

# A Tutorial for Rational Rose Real-Time

Cheng Peng

SITE, University of Ottawa  
[cpeng@site.uottawa.ca](mailto:cpeng@site.uottawa.ca)

Sep. 10, 2003



## Acknowledgement

- The document has been reviewed by **Dr. Misbah Islam**. I, herein, appreciate his valuable comments.





## Agenda

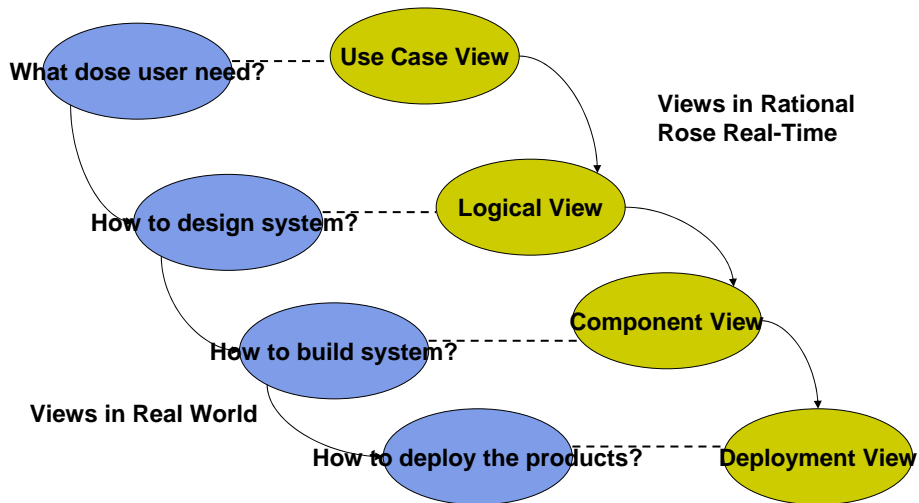
- Overview
- Rose RealTime Software Development Lifecycle
  - Use Case View
  - Logical View
    - Capsules
      - Structure Diagram
      - State Diagram
    - Classes
    - Protocols
  - Component View
  - Deployment View
- Using Rose Real-Time step-by-step
- Case Study: A Demo System TomJerry
- Where to get the document
- References



## Overview

- Rose RealTime is a software development environment tailored to the demands of real-time software.
- Developers use Rose RealTime to create models of the software system based on the Unified Modeling Language constructs, to generate the implementation code, compile, then run and debug the application.

# Rose RealTime Software Development Lifecycle



## 1. Use Case View



- The contents of this part are out of this slide.



## 2. Logical View

- **Logical View** involves various capsules, classes, and protocols to make up the design solution for the problem
  - Capsules
  - Classes
  - Protocols

**Notes:** In this slide, we only introduce capsules.



## Capsules

- Capsules are the fundamental modeling element of real-time systems, which are similar to classes.
- A capsule represents independent flows of control in a system.
- Capsules Versus Classes
  - Classes store and manage information in the system, whereas capsules provide coordinating behavior in the system.
  - Flows of events should be encapsulated in a separate capsule when it is complex and consists of dynamic behavior.
  - Capsules give transparent concurrency, easy thread assignment, state diagram generation, and message passing.



## Capsules (Cont'd)

- Two viewpoints on capsules
  - Structure Diagram
    - The structure diagram captures the interface and internal structure of the capsule in terms of its contained capsules and ports.
  - State Diagram
    - The state diagram captures the high-level behavior of the capsule.



## Structure Diagram

- Three main elements in a structure diagram
  - Capsules
  - Ports
    - *Visibility*: Public | Protected
    - *Connector* type: Wired | Non-wired
    - *Termination*: Relay | End
  - Connectors



## Classification of ports

- **Visibility**
  - Public - Public ports are ports that are part of a capsule's interface. These ports may be visible both from outside the capsule and inside, i.e., interface of a module.
  - Protected - Protected ports are used to connect capsules to contained capsule roles. These ports are not visible from the outside of a capsule since they are not part of the capsule's interface, i.e., timing port.
- **Connector type**
  - Wired - Wired ports must be connected by a connector to other ports in order to send messages.
  - Non-wired - Non-wired ports are used to model dynamic communication channels. These ports cannot be connected with connectors to other ports. For example, in client/server model, connections can be made dynamically .



## Classification of ports (Cont'd)

- **Termination**
  - Relay - Relay ports are by nature implicitly public and wired. They are used to model connections that funnel signal events directly to protected capsule components without being processed by the capsule itself.
  - End - End ports can be public or protected, wired or non-wired. Messages sent to an end port can be processed directly by the capsule's behavior (state machine).



## Connectors

- Connectors really capture the key communication relationships between capsule roles.



## State Diagram

- A state diagram shows the sequence of states that an object or an interaction goes through during its life in response to received messages, together with its responses and actions.
- State diagrams use state machines.
- A state machine is a graph of states and transitions
  - States
  - Transitions



## States

- A state has the following parts:
  - Name
  - Entry/Exit actions - Actions that are executed on entering and exiting the state.
- A state may contain substates and may be nested to any level.



## Transitions

- A transition has the following parts:
  - **Trigger** - With the exception of the initial transition all behavior in a state machine is triggered by the arrival of events on one of an object's interfaces. Therefore, a trigger defines which events from which interfaces will cause the transition to be taken.
  - **Guard Condition** - Each trigger can have a boolean expression associated with it which will be evaluated before the transition is triggered. This is referred to as a guard condition.
  - **Actions** - The actions in a behavior are where an object does work.



### 3. Component View

- A component diagram shows the dependencies among software components.
- Some components exist at compile time, some exist at link time, some exist at run time, and some exist at more than one time. The run-time component in this case would be an executable program.
- A component diagram has only a type form, not an instance form.



### Deployment View

- Component instances may be contained in nodes, which indicates that the component runs on the node.
- The deployment diagram provides a basis for understanding the physical distribution of the run-time processes across a set of processing nodes.

## Using Rose Real-Time step-by-step



- In this section, we just emphasize some key points you might miss.
- For detail tutorial, please refer to ***Rational Rose Real-Time Help***, where two examples are shown in the ***Tutorials*** section.
- In this section, we use Visual C++ 6.0 as our programming language. Actually, Java can also be used.

## Before you start



- You must have Microsoft Visual C++ 6.0 installed on your system and configured to be run from the DOS prompt to make use of the code generation and execution capabilities of Rose RealTime.
- The following instructions help you to determine whether you have Visual C++ properly installed and configured on your system.
  - From the Windows **Start** menu:
    - In Windows NT, choose **Start > Programs > Command Prompt**
    - In Windows 2000 and Windows XP, choose **Start > Programs > Accessories > Command Prompt**
  - Type the following commands to test your environment
    - Type ***nmake*** and press ENTER.
    - Type ***cl*** and press ENTER.



## Testing your environment

- If your environment is correct, then you should see as follows:

**Command Prompt**

Microsoft © Windows NT TM  
© Copyright 1985-1996 Microsoft Corp.  
C:\>nmake

Microsoft © Program Maintenance Utility Version 6.00.8168.0  
Copyright © Microsoft Corp 1988-1998. All rights reserved.

**NMAKE** = fatal error B1864: MAKEFILE not found and no target specified  
Stop.

C:\>cl

Microsoft © 32-bit C/C++ Optimizing Compiler Version 12.00.8168 for 80x86  
Copyright © Microsoft Corp 1984-1998. All rights reserved.

Usage = cl { option... } filename... { /link linkoption... }

- If your environment is NOT properly configured, then you will see an error similar to this one:

**Command Prompt**

C:\> nmake

The name specified is not recognized as an internal or external command, operable program or batch file.



## Viewing the Generated Code

- To examine the generated code, select the folder where you saved your model.
- In this folder, open the (the name of your created component)\src folder.
- The src folder contains the generated code for your model

## Case Study: A Demo System TomJerry



- **Tom** is a cat and **Jerry** is a mouse. They meet one day on Internet. **Tom** wants to know whether **Jerry** is a cat or not by asking him some questions.

## Case Study: A Demo System TomJerry (Cont'd)



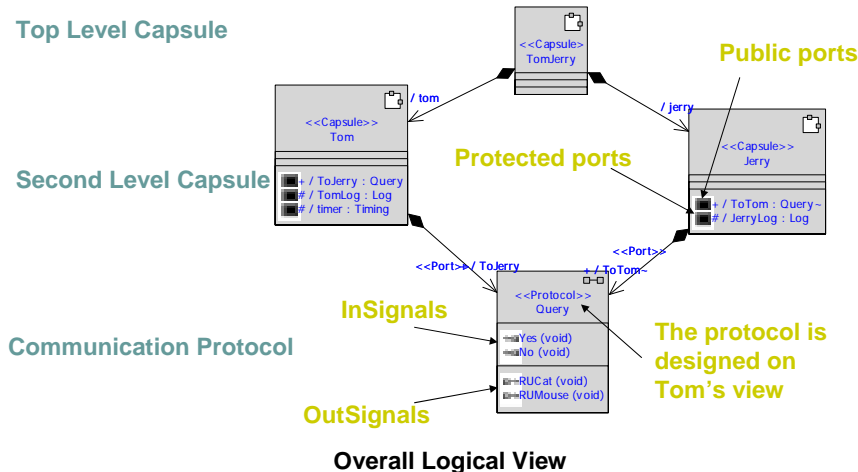
- The system behaviours are like this.
  - Initially, **Tom** sends message **RUCat** to **Jerry**
  - **Jerry** replies with message **No** to **Tom**, having received **RUCat**.
  - **Tom** then sends message **RUMouse** to **Jerry** on receiving **No**.
  - **Jerry** replies with message **Yes** to **Tom** on receiving **RUMouse**.
  - Then Tom knows whether or not Jerry is a cat.

## Case Study: A Demo System TomJerry (Cont'd)

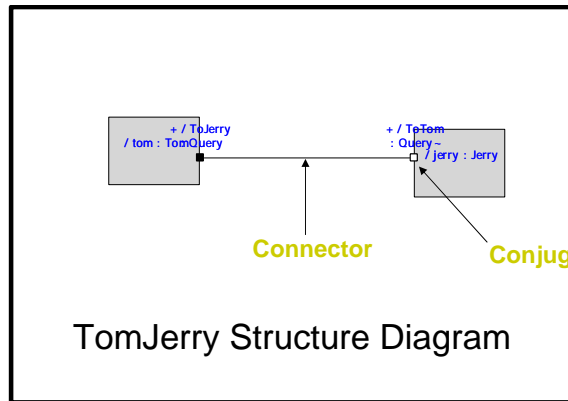


- We consider the simple communication system consisting two entities.
  - Entity 1 is named as **Tom**.
  - Entity 2 is named as **Jerry**.
- The signals used in the common communication protocol between Tom and Jerry are:
  - Tom->Jerry: RUCat and RUMouse
  - Jerry->Tom: Yes and No

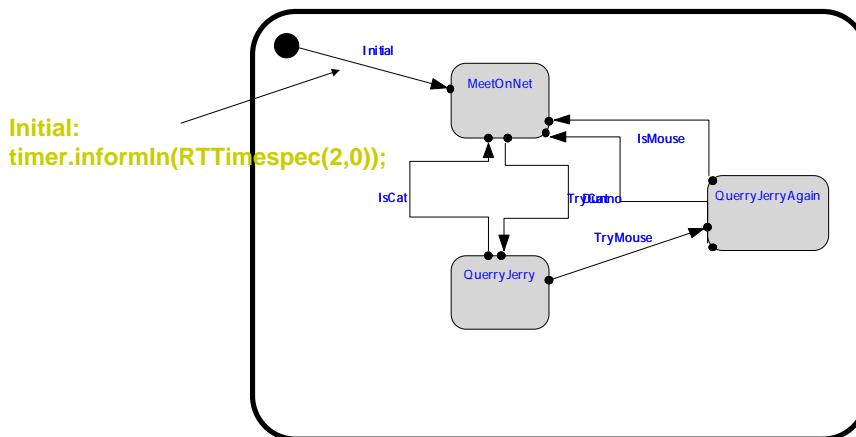
## A Demo System TomJerry Overall Logical View



# A Demo System TomJerry TomJerry Structure Diagram

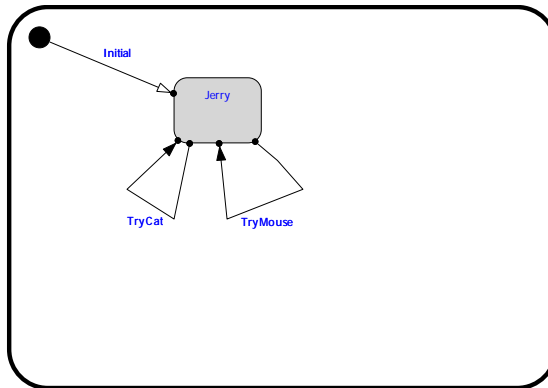


# A Demo System TomJerry Tom's State Diagram



# A Demo System TomJerry

## Jerry's State Diagram



Jerry's State Diagram

# A Demo System TomJerry

## Executable Results



```
D:\Cheng\TomJerry\Model\TomJerryComponent\build\TomJerry.ttl
Rational Base RealTime C++ Target Run Time System
Release 6.4H.C.00 (c++)
Copyright (c) 1993-2003 Rational Software
insert: observability listening at tcp port 30096

=====
Please note: ETDM is turned off.
- To use the command line, select to the above mentioned part. -
- The output of any command will be displayed in this window. -
=====

Jerry is a mouse? :>>>
-
```

Running the Model



## Where to get the document

- You can visit my website to get this document and TomJerryModel source code.
- Visit <http://www.site.uottawa.ca/~cpeng/> and go to TAs button on the left side of the menu.



## References

- Rational Rose RealTime Help (version 6.0.63.0)



**Thank You!**