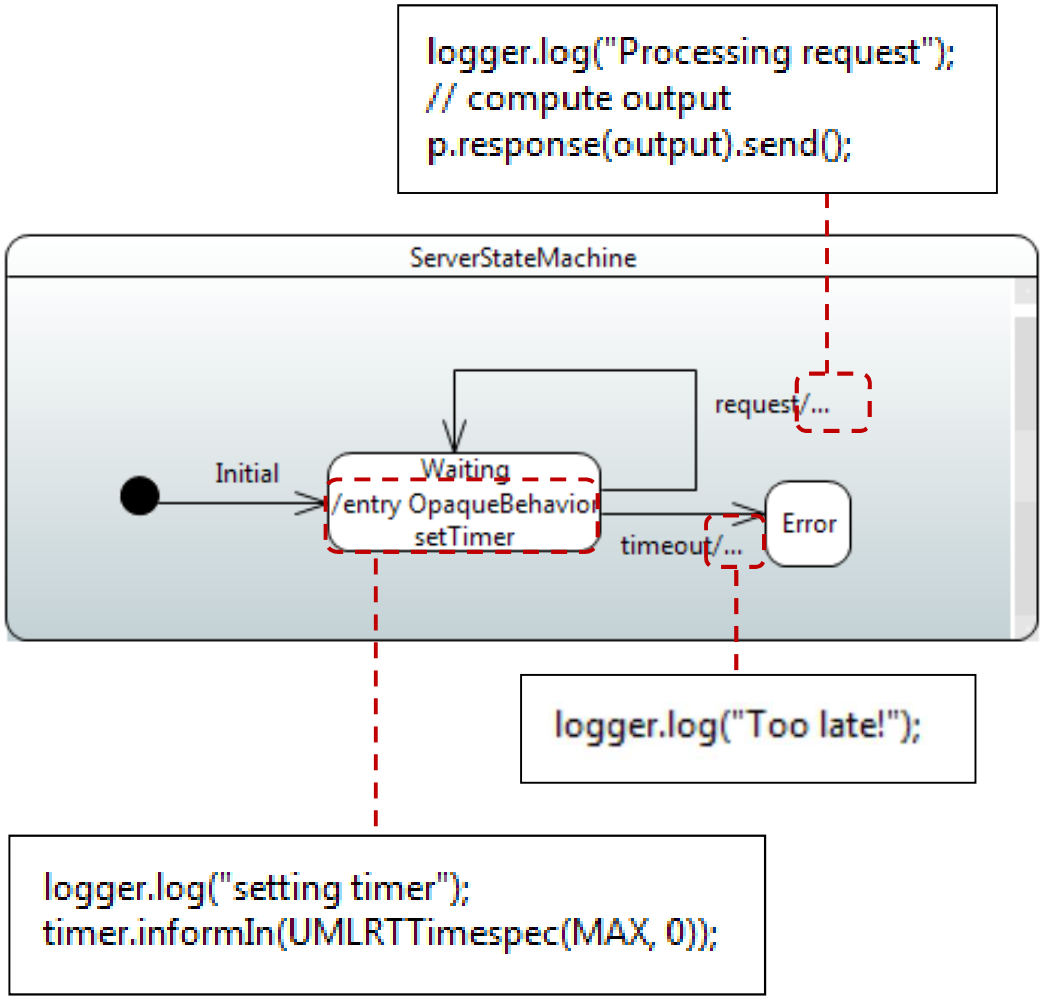


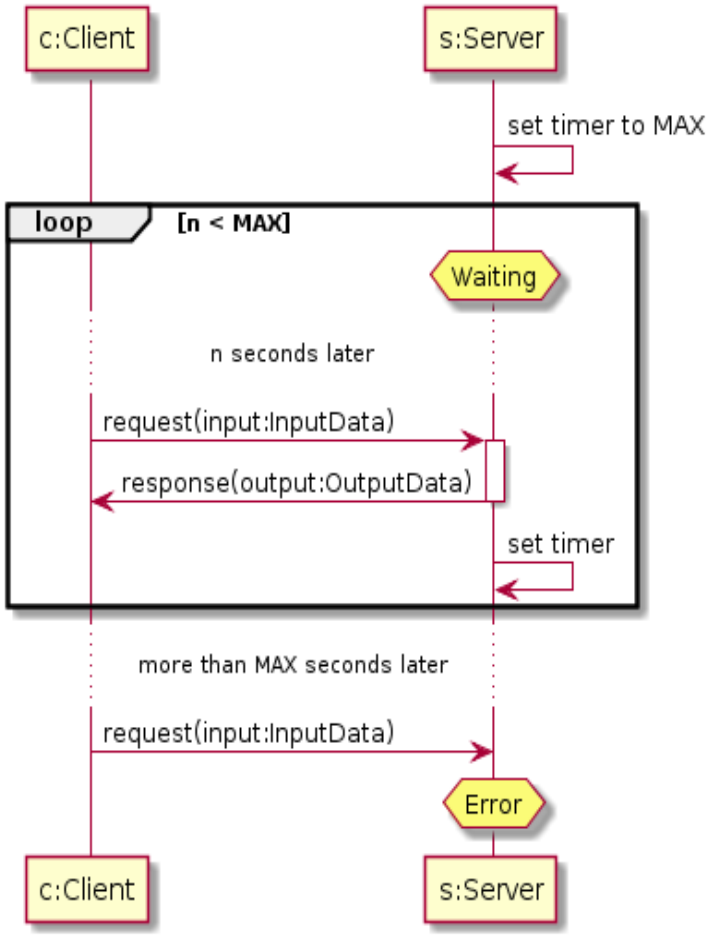

```
$ ./TopMain.exe
Controller "DefaultCon
Pinger: ready
Pinger: sending ping
Ponger: ready
Ponger: received ping
Ponger: sending pong
Pinger: received pong
Pinger: sending ping
Ponger: received ping
Ponger: sending pong
Pinger: received pong
Pinger: sending ping
Ponger: received ping
Ponger: sending pong
Pinger: received pong
Pinger: sending ping
```





Example: Timers





Overview

2 – 3:30pm:

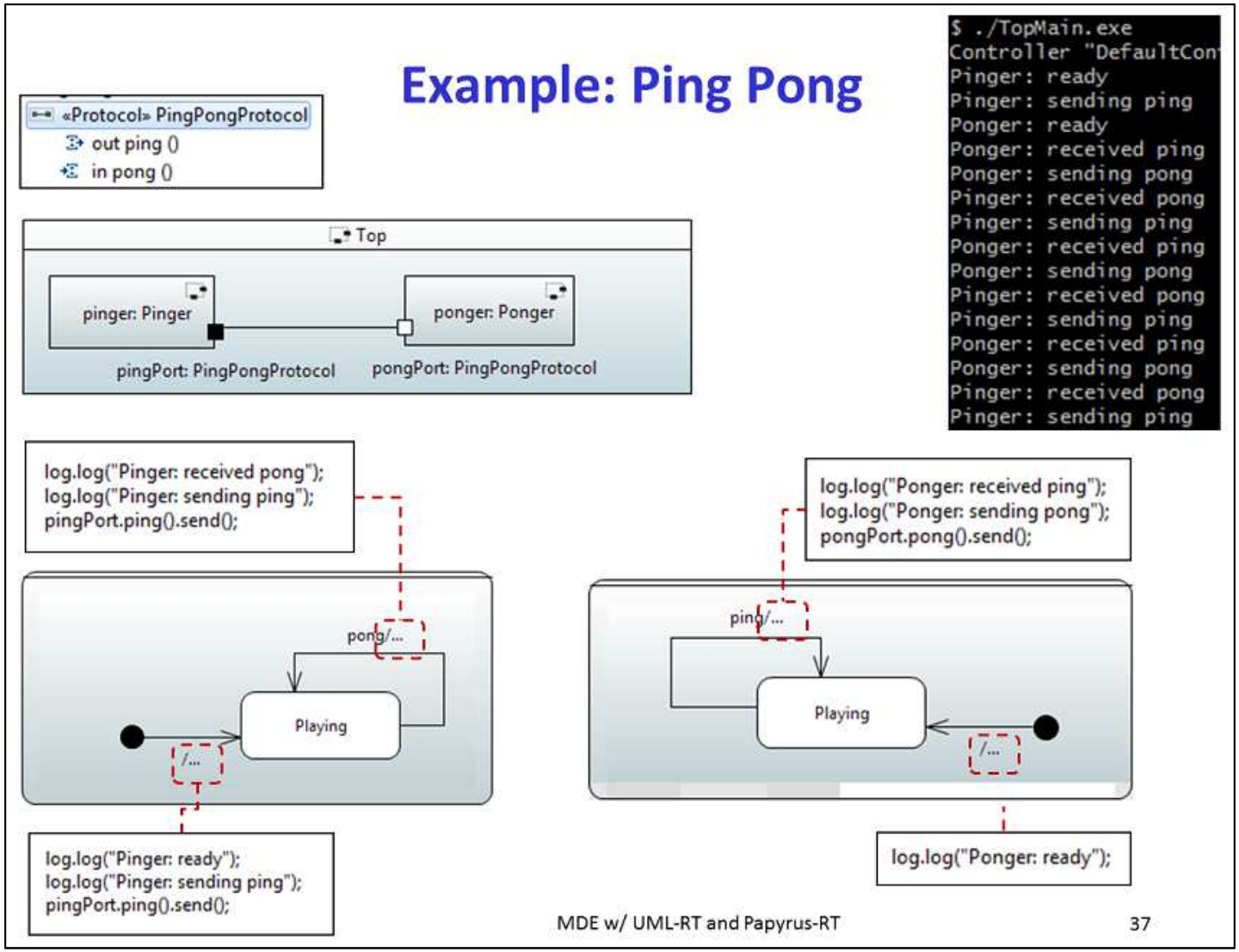
1.	Intro		
2.	MDE	(10 mins)	(10 slides)
3.	Overview	(1 min)	(1 slide)
4.	Papyrus-RT	(10 mins)	(3 slides)
	• Hands on (installation)		
5.	UML-RT: Part I	(60 mins)	(24 slides)
	• Core concepts		
	• Demo and hands on		

3:30 – 4pm: Coffee break

4 – 5:30pm:

6.	UML-RT: Part II	(60 mins)	(14 slides)
	• More advanced concepts		
	• Demo and hands on		
7.	Ongoing and future work	(10 mins)	(5 slides)
8.	Conclusion	(10 mins)	(3 slides)

Demo and Hands on



Overview

2 – 3:30pm:

1. Intro
2. MDE (10 mins) (10 slides)
3. Overview (1 min) (1 slide)
4. Papyrus-RT (10 mins) (3 slides)
   - Hands on (installation)
5. UML-RT: Part I (60 mins) (24 slides)
   - Core concepts
   - Demo and hands on

3:30 – 4pm: Coffee break

4 – 5:30pm:

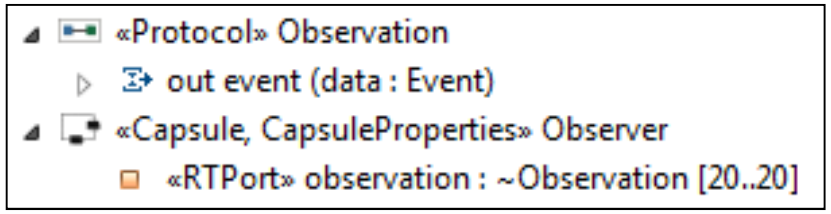
6. UML-RT: Part II (60 mins) (14 slides)
   - More advanced concepts
   - Demo and hands on
7. Ongoing and future work (10 mins) (5 slides)
8. Conclusion (10 mins) (3 slides)

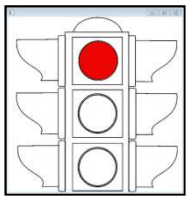
UML-RT Part II

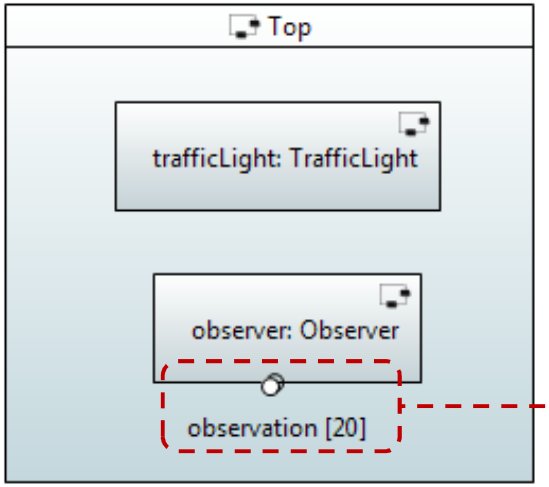
- More on ports
- More on state machines

Ports: SPP and SAP

- So far, only wired ports
  - Connected automatically when instances are created
- Unwired ports
  - Connected at run-time
  - Publish/subscribe
    - Port on publisher: Service Provision Point (SPP)
    - Port on subscriber: Service Access Point (SAP)
    - Register with RTS using unique service name (manually or automatic)

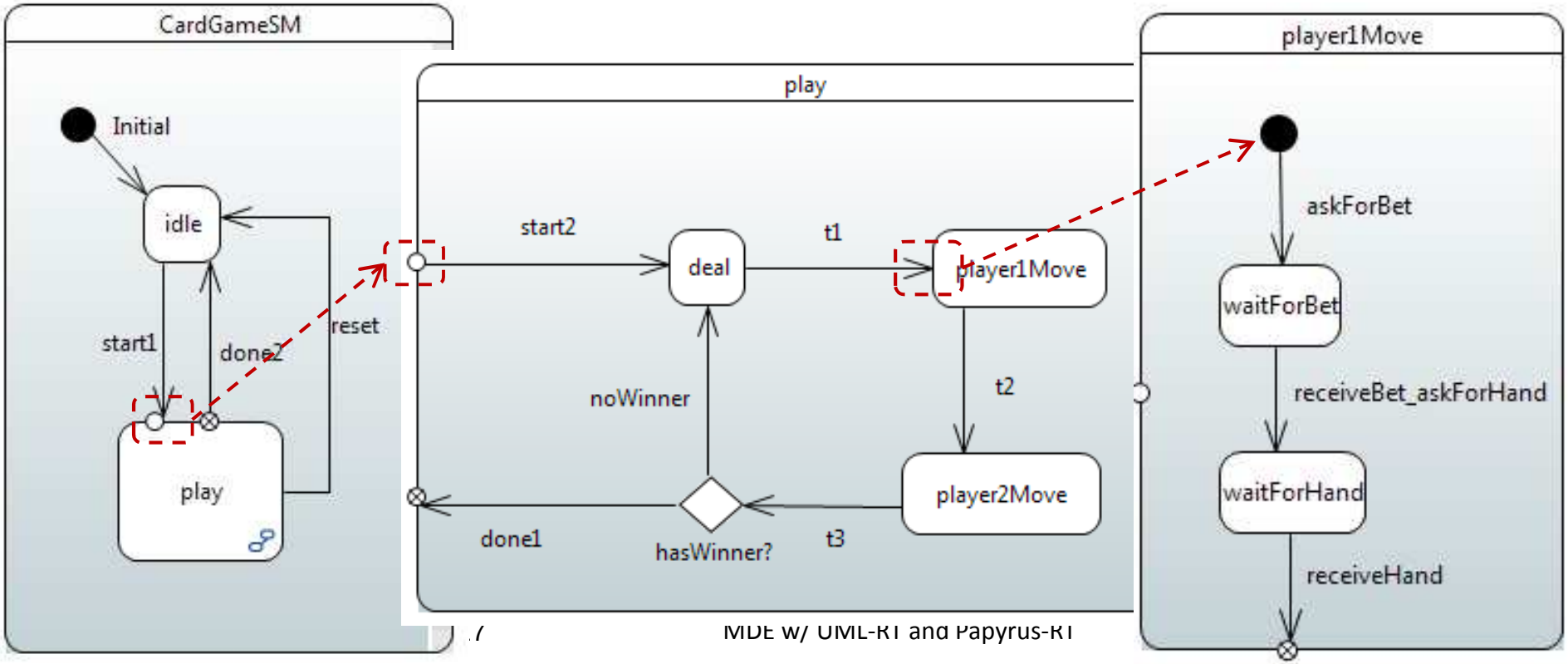






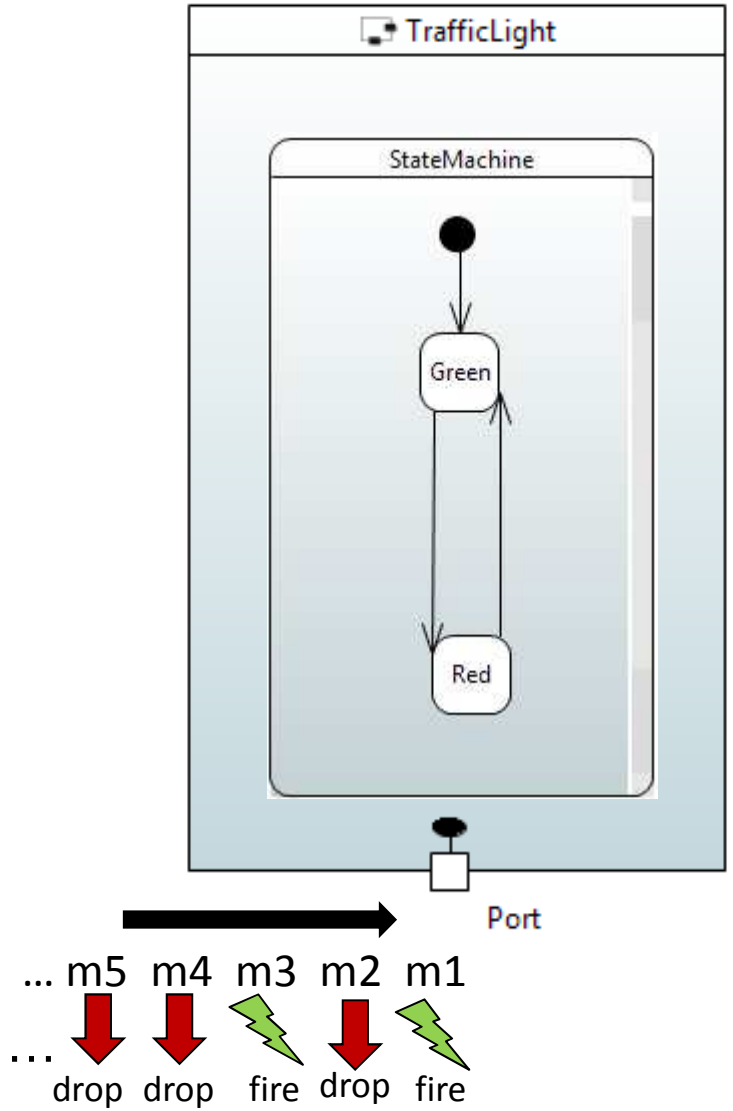
State Configuration

- States can be active: flow of control resides at state
- If a substate is active, its containing superstate is, too
- State configuration: list of active states
- Stable state configuration: no pseudo states and ends in basic state
- Example: <‘play’, ‘player1Move’, ‘waitForHand’>



Transition Execution

1. Machine in stable state configuration
2. Message m1 has arrived and is dispatched
3. If dispatching enables no transition, m1 is 'dropped'
4. If dispatching enables transition t,
   - source state of t active,
   - message matches trigger of t, and
   - guard evaluates to 'true'
5. then transition t executed
   a. execute exit action of source state of t (if any)
   b. execute action code of t (if any)
   c. execute entry code of target state of t (if any)
6. If target of t is pseudo state
   a. continue by choosing and executing outgoing transition (i.e., goto 5.)
7. Machine in stable state configuration



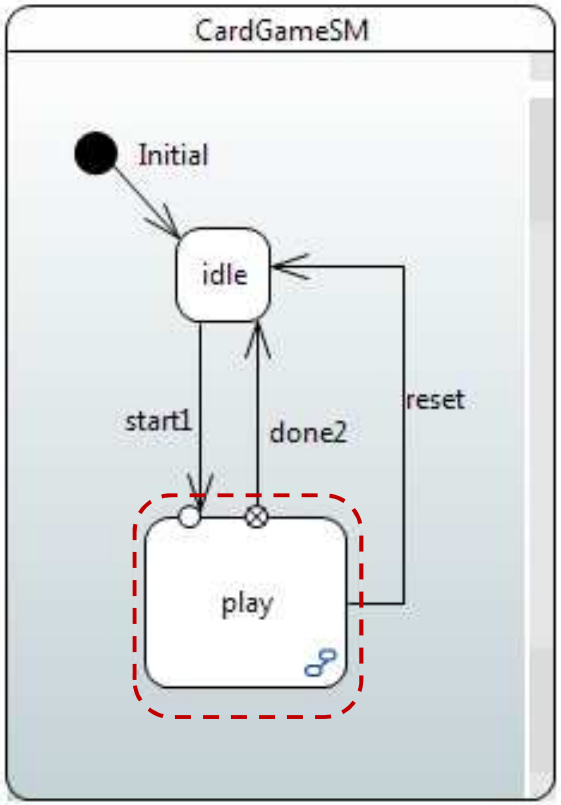
Run-to-Completion

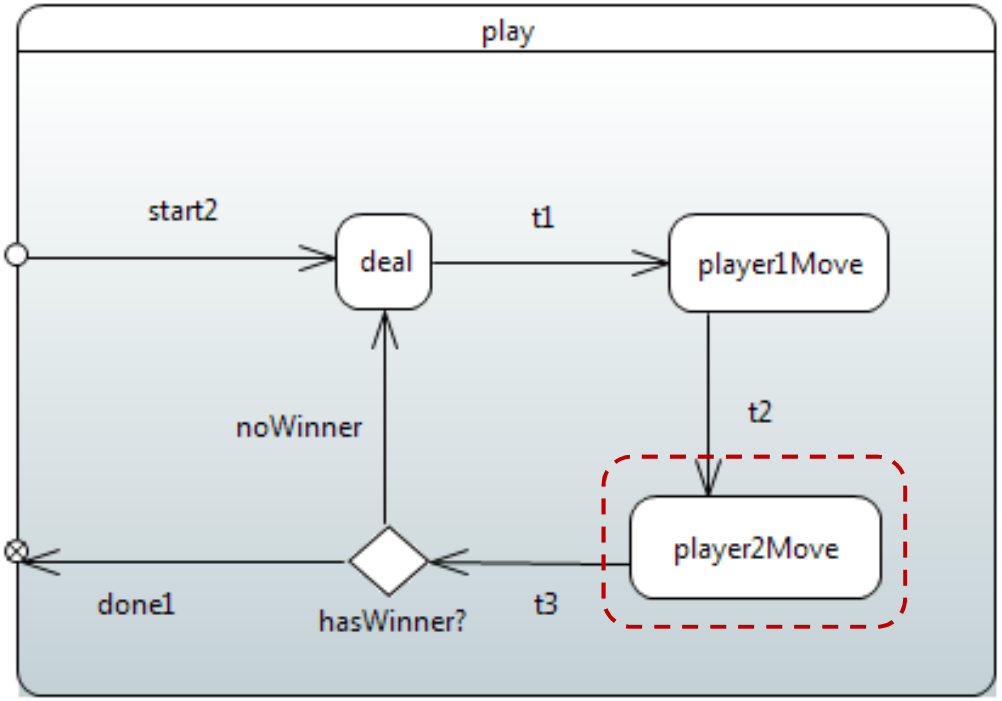
- The event processing of state machines follows 'run-to-completion' semantics
- Dispatching of message triggers execution of possibly entire chain of transitions (Steps 5 and 6 on previous slide)
- Execution lasts until stable state configuration has been reached (last state in transition chain not a pseudo state)
- During transition execution, no other message will be dispatched

$\Rightarrow$ better concurrency control

Group Transitions

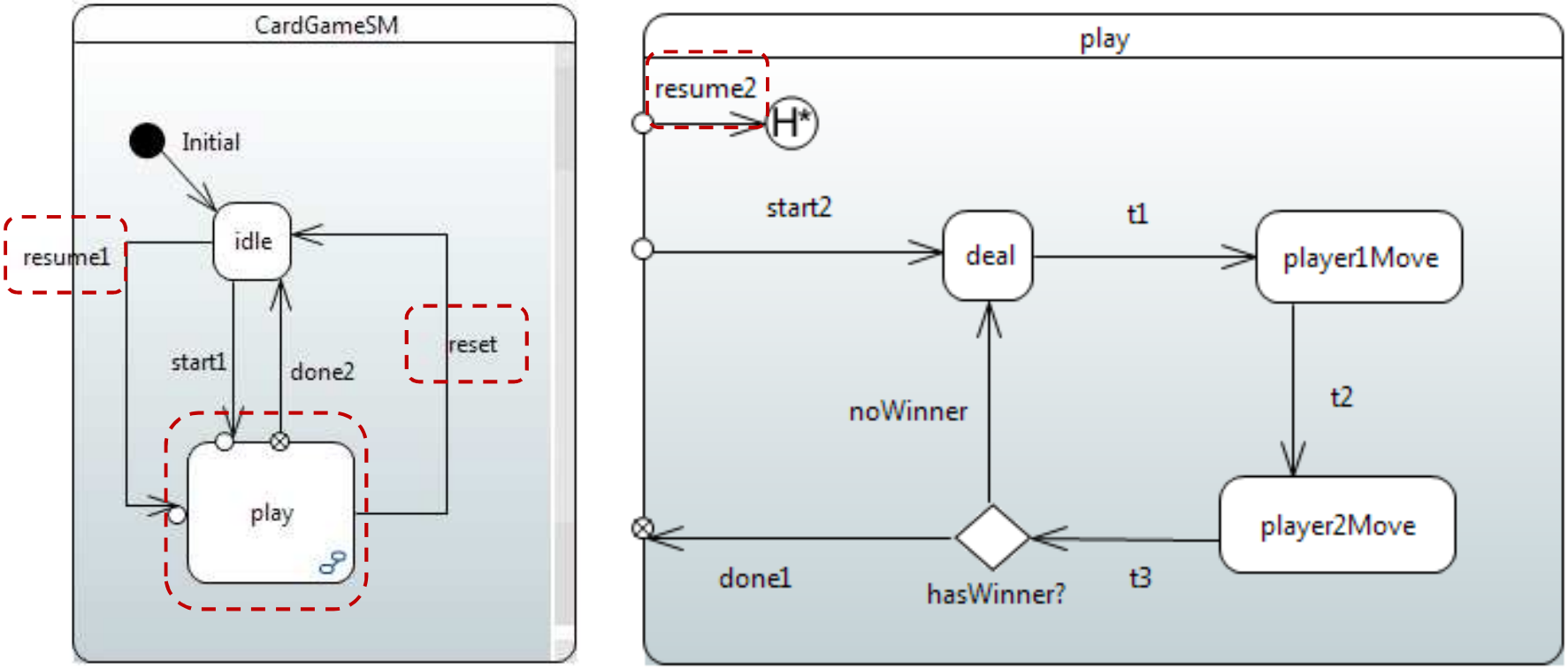
- Source state is composite
- Example:
  - Start configuration <‘play’,’player2Move’>
  - Execute transition ‘reset’:
    - exit code ‘player2Move’, exit code ‘play’, effect ‘reset’, entry code ‘idle’
  - End configuration <‘idle’>





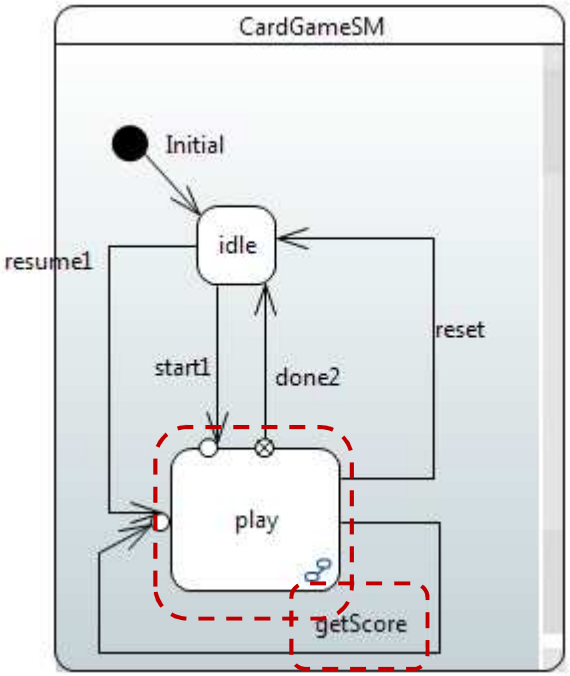
History

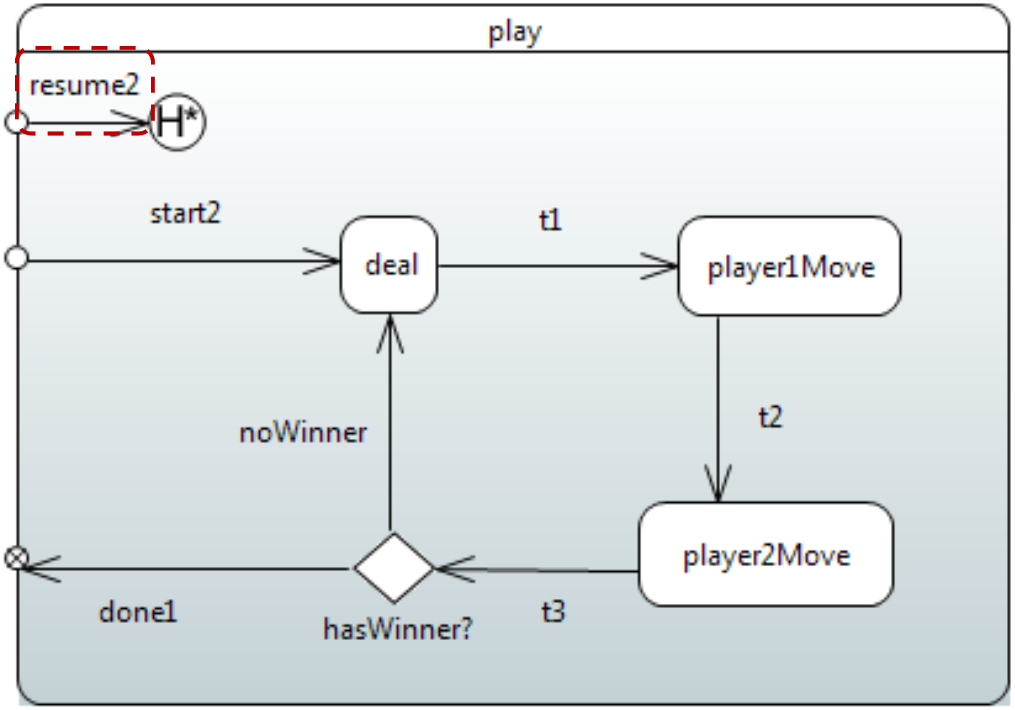
- Re-establish full state configuration that was active when containing state was active most recently
- Example: from <‘play’, s> to <‘play’, s> with ‘reset’ ‘resume1’

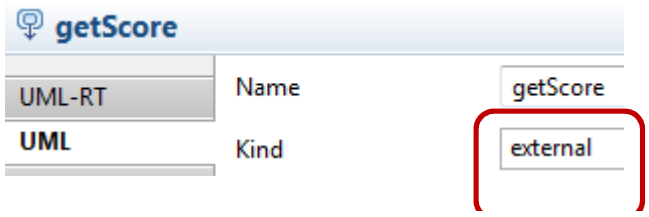


Self Transitions

- Source and target states are the same
- 2 kinds: external, internal
- External: source state (and all substates) exited and target state entered

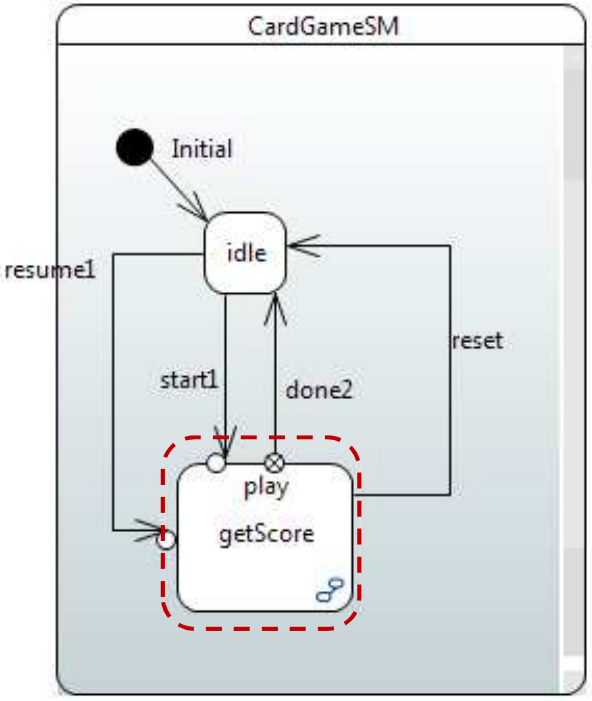


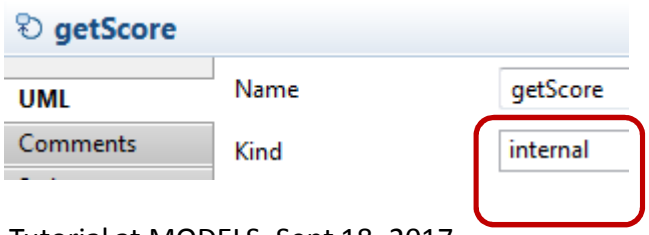


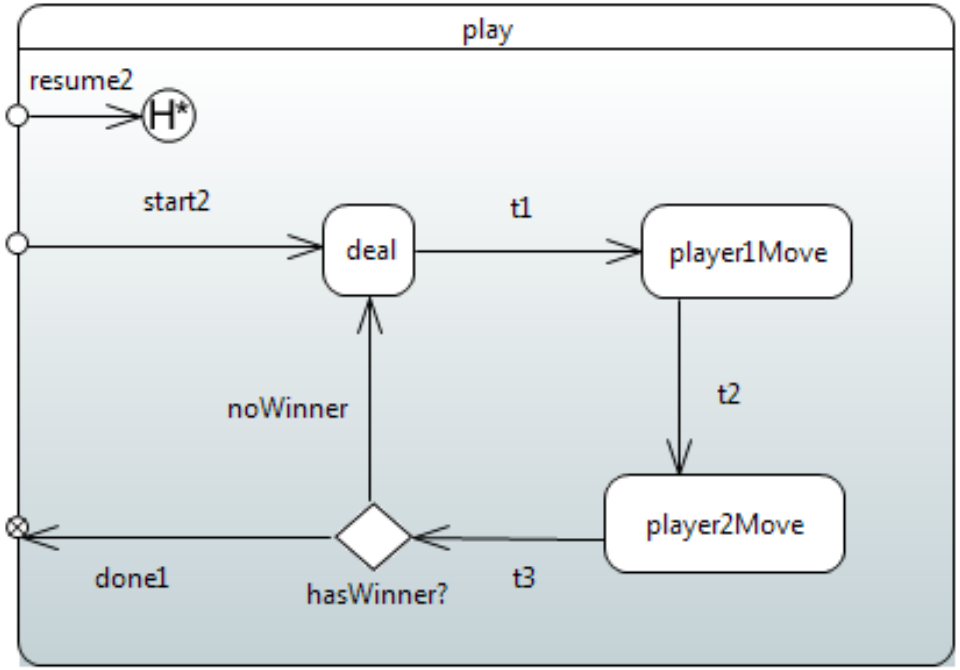


Self Transitions: Internal

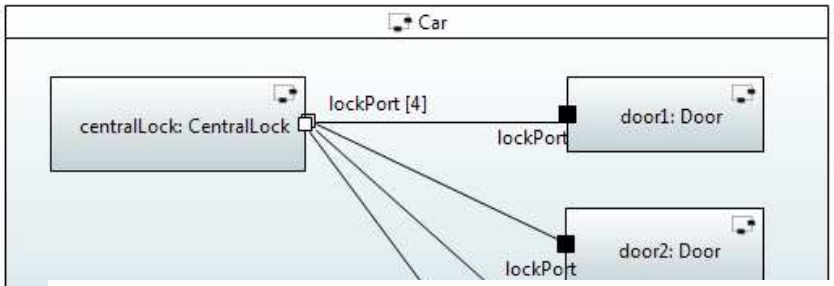
- Source state (and all substates) remain active; no exit or entry actions executed

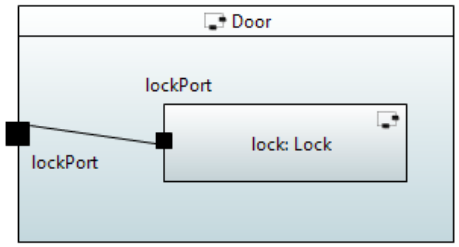


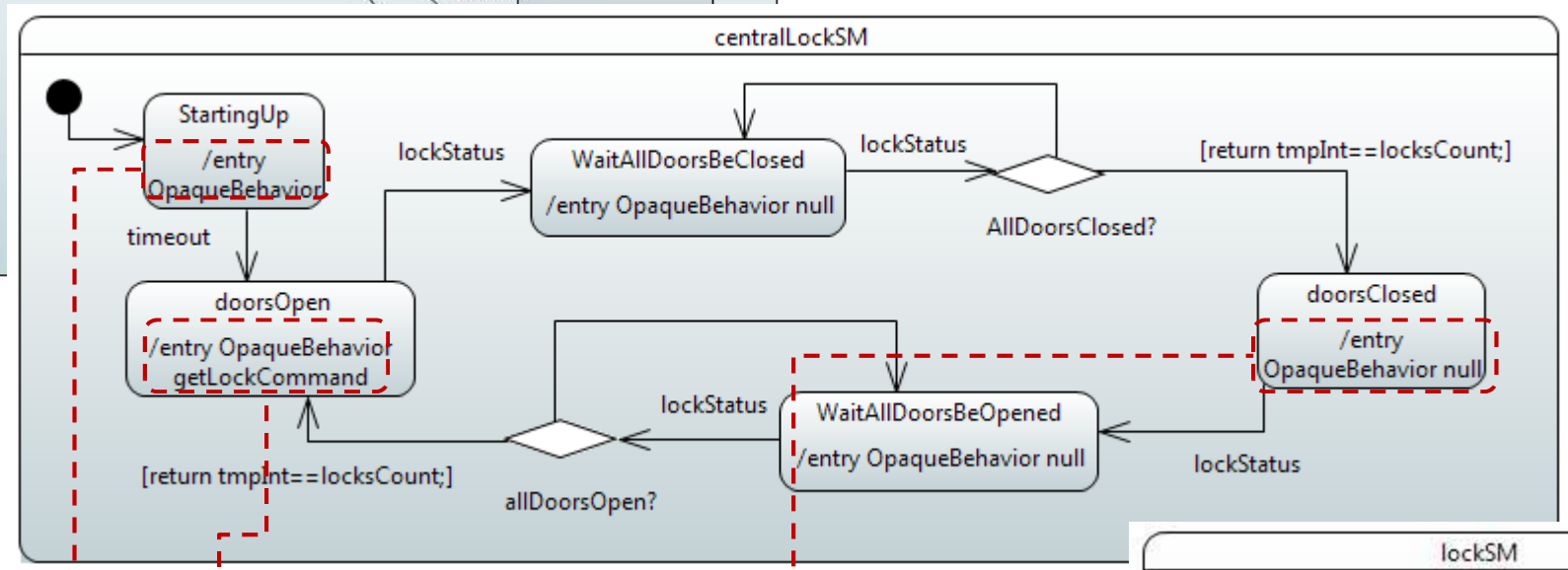


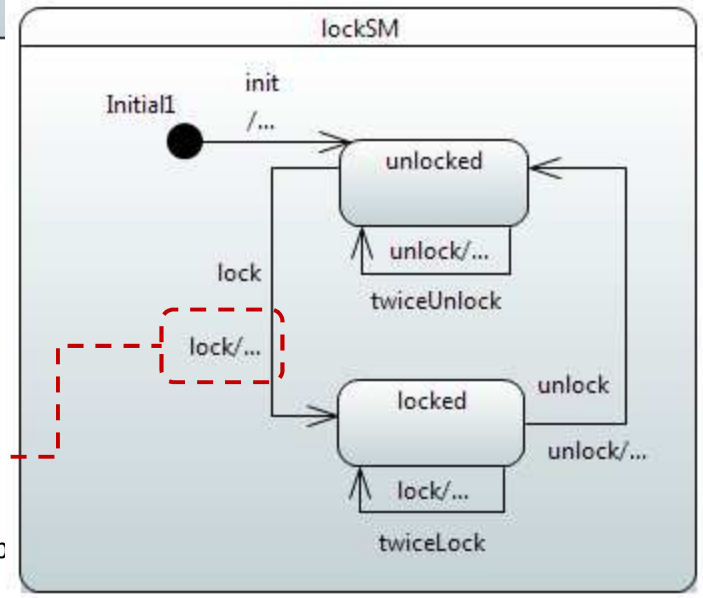


Example: Door Lock









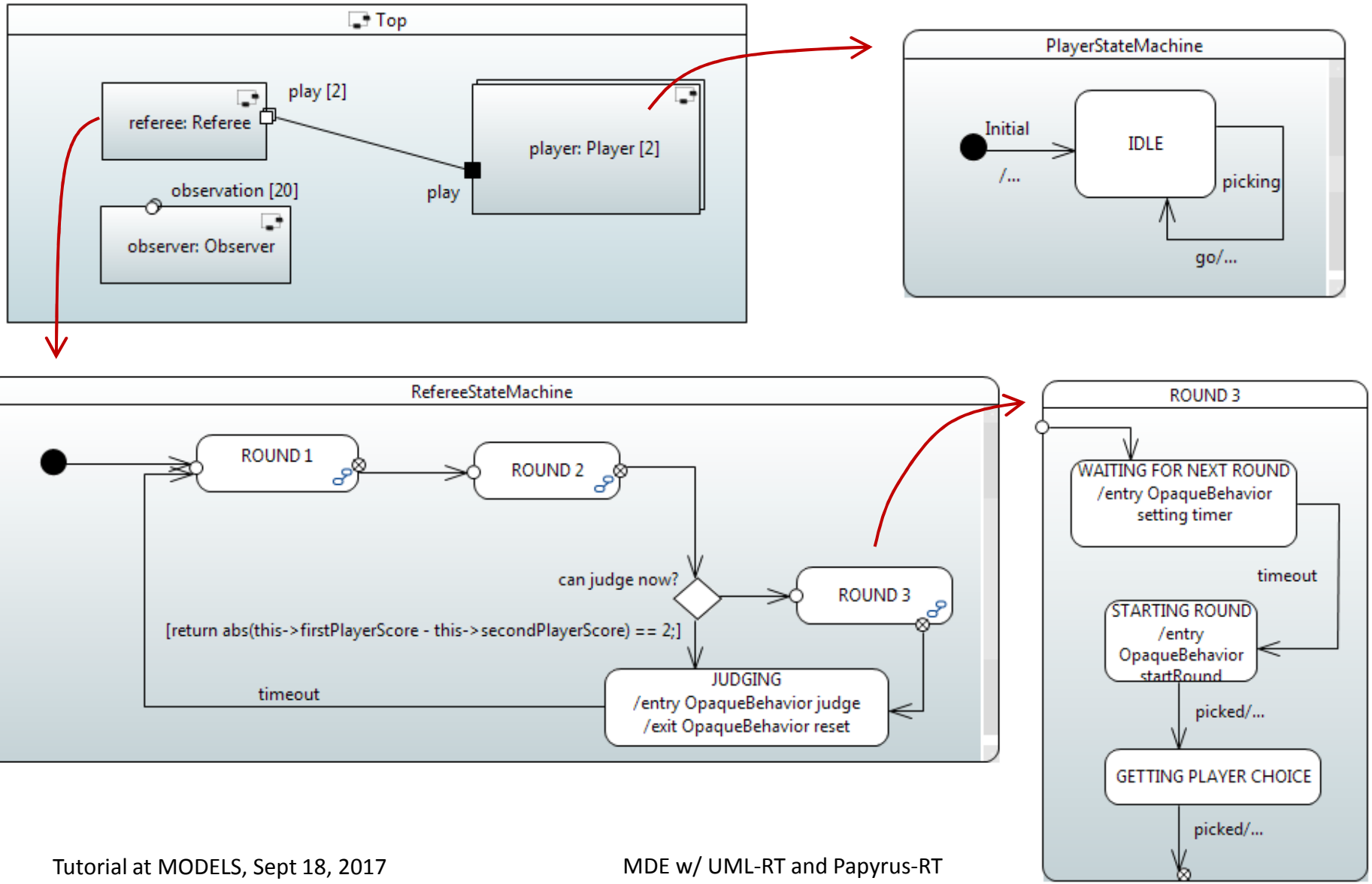

```
set timer
```

```
“doors open”;
“hit key to lock”
getchar();
lockPort.lock().send()
```

```
“doors locked”;
“hit key to open”
getchar();
lockPort.unlock().send()
```

```
“lock”+i+”locked”;
lockPort.lockStatus(true).send
```

Example: Rock/Paper/Scissors



Overview

2 – 3:30pm:

1.	Intro		
2.	MDE	(10 mins)	(10 slides)
3.	Overview	(1 min)	(1 slide)
4.	Papyrus-RT	(10 mins)	(3 slides)
	• Hands on (installation)		
5.	UML-RT: Part I	(60 mins)	(24 slides)
	• Core concepts		
	• Demo and hands on		

3:30 – 4pm: Coffee break

4 – 5:30pm:

6.	UML-RT: Part II	(60 mins)	(14 slides)
	• More advanced concepts		
	• Demo and hands on		
7.	Ongoing and future work	(10 mins)	(5 slides)
8.	Conclusion	(10 mins)	(3 slides)

Additional UML-RT Features

- Structure
  - Optional capsules
  - Inheritance
- Behaviour
  - Junction pseudo state
  - Defer/recall
  - Synchronous communication
  - Message priorities

Additional Papyrus-RT Capabilities

- Generation of multi-threaded code
  - Logical thread
    - = flow of control for capsule instance
  - Physical thread
    - Executes RTS controller
      - Oversees execution of all capsules assigned to physical thread
  - Generating single threaded code
    - 1 physical thread executing one controller executing all capsules
  - Generating multi threaded code
    - Several physical threads each executing their own controller
- Legacy model import

Overview

2 – 3:30pm:

1.	Intro		
2.	MDE	(10 mins)	(10 slides)
3.	Overview	(1 min)	(1 slide)
4.	Papyrus-RT	(10 mins)	(3 slides)
	• Hands on		
5.	UML-RT: Part I	(60 mins)	(24 slides)
	• Core concepts		
	• Demo and hands on		

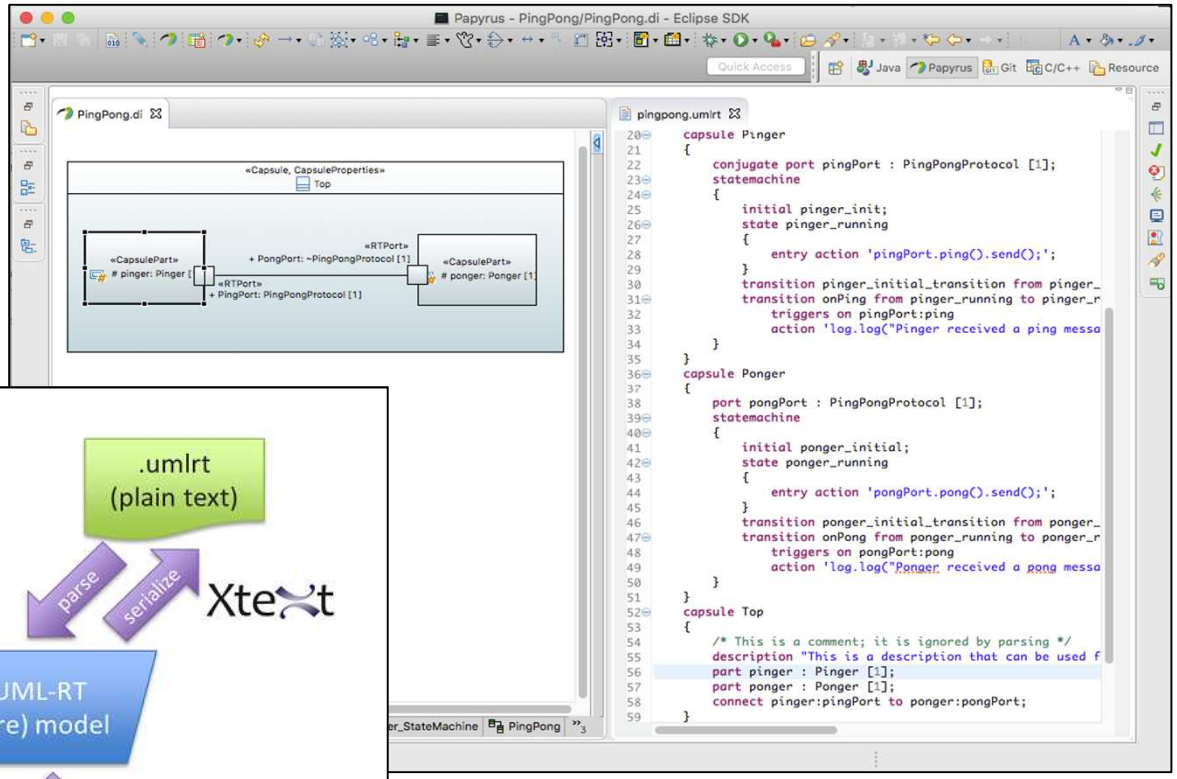
3:30 – 4pm: Coffee break

4 – 5:30pm:

6.	UML-RT: Part II	(60 mins)	(14 slides)
	• More advanced concepts		
	• Demo and hands on		
7.	Ongoing and future work	(10 mins)	(5 slides)
8.	Conclusion	(10 mins)	(3 slides)

Ongoing Work I

- Hybrid textual/graphical modeling
  - Switch seamlessly between graphical and textual representations



Ongoing Work II

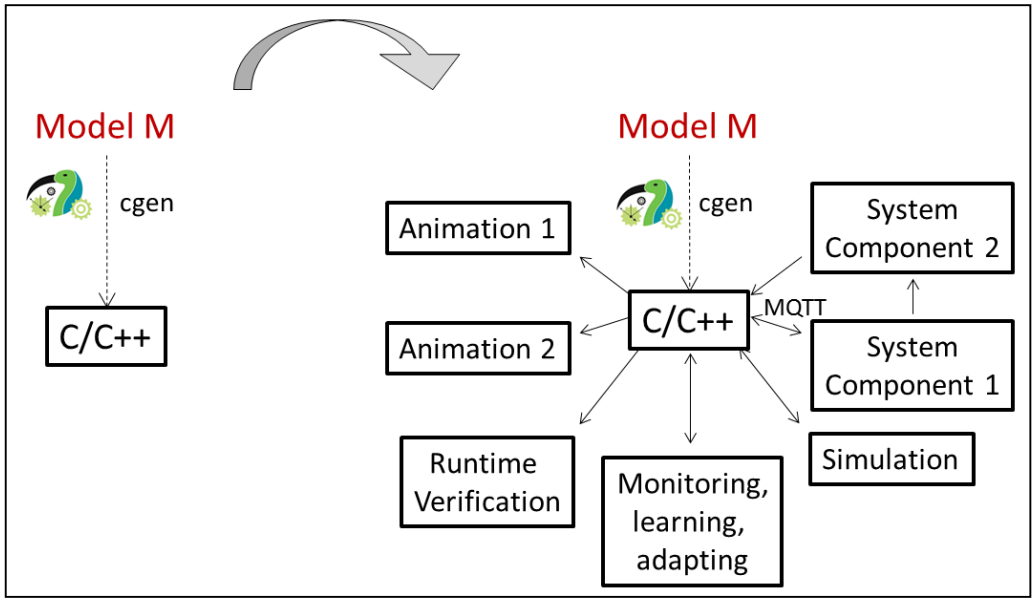
- Dedicated action language
  - Better alignment of action code and model
    - Improved user experience, analysis opportunities, etc
  - Better support for multiple target languages

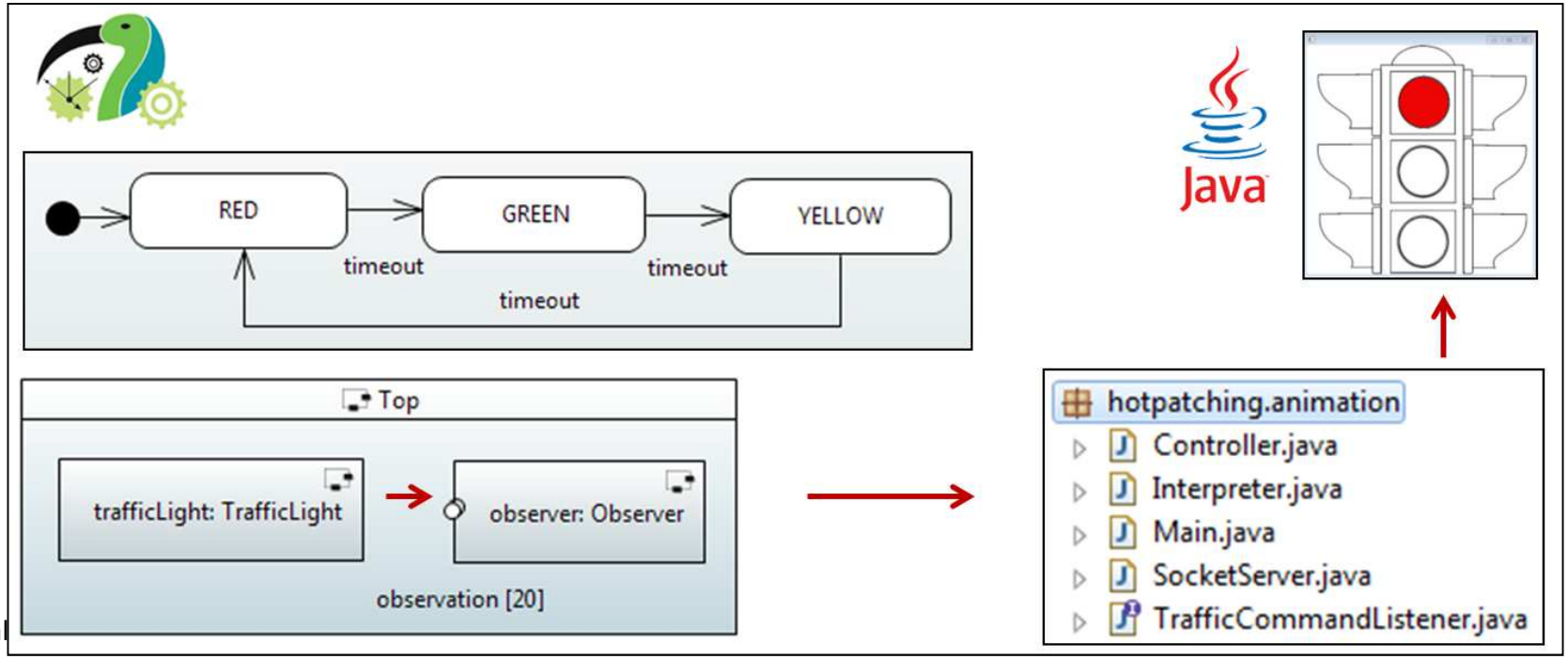
```
model PingPong {
    protocol P { inout message ball(); }
    capsule Player {
        port p : P;
    }
    capsule Top {
        part player1 : Player;
        part player2 : Player;
        connect player1:p to player2:p;
    }
}
```

```
<statement> ::=
 send <port-ref>:<message-name> <args>? ;
| incarnate <capsule-name> at <part-ref> <args>? ;
| destroy <part-ref> ;
| import <capsule-name> at <part-ref> <args>? ;
| deport <part-ref> ;
| register <port-ref> to <port-ref> ;
| deregister <port-ref> ;
| log <message> <args> ? ;
| timer <name> : inform in <expr> milliseconds ;
| timer <name> : inform every <expr> milliseconds
| cancel timer <timer-name> ;
| var <name> : <type-name> ;
| <var-name> := <expr> ;
| <var-name> . <operation-name> <args>? ;
| if <expr> then <statement> ( else <statement> )?
```

Ongoing Work III

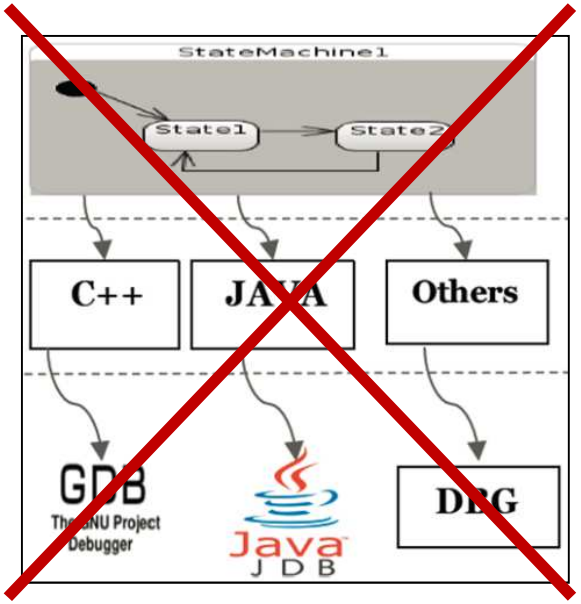
- Observer/gateway service & MQTT extension
  - Bi-directional connection between your running model and external tools

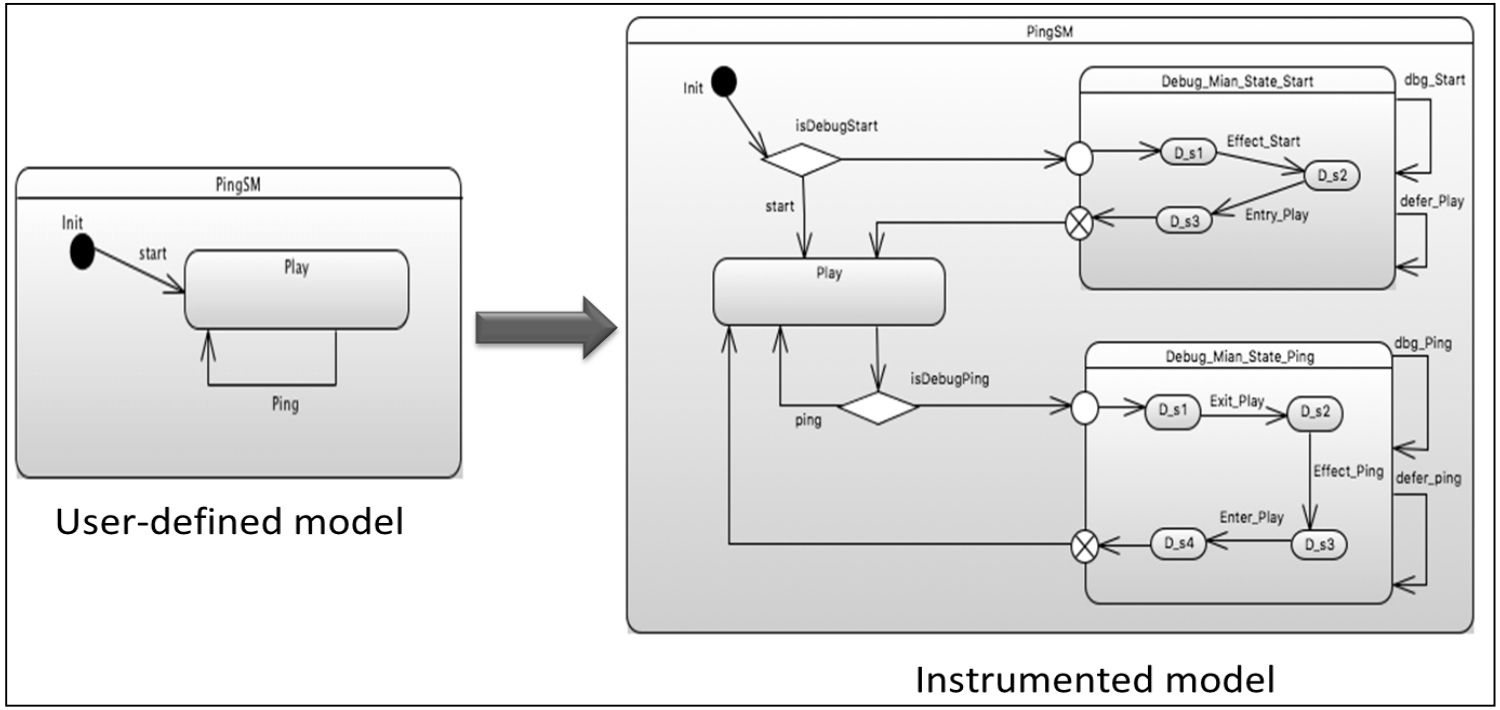




Ongoing Work IV

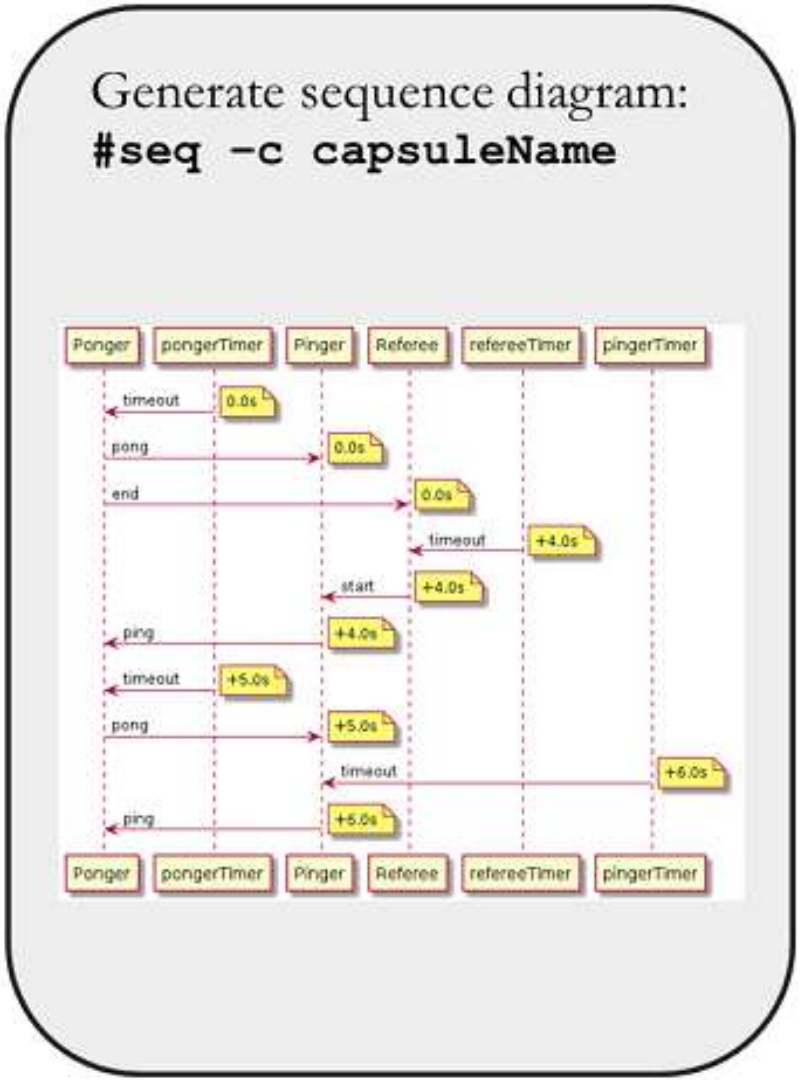
- Model-level debugging
  - No need for
    - program debugger
    - mapping info
  - Idea:
    - refine model to allow it to support debugging





Ongoing Work V

- Model interpretation/simulation
  - Use Moka model execution environment in Papyrus to execute UML-RT models
- Sequence diagram integration



Overview

2 – 3:30pm:

1.	Intro		
2.	MDE	(10 mins)	(10 slides)
3.	Overview	(1 min)	(1 slide)
4.	Papyrus-RT	(10 mins)	(3 slides)
	• Hands on		
5.	UML-RT: Part I	(60 mins)	(24 slides)
	• Core concepts		
	• Demo and hands on		

3:30 – 4pm: Coffee break

4 – 5:30pm:

6.	UML-RT: Part II	(60 mins)	(14 slides)
	• More advanced concepts		
	• Demo and hands on		
7.	Ongoing and future work	(10 mins)	(5 slides)
8.	Conclusion	(10 mins)	(3 slides)

Conclusion

- Intro to
  - MDE
    - Abstraction, Automation, Analysis
    - Core techniques to deal with complexity
  - UML-RT
    - small, proven subset of UML for real-time systems
  - Papyrus-RT
    - open-source MDE tool w/ full code generation
- Lots of opportunity to use, research, contribute
- More questions?
  - dingel@cs.queensu.ca
  - hili@cs.queensu.ca
  - eposse@zeligsoft.com

Resources and References

- Links
  - Tutorial resources: http://flux.cs.queensu.ca/mase/papyrus-rt-resources/supporting-material-for-the-models17-tutorial/
  - UML-RT: Formal semantics [7,8]
  - Papyrus-RT: https://eclipse.org/papyrus-rt
    - Installation, tutorial, etc: https://wiki.eclipse.org/Papyrus-RT/User
    - Wiki: https://wiki.eclipse.org/Papyrus-RT
    - Forum: https://www.eclipse.org/forums/index.php/f/314/
  - Papyrus: https://eclipse.org/papyrus/
    - Papyrus industrial Consortium: https://wiki.polarsys.org/Papyrus_IC
  - PolarSys: https://www.polarsys.org/
- References

[1] Selic. What will it take? A view on adoption of model-based methods in practice. Software and Systems Modeling (SoSyM) 11(4):513-526. 2012.

[2] Whittle, Hutchinson, Rouncefield. The state of practice in model-driven engineering. IEEE Software 31 (3), 79-85. 2014.

[3] SPARC. Robotics 2020 Multi-Annual Roadmap: For Robotics in Europe, Horizon 2020 Call ICT-2017 (ICT-25, ICT-27 & ICT-28). Dec 2016.

[4] Dingel. Complexity is the Only Constant: Trends in Computing and Their Relevance to Model Driven Engineering. ICGT'16. LNCS 9761:79-85. .

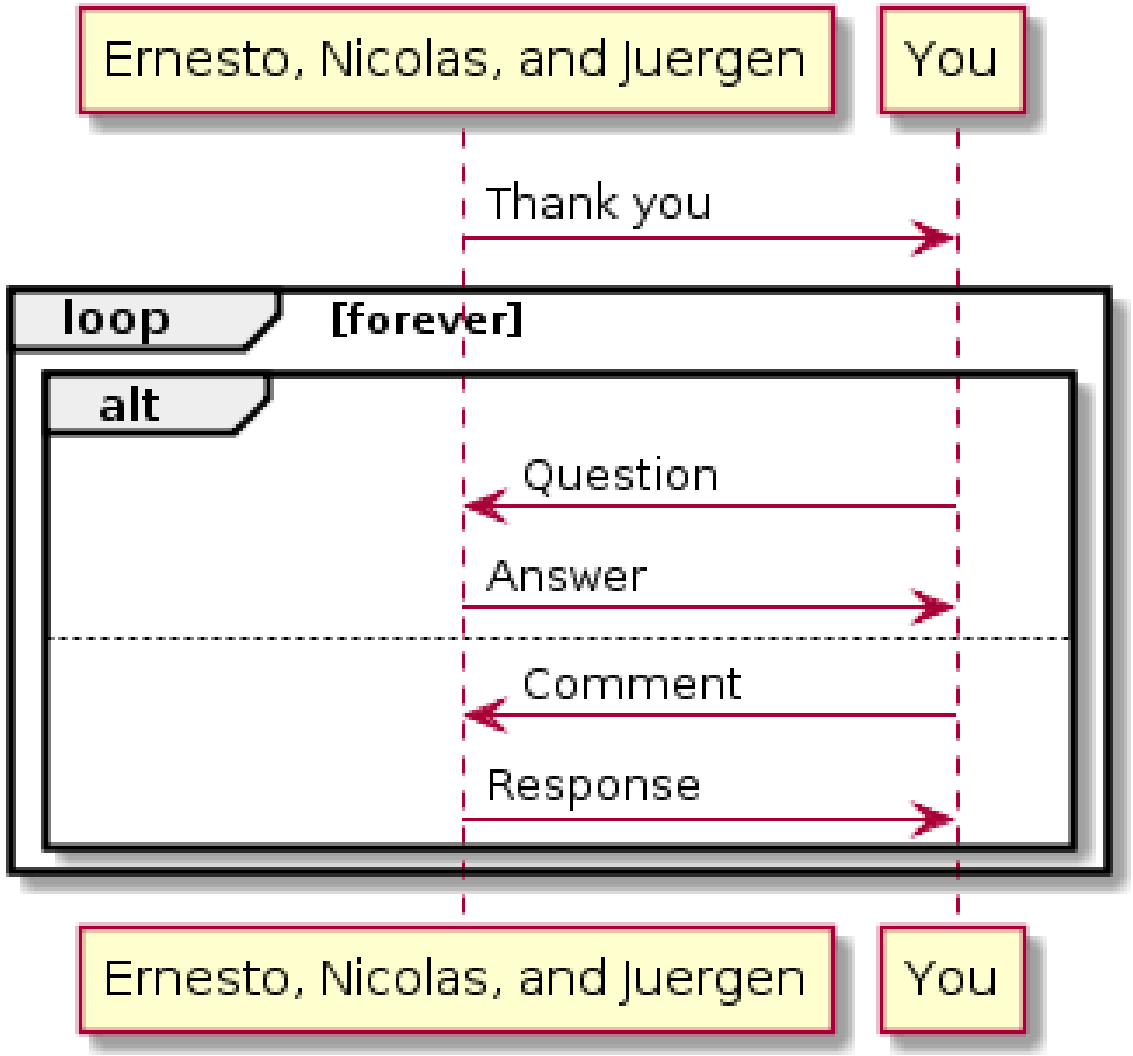
[5] Whittaker, Goldsmith, Macolini, Teitelbaum, "Model Checking UML-RT Protocols", *Proc. Workshop Formal Design Techniques for Real-Time UML*, 2000-Nov.

[6] R. Alur. Formal Analysis of Hierarchical State Machines. Verification: Theory and Practice. 2003.

[7] Selic, "Using UML for modeling complex real-time systems," in Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES'98), 1998, pp. 250–260.

[8] Posse, Dingel. An executable formal semantics for UML-RT. SoSyM 15(1):179-217. Feb 2016

The End


Ernesto, Nicolas, and Juergen
You
Thank you
loop
[forever]
alt
Question
Answer
Comment
Response
Ernesto, Nicolas, and Juergen
You