Modeling Behavioral Design Patterns of Concurrent Objects

Hassan Gomaa

Dept. of Computer Science
George Mason University
Fairfax, Virginia,
USA

hgomaa@gmu.edu

Joint research conducted with Dr. Robert Pettit

Overview

- Goals
  - Provide executable behavioral analysis capabilities
    - For concurrent object-oriented software architectures
    - At **design** stage
- Concurrent software architectures are depicted in UML
- Colored Petri nets (CPNs) used as underlying formalism
- CPN templates created to model executable behavioral design patterns
  - Promotes systematic, repeatable model construction

Modeling Executable Software Architectures

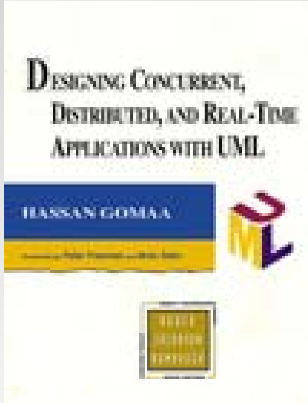
- Design and analyze concurrent software architecture
- Behavioral design patterns
  - Concurrent component
  - Connector
  - Mapped to Colored Petri Net template
- Map concurrent software architecture to CPN model
  - Select and interconnect CPN templates for components and connectors
- Analyze executable CPN model
  - Application behavior
  - Application performance
- *R. Pettit and H. Gomaa, "Modeling Behavioral Design Patterns of Concurrent Objects", Proc. Int. Conf. on Software Eng. (ICSE), Shanghai, May 2006.*

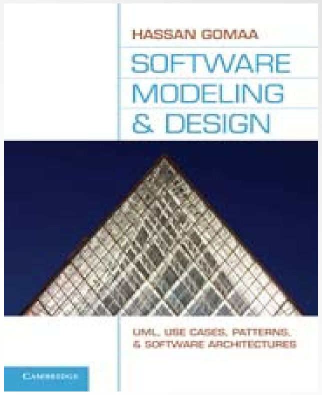
Using Behavioral Design Patterns

- Start with software design captured in UML
  - Depicted on UML 2 communication diagrams
- Structure concurrent system into concurrent objects
  - Categorize concurrent objects by behavioral role
  - Each concurrent object is represented by behavioral design pattern
  - Mapped to CPN template

Software Modeling and Design for Concurrent Systems

- COMET design method
  - From Use Case Models to Software Architecture
  - COMET = method + UML
  - Requirements and Analysis Modeling
    - Use case modeling
    - Static and Dynamic modeling
  - Design modeling
    - Concurrent, distributed, and real-time applications
  - *H. Gomaa, “Designing Concurrent, Distributed, and Real-Time Applications with UML”, Addison Wesley Object Technology Series, 2000*
  - *H. Gomaa, Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures*, Cambridge University Press, February 2011

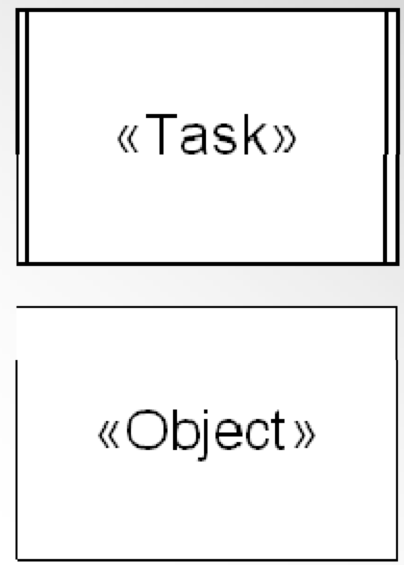




Concurrent and Passive Objects

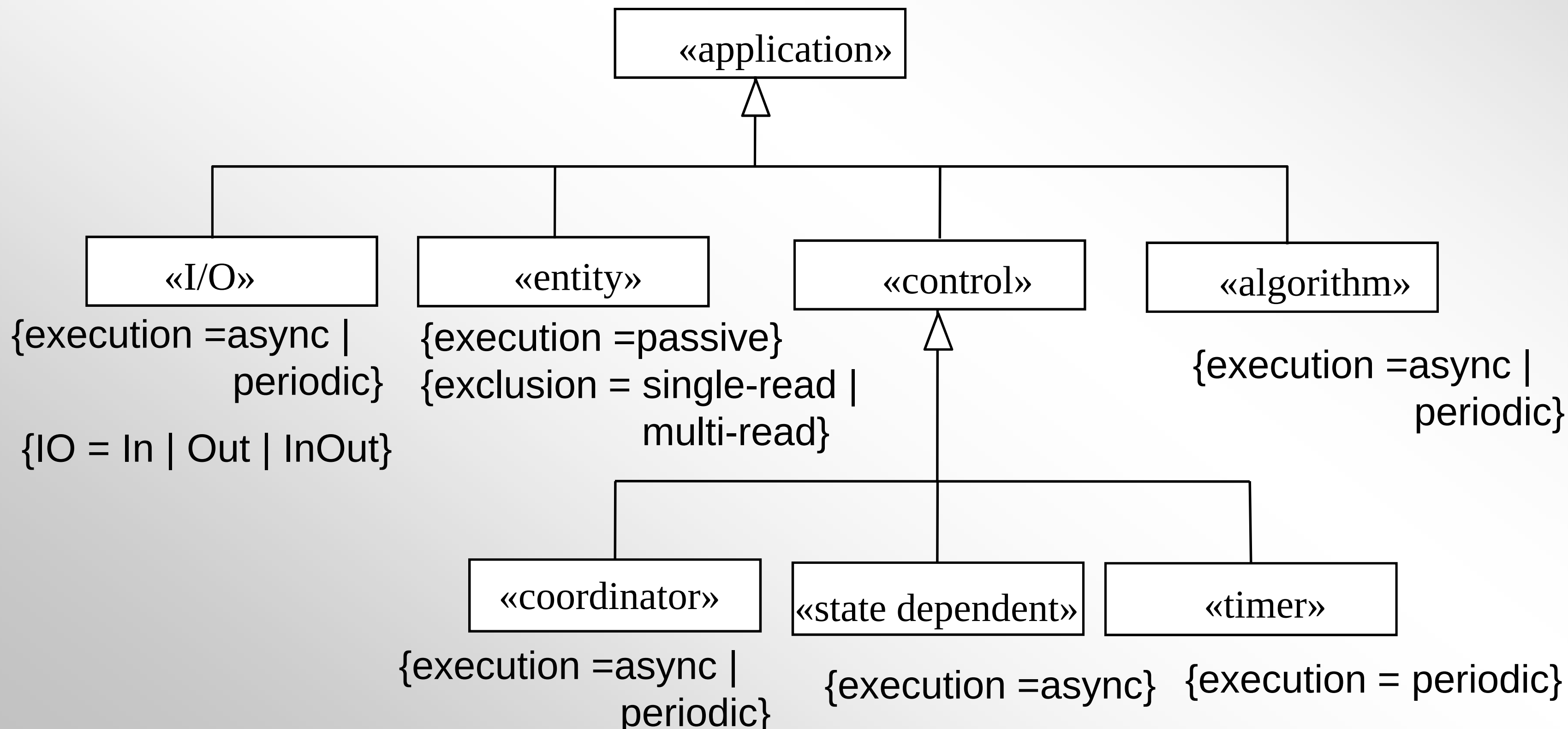
- Concurrent system consists of **concurrent objects** and **passive objects**
- **Concurrent object**
  - Has a **thread** of control
  - Executes autonomously
  - Also known as
    - **Active object**
    - **Concurrent process (lightweight)**
    - **Concurrent task**
    - **Concurrent component**
    - **Thread (Java)**
    - **Processor (Scoop)**
- **Passive object**
  - Has no thread of control
  - Also known as
    - Sequential object
    - Object

UML notation



Structure and Categorize Concurrent Objects

- Use COMET structuring criteria to categorize concurrent objects
  - Each concurrent object depicted using UML stereotype
  - Specify architectural parameters for each concurrent object
  - Identify concurrent object behavioral design pattern

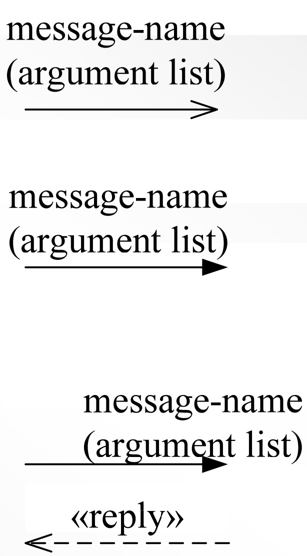


UML Notation for Messages

a) Asynchronous message communication

b) Synchronous message communication

c) Synchronous message communication with reply



Asynchronous I/O Concurrent Object (Component/task/thread)

One concurrent object for each asynchronous I/O device

Activated by device interrupt

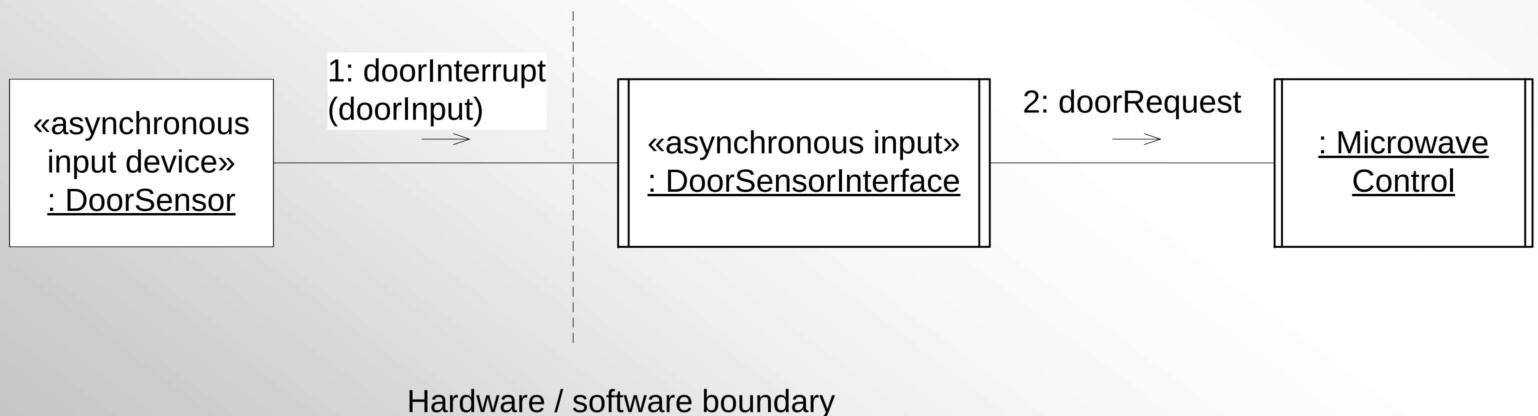
Reads input

Converts to internal format

Sends message containing data

Waits for next interrupt

Figure 14.1b Design model – UML concurrent communication diagram

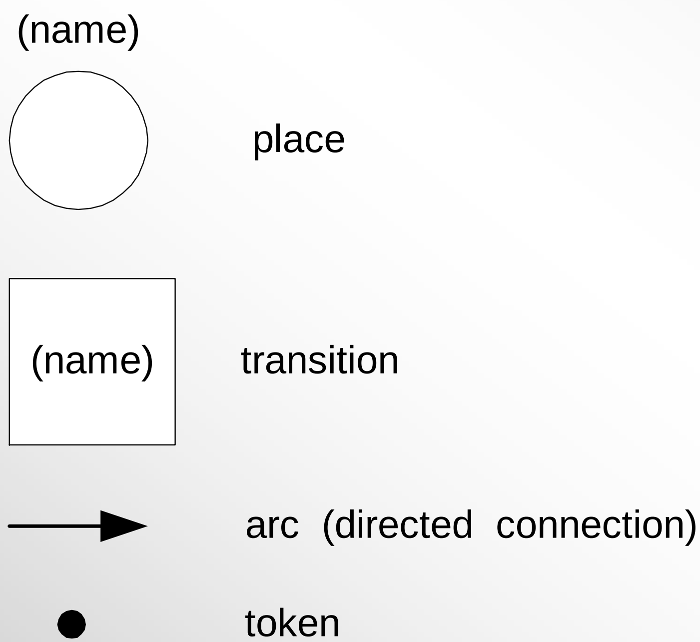


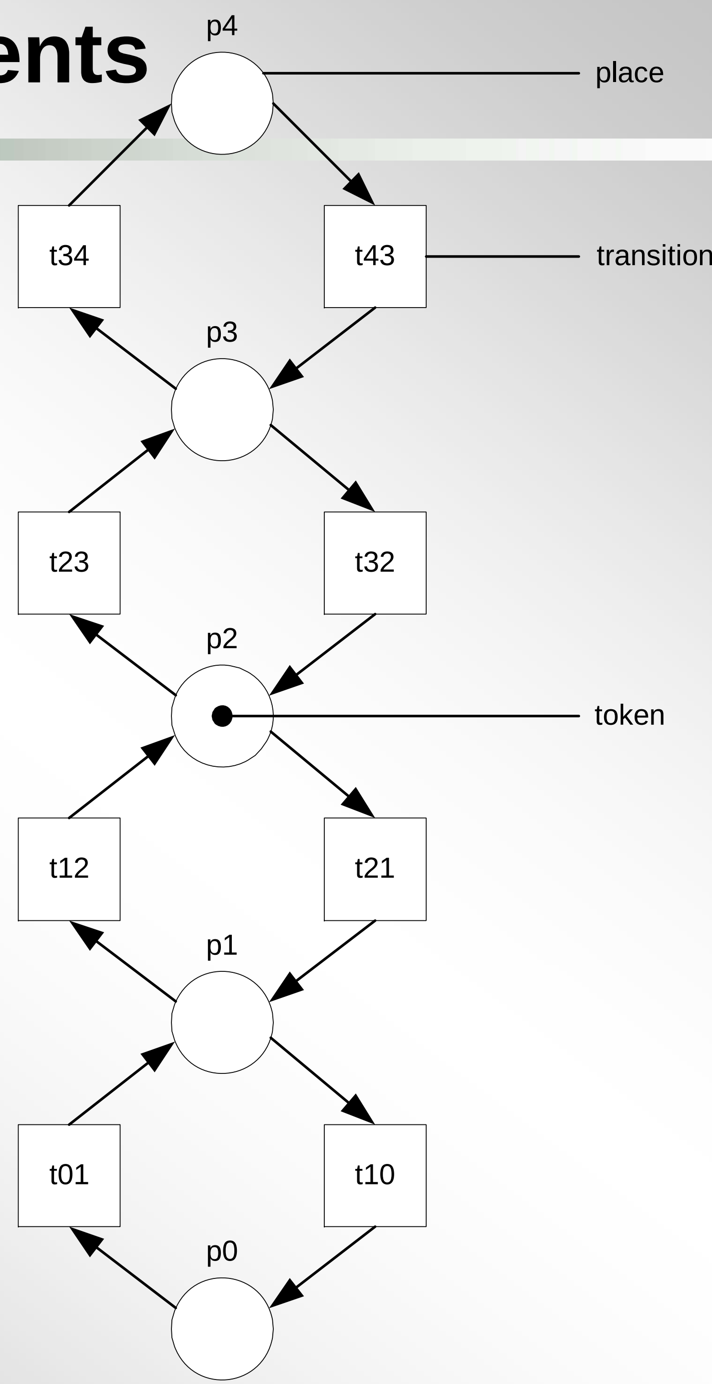
Classical Petri Nets

- Simple concurrency model
  - Just three elements: **places**, **transitions** and **arcs**.
  - Graphical and mathematical description.
  - Formal semantics and allows for analysis.
- History:
  - Carl Adam Petri (1962, PhD thesis)
  - In sixties and seventies focus mainly on theory.
  - Since eighties also focus on tools and applications (cf. Colored Petri Net work by Kurt Jensen).

- Source: Intro to Petri Nets, Wil van der Aalst

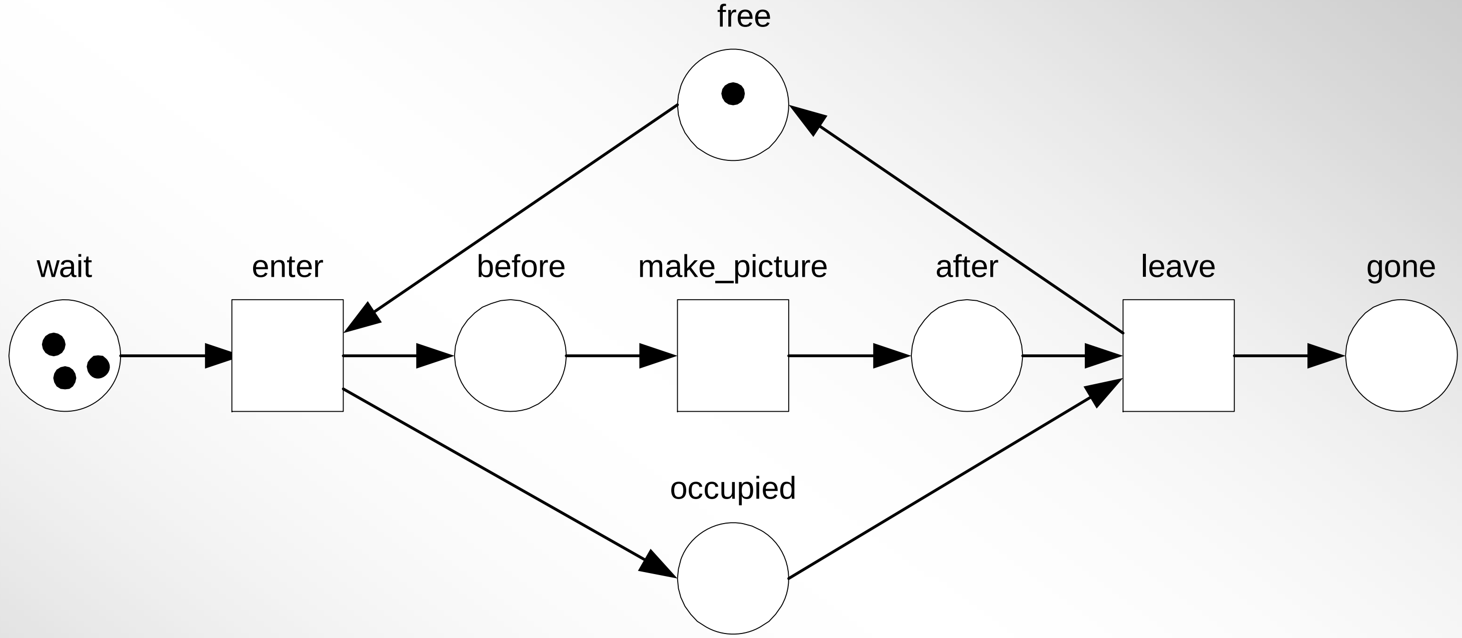
Petri Net Elements





Source: Intro to Petri Nets,
Wil van der Aalst

Petri Net Rules

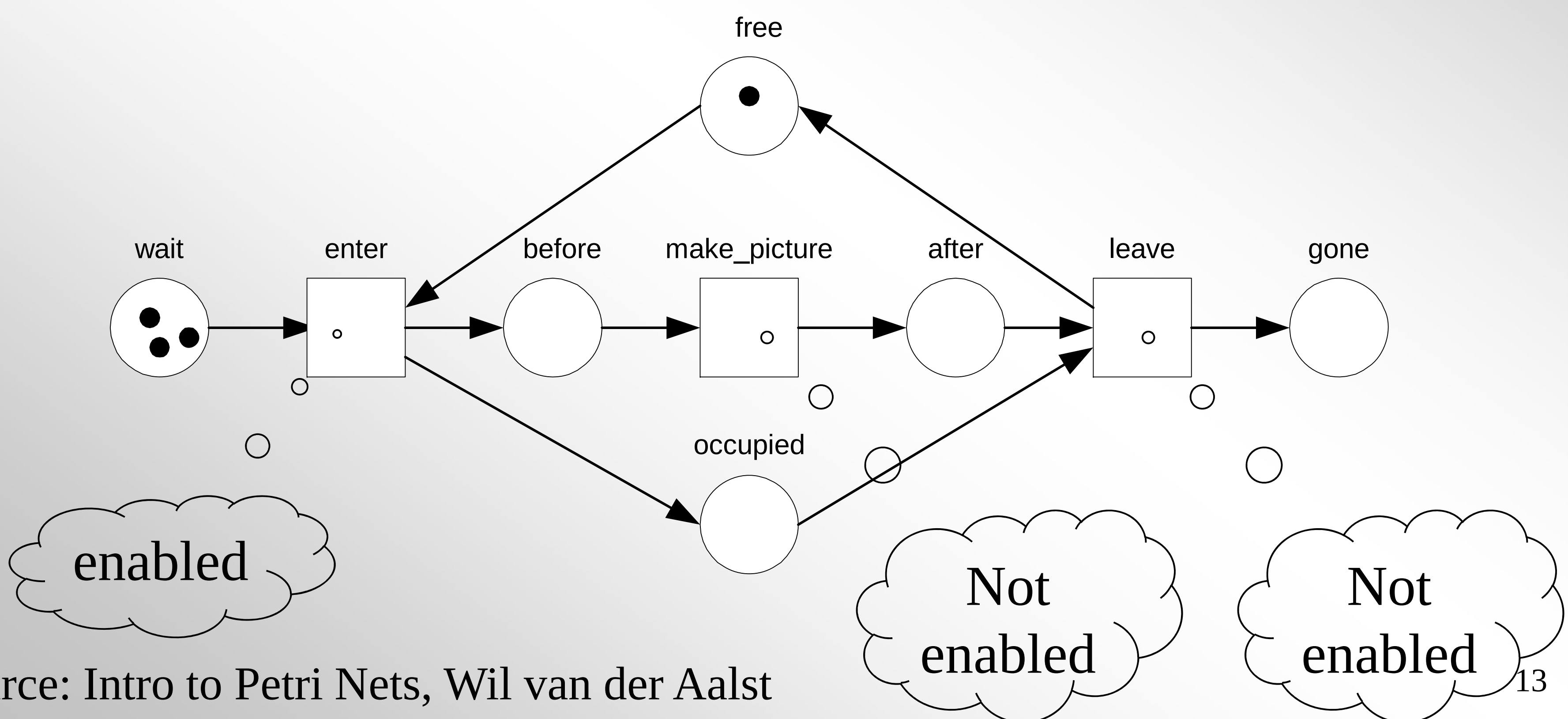


- Connections are directed.
- No connections between two places or two transitions.
- Places may hold zero or more tokens.

Source: Intro to Petri Nets, Wil van der Aalst

Enabled Transition

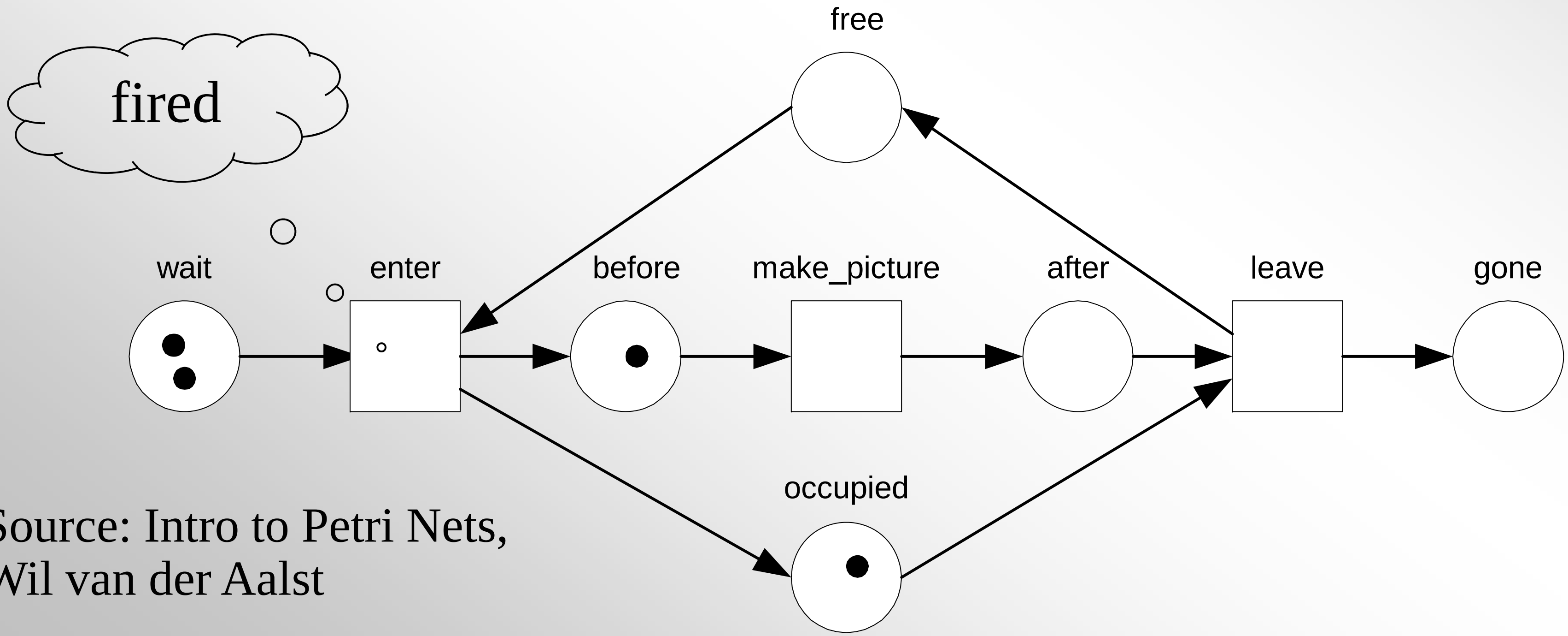
- A transition is **enabled** if each of its input places contains at least one token



Source: Intro to Petri Nets, Wil van der Aalst

Firing of Transition

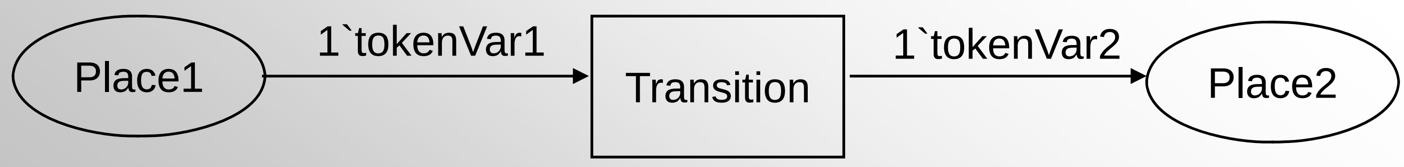
- An **enabled** transition can **fire** (i.e., it occurs).
- When it **fires** it **consumes** a token from each input place and **produces** a token for each output place.



Source: Intro to Petri Nets, Wil van der Aalst

Colored Petri Nets (CPN)

- Developed by Kurt Jensen
- Petri nets extended with:
  - Color
    - Tokens given data value
  - Time
    - Enabled transition fires after specified time
  - Hierarchy
    - Transition can be decomposed to lower level CPN subnet
- Tool support
  - Design CPN



Concurrent Software Architecture

- Uses component / connector paradigm
- Component
  - Concurrent object with single thread of control
  - Passive entity object
    - Encapsulates data
- Connector
  - Provides message communication between concurrent objects
- Model components and connectors using Colored Petri Net templates

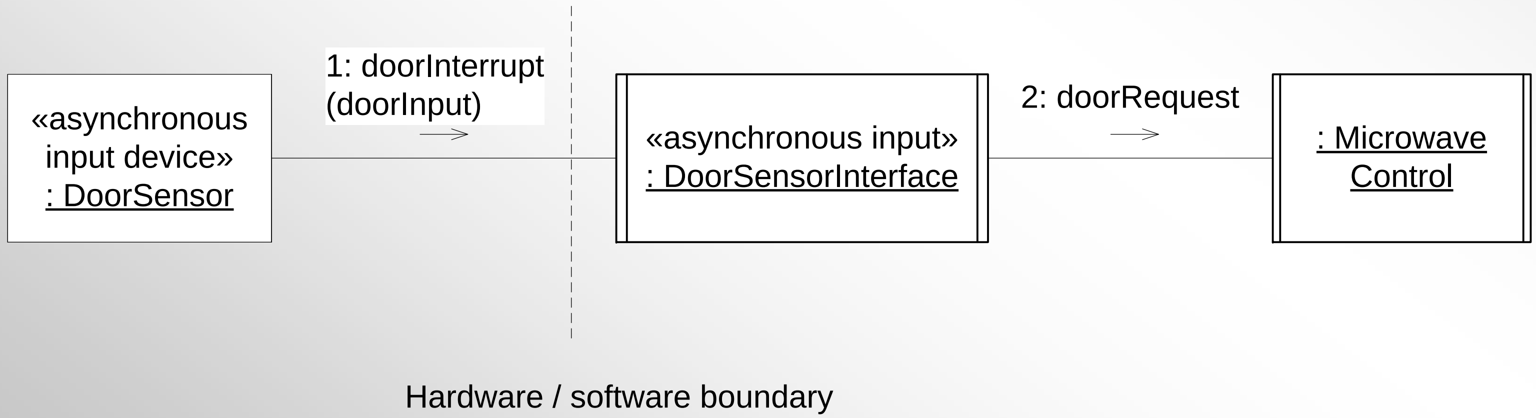
Mapping Concurrent Software Architecture to Colored Petri Nets

- CPN behavioral design template designed for each
  - Component
  - Connector
- Colored Petri Net notation
  - Transition executes function when fired
    - Consumes colored tokens from input places
    - Produces colored tokens on output places
    - Transitions can have timing parameters

Asynchronous I/O Component

- One concurrent component for each asynchronous I/O device
- Activated by device I/O interrupt
- Reads input
- Converts to internal format
- Sends message containing data
- Waits for next interrupt

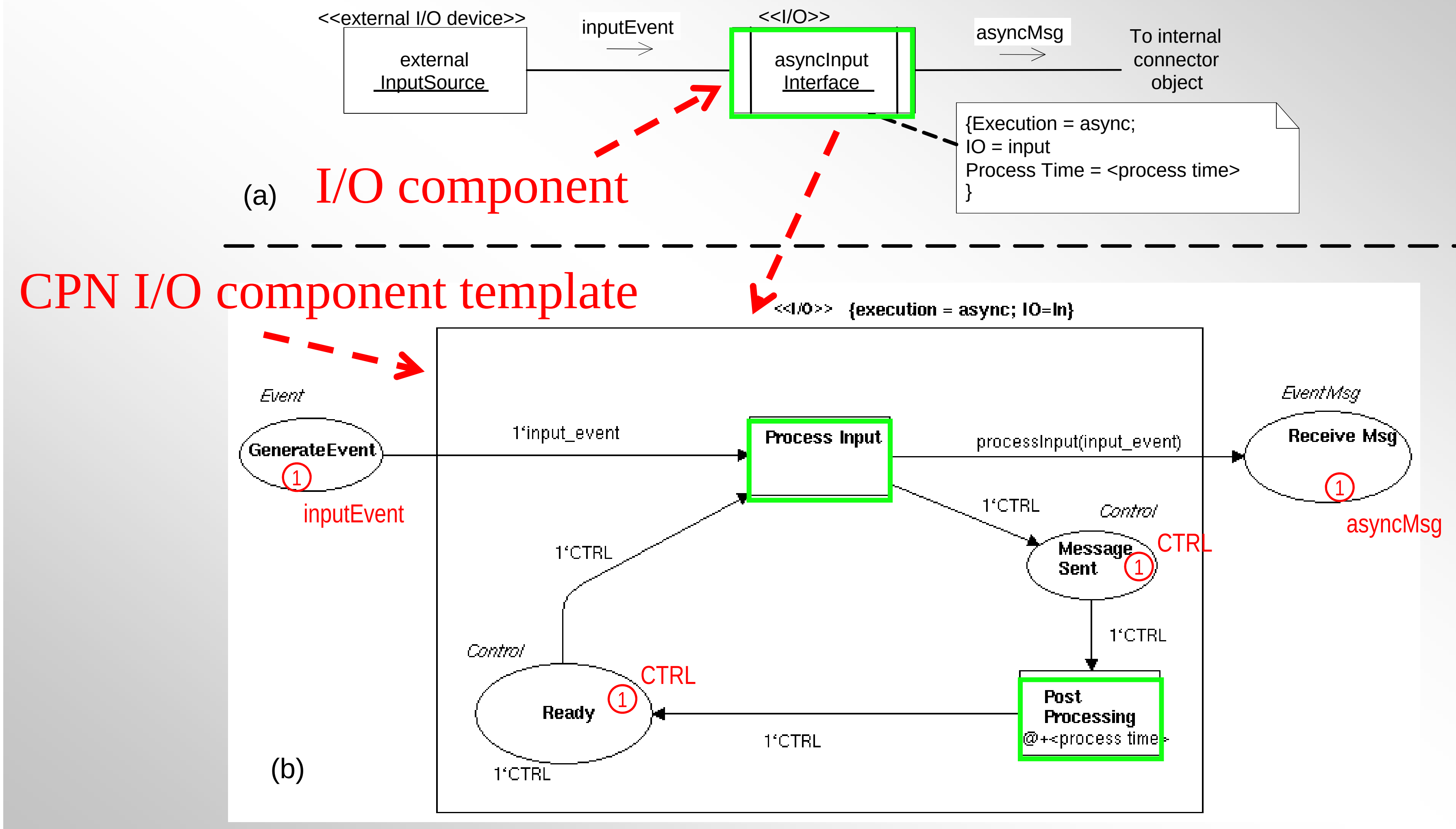
Figure 14.1b Design model – concurrent communication diagram



Asynchronous I/O Pattern

- I/O component
  - Handles external input/output on demand
- CPN pattern
  - Thread of control maintained by control token
  - Each component has its own control token
- CPN Transition executes function
  - Processing time associated with transition
- Colored tokens to differentiate role of tokens
  - Control token
  - Input event
  - Output message

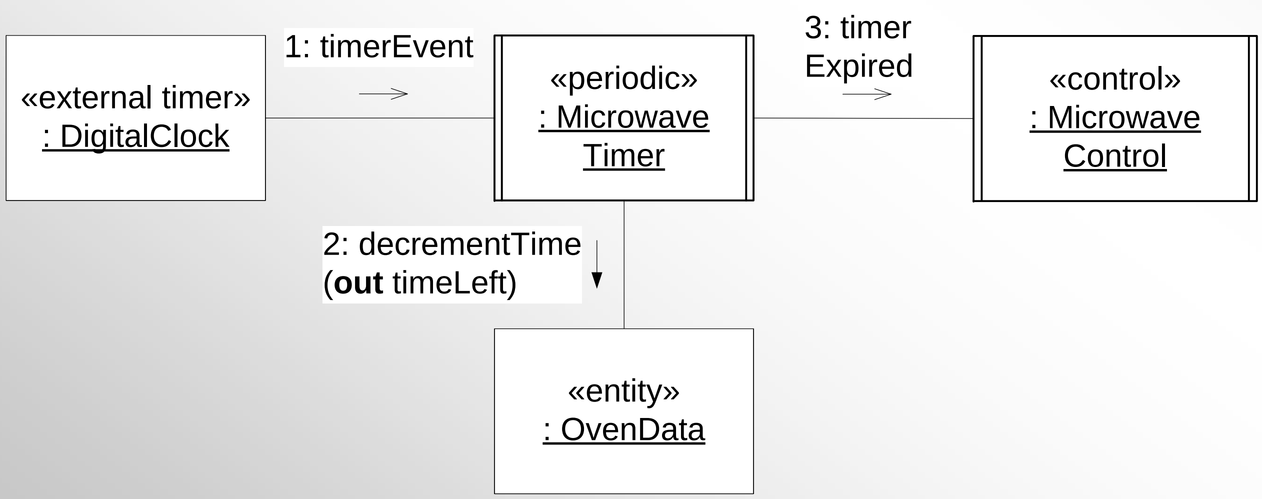
Asynchronous I/O Pattern



Periodic Algorithm Component

- Component for each periodic algorithm
- Component activated periodically
  - Activated by timer event
  - Executes algorithm
  - Waits for next timer event

Figure 14.5b Design model – concurrent communication diagram

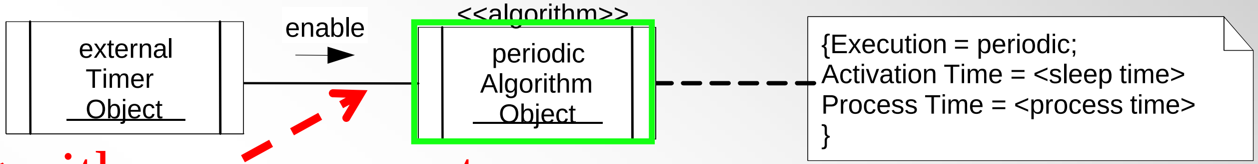


Periodic Algorithm Pattern

- Algorithm component
  - Encapsulate application logic
    - Modeled by transition
  - Execute asynchronously or periodically
- Periodic behavior modeled by
  - Sleep – Wakeup – Ready – Timeout cycle

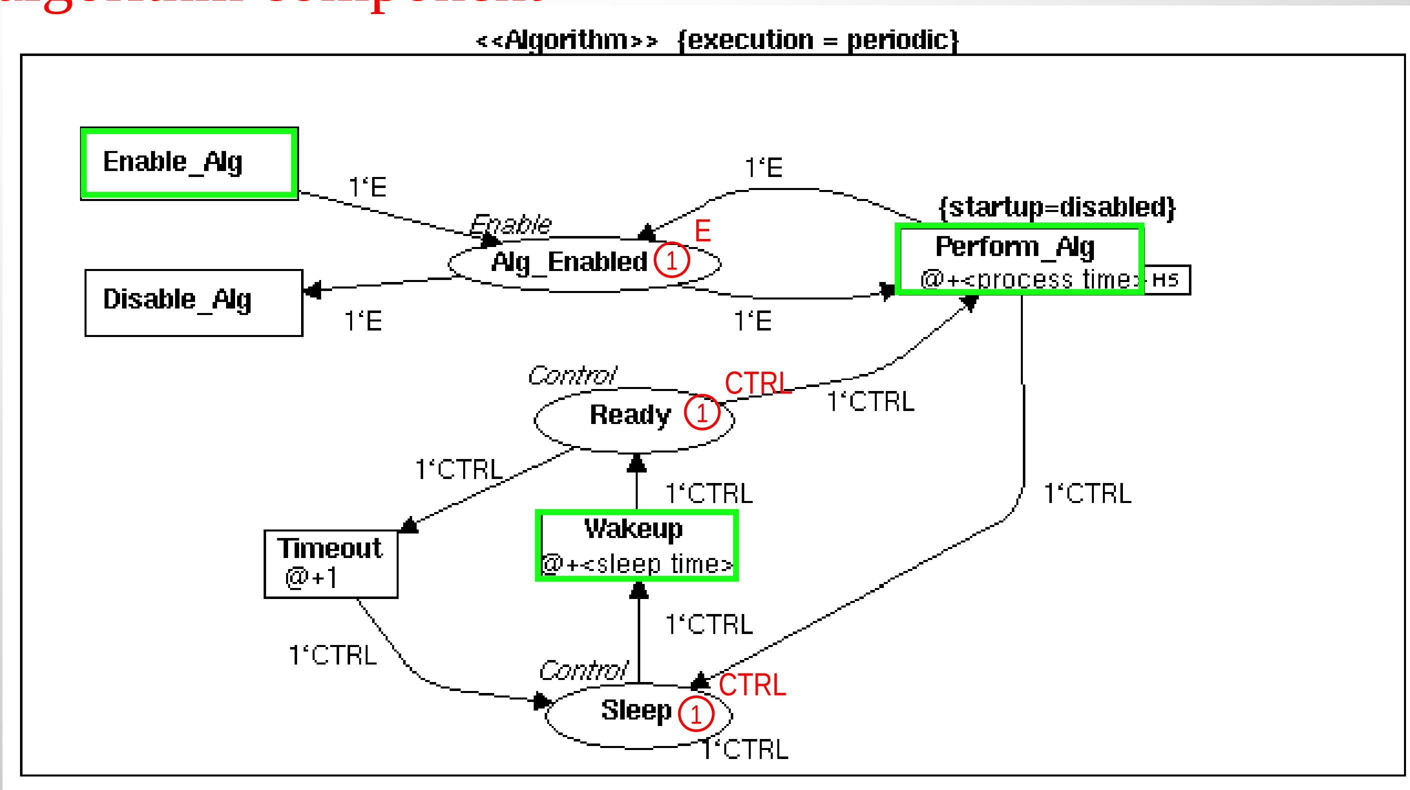
Periodic Algorithm Pattern

(a)



algorithm component

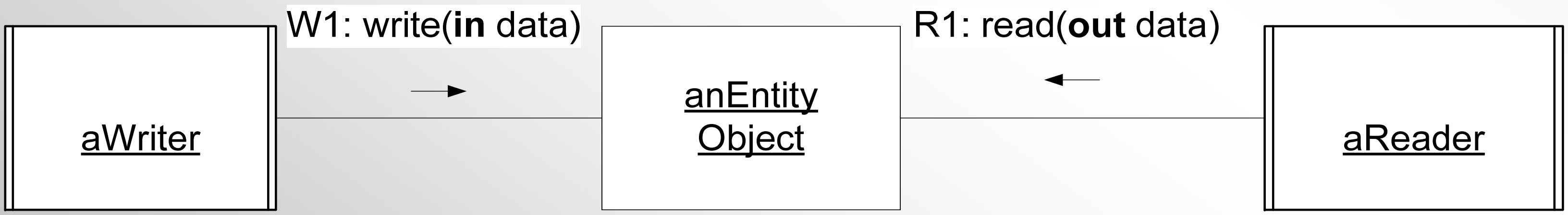
(b)



CPN algorithm template

Entity Object

- Entity object is a passive object
  - Encapsulates data
  - Hides contents of data structure
  - Data accessed indirectly via operations
- Passive object accessed by two or more components
  - Operations must synchronize access to data
    - E.g., by mutual exclusion
  - Use semaphore or monitor object

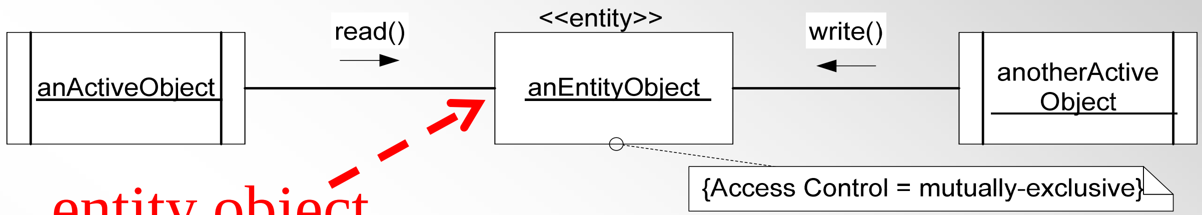


Entity Object Pattern

- Entity objects are passive
  - Encapsulate data
  - No thread of control
    - > No control token
  - Interfaces are through places rather than transitions
    - Facilitates connection to concurrent objects
  - Interfaces represent access operations
    - Operation behavior modeled with transition
    - Execution uses caller's control token

Entity Object Pattern

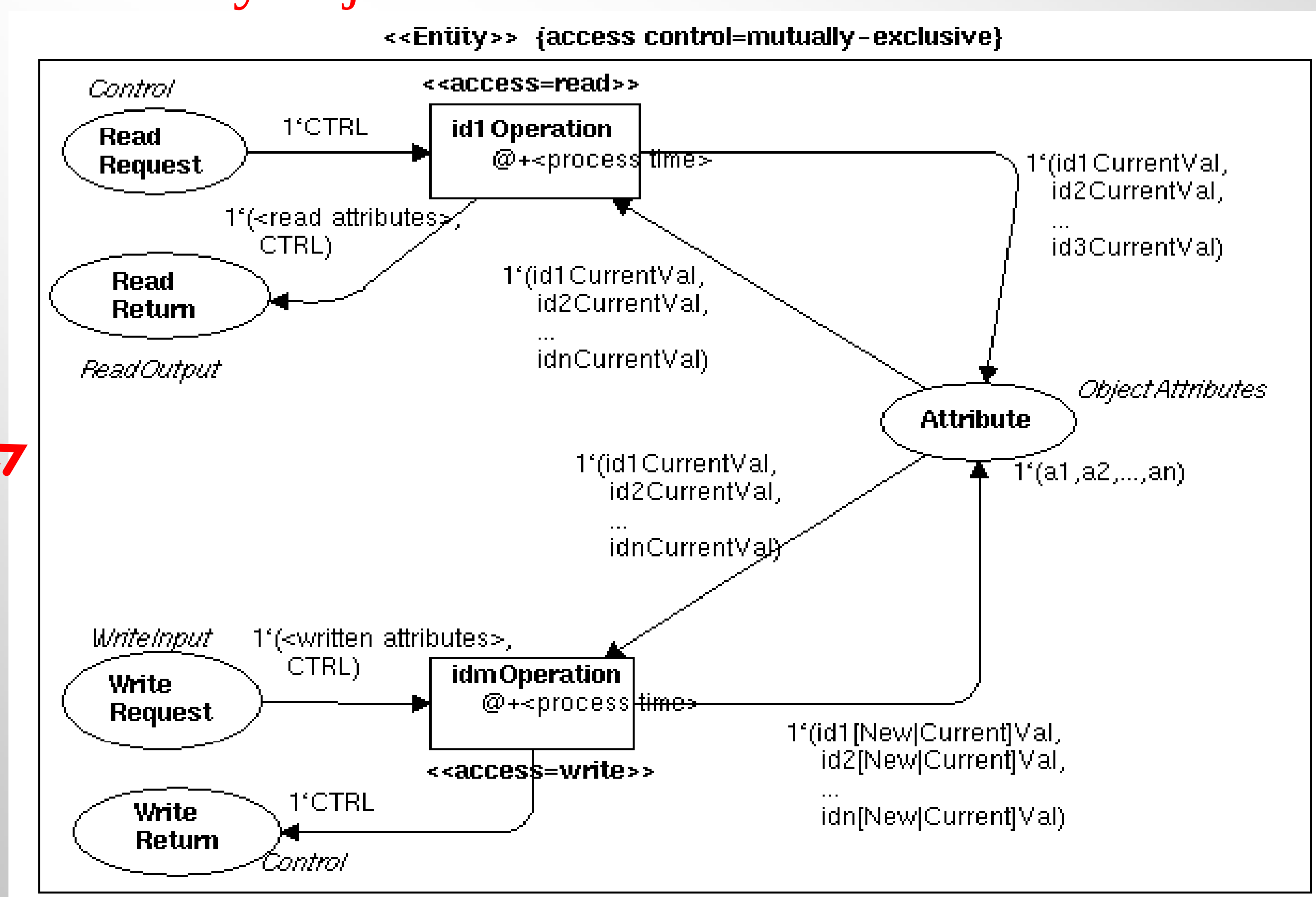
(a)



entity object

CPN entity template

(b)

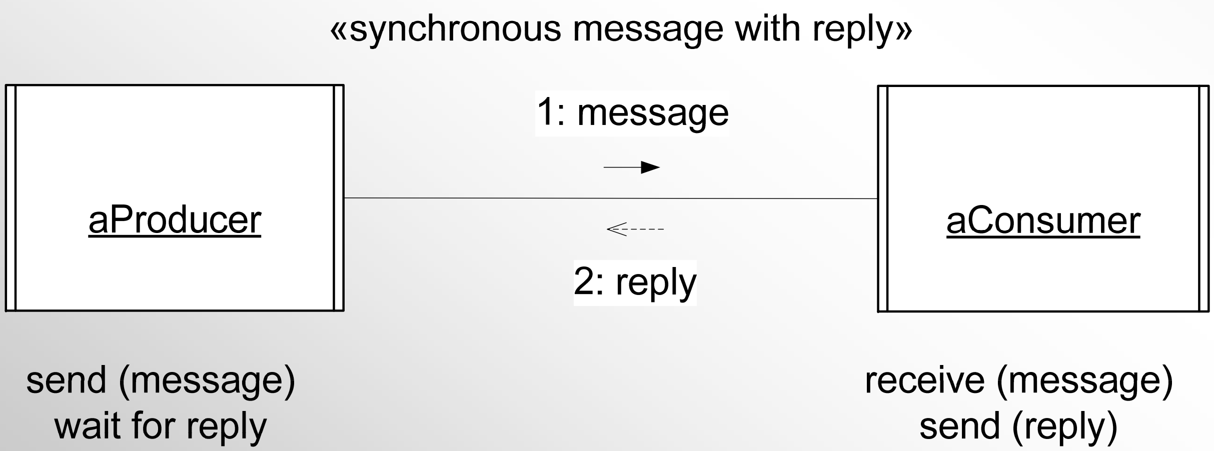


Connectors

- Connector
  – Provides message communication between concurrent components
    - Queue - Asynchronous communication
    - Buffer - Synchronous communication
- Interface to connector uses CPN places
  – Facilitates interconnection between concurrent component templates and connector templates

Synchronous Message Communication With Reply

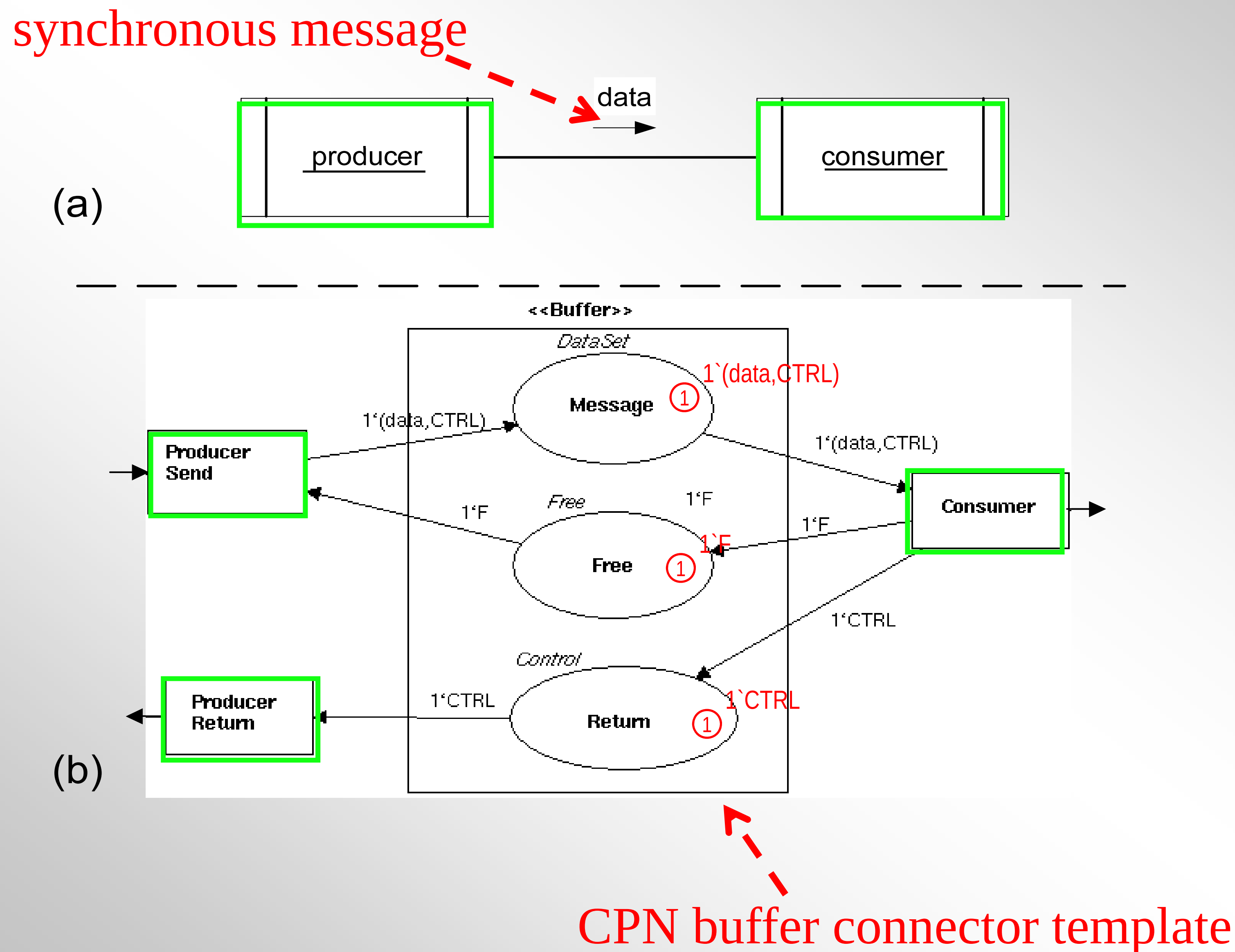
- Producer sends message and waits for reply
- Consumer receives message
  - Suspended if no message is present
  - Activated when message arrives
  - Generates and sends reply
- Producer and Consumer continue



Synchronous Communication Pattern

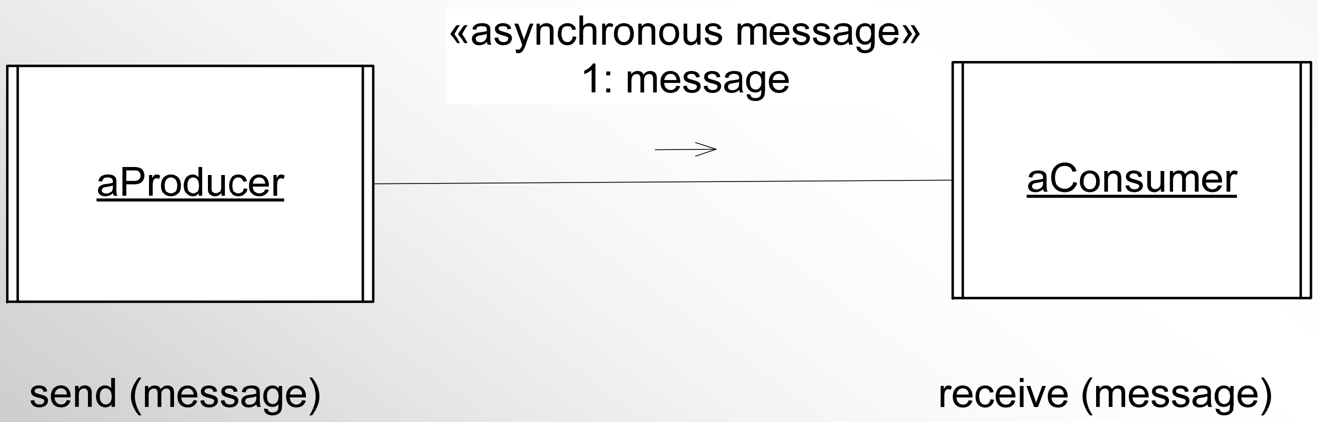
- Synchronous buffer models synchronous communication
- Producer sends message and waits for reply
- One message at a time allowed in the buffer
- Producer and consumer are blocked until message has been passed

Synchronous Communication Pattern



Asynchronous Message Communication

- Producer sends message and continues
- Consumer receives message
  - Suspended if no message is present
  - Activated when message arrives
- Message queue may build up at Consumer



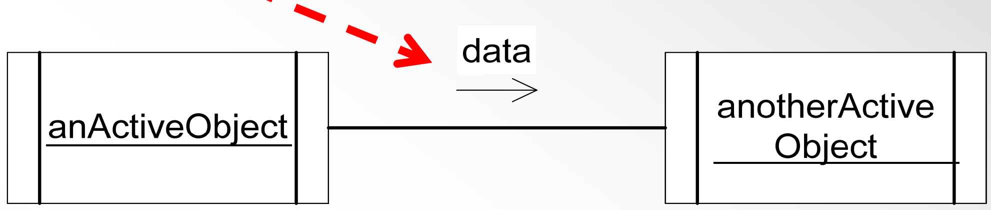
Asynchronous Message Communication (Queue) Pattern

- Asynchronous communication
  – Modeled using FIFO message queue
- Producer is not blocked during the communication
- Consumer is only blocked if no messages in queue

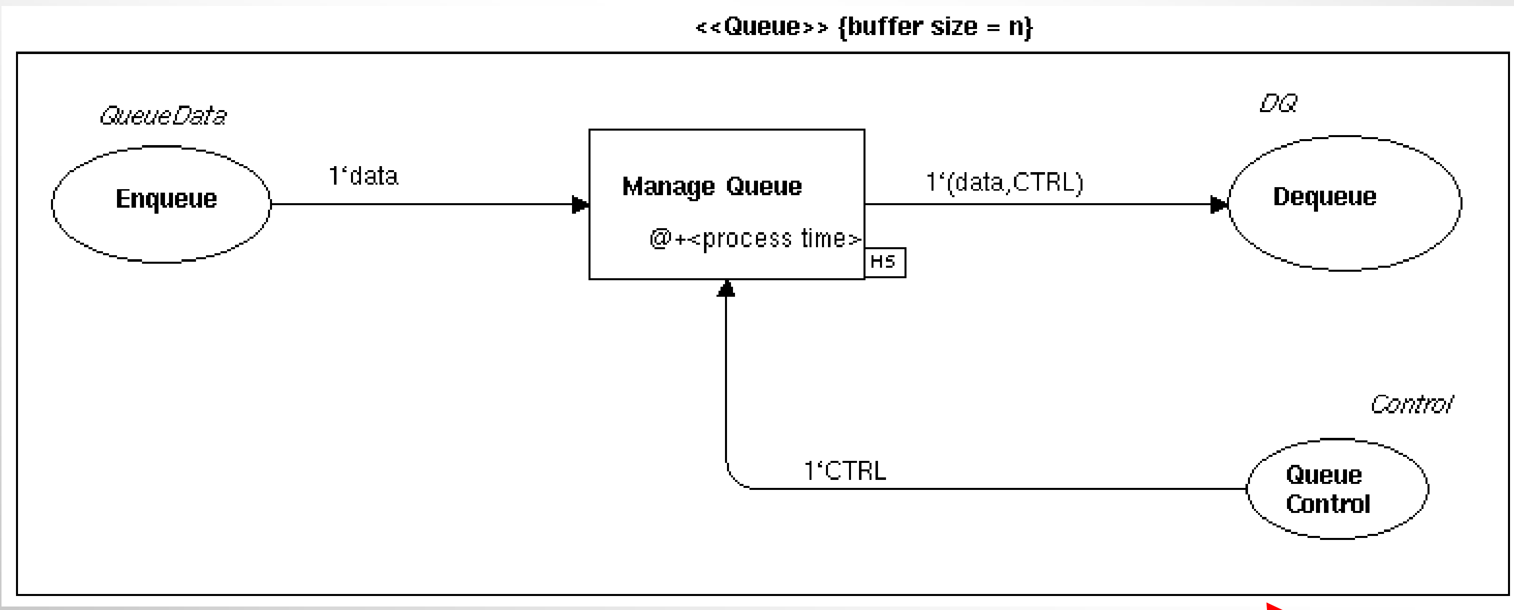
Asynchronous Message Communication (Queue) Pattern

asynchronous message

(a)



(b)

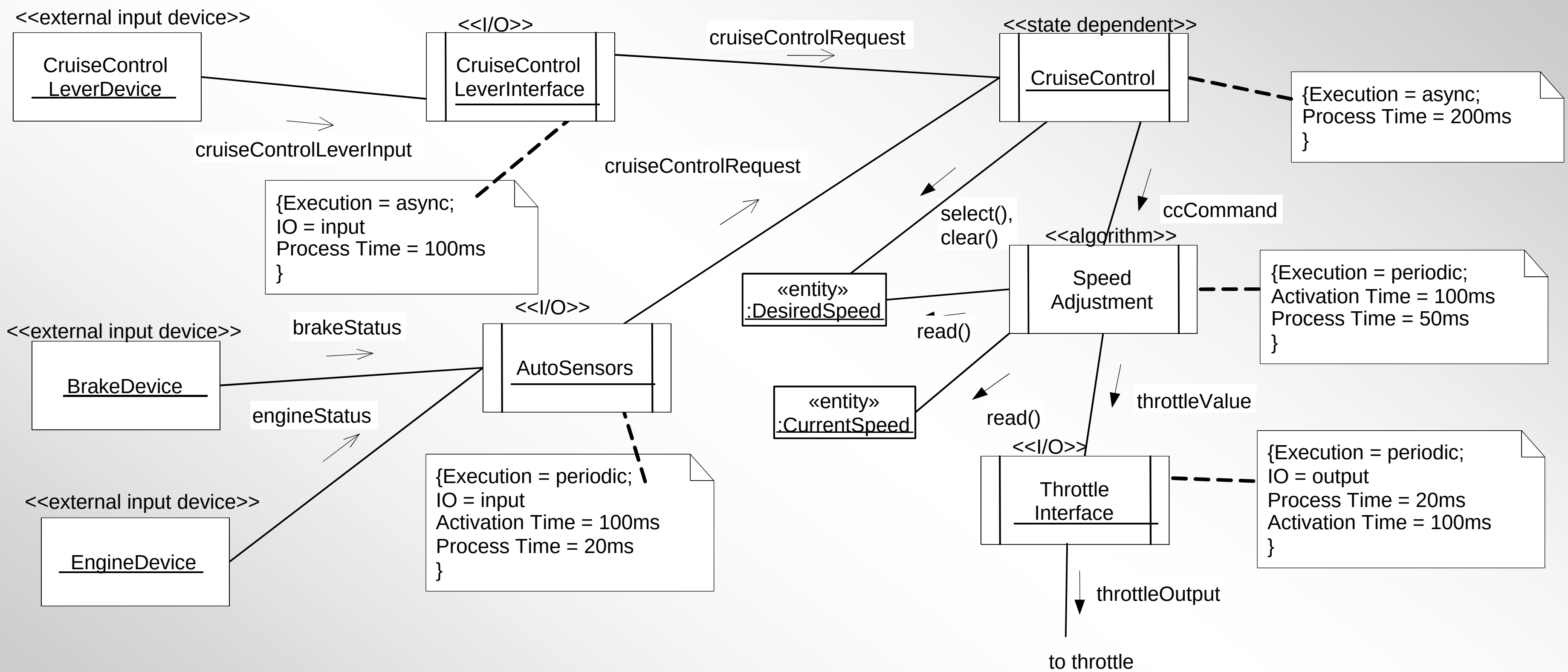


CPN queue template

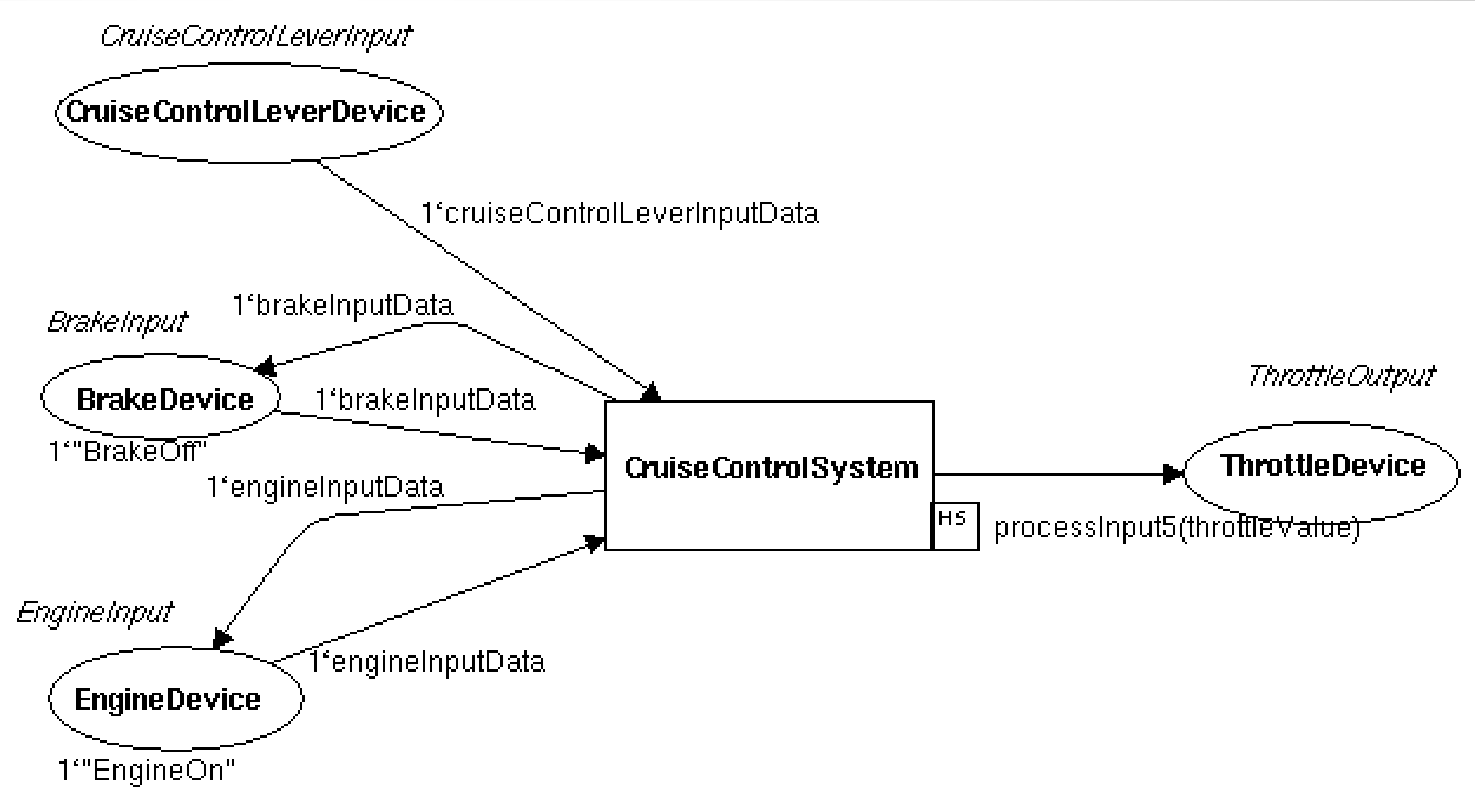
Constructing CPN Model from Concurrent Design Model

1. Develop COMET design model
   – COMET structuring criteria
2. Construct Architecture-Level CPN Model
   – Represent each component & connector by CPN template
   – Templates developed using DesignCPN
   – Interconnect CPN templates
3. Model characteristics of individual component
   – Customize CPN templates for application
4. Exercise model in DesignCPN simulator
   – Analyze functional behavior
     • Detect and correct design problems
   – Analyze performance characteristics
     • Does software architecture meets timing constraints?

Example – Cruise Control Architecture



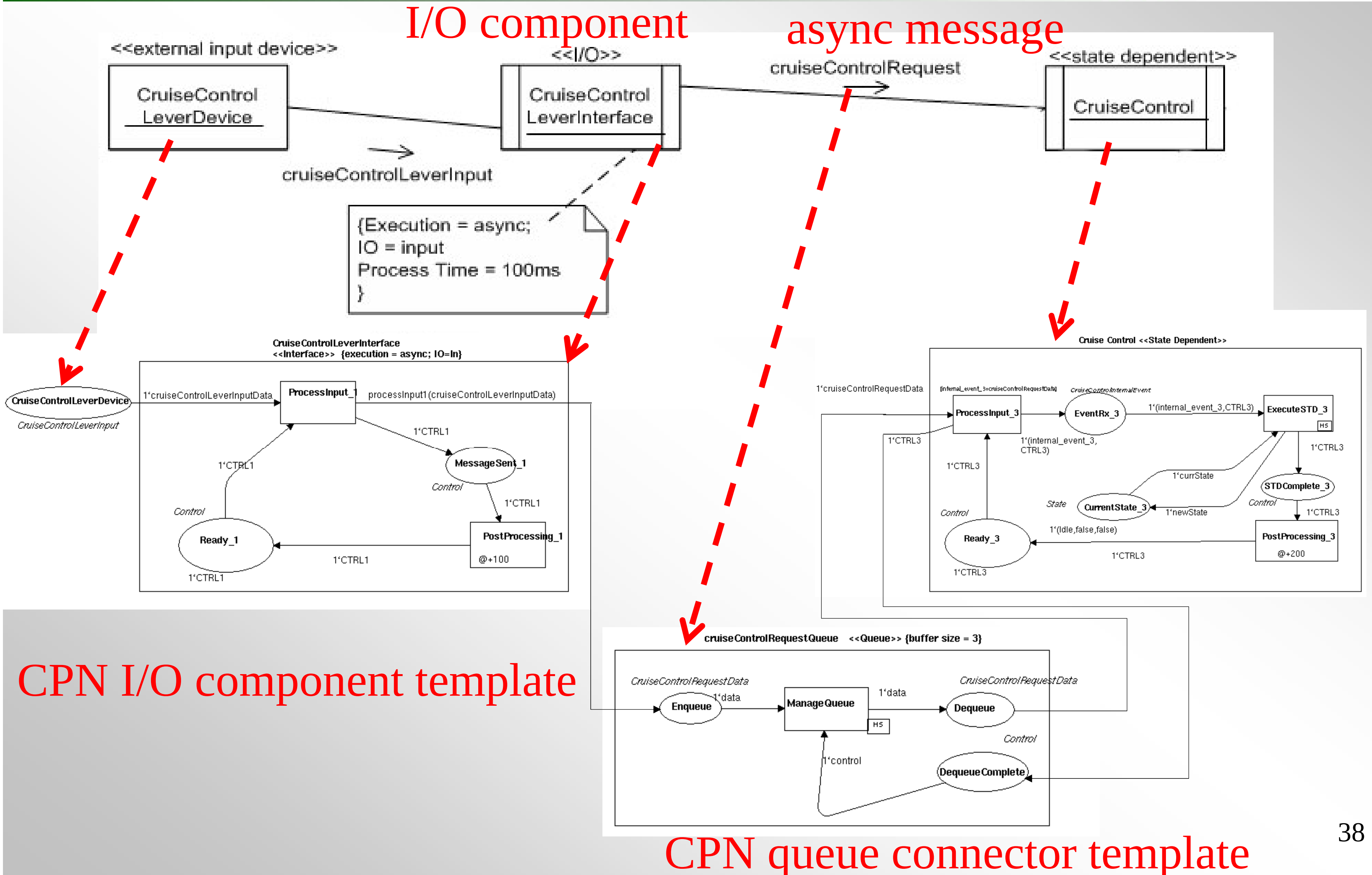
1. Applying CPN Templates – Context Model



2. Construct CPN Architecture Model

- Interconnect CPN templates
  – Decompose context-level CPN model into architecture-level model
  – Each component and connector mapped to CPN template
- CPN Interfaces for components and connectors
  – Concurrent object CPN templates use transitions
  – Passive & connector object templates use places
  – Concurrent object templates are connected to passive / connector object templates
    - Maintains CPN place-transition connection rules

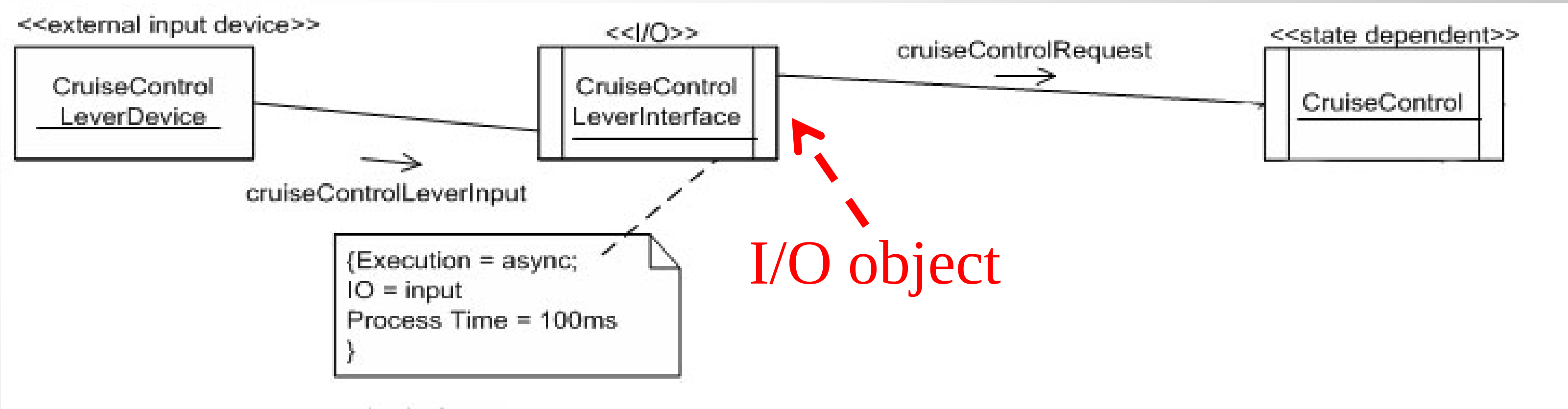
Connecting CPN Templates to form CPN Architecture

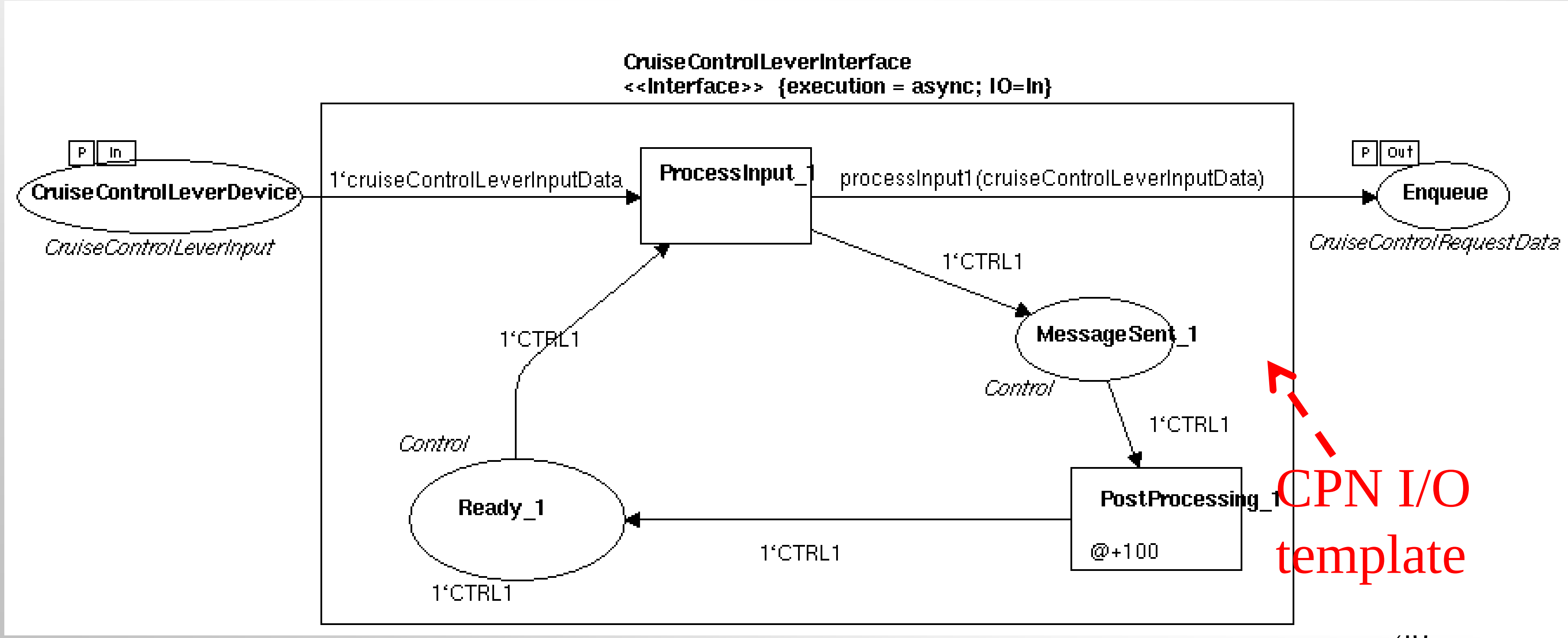


3. Modeling Individual Components

- Each CPN template must be customized to capture specific object behavior
  - Architectural parameters
    - Processing time / sleep time – set CPN timing parameters
    - Buffer size - set queue limits
  - Passive classes
    - Capture attributes and operations to be included in entity objects
    - Exclusion / Access type - capture desired mutual exclusion behavior on entities
  - Message / data specifications
    - Use to define CPN colorsets and token variables

3 Applying CPN Templates – Customizing Individual Object

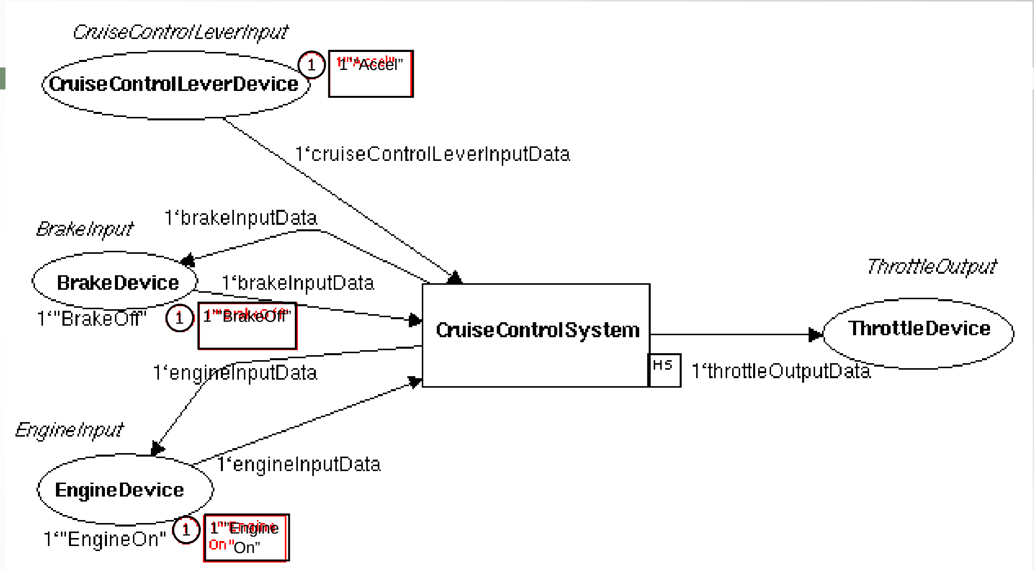




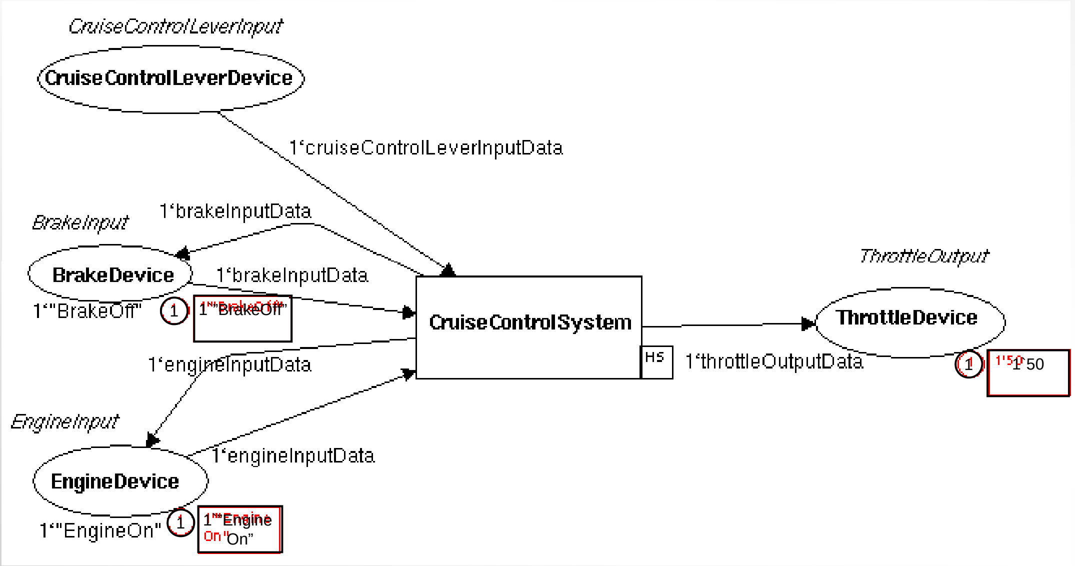
4. Analyzing Software Architecture with CPN Model

- CPN model used to execute architectural design
- Validation
  - Two detailed case studies
  - Exercised using Design CPN simulator
- Functional analysis
  - Execute test scenarios to determine if architecture outputs expected / desired results
  - Can examine architecture at varying levels of detail
- Performance analysis
  - Throughput analysis
  - Timing analysis
  - Queuing backlogs

Analyzing Software Architecture with CPN Model

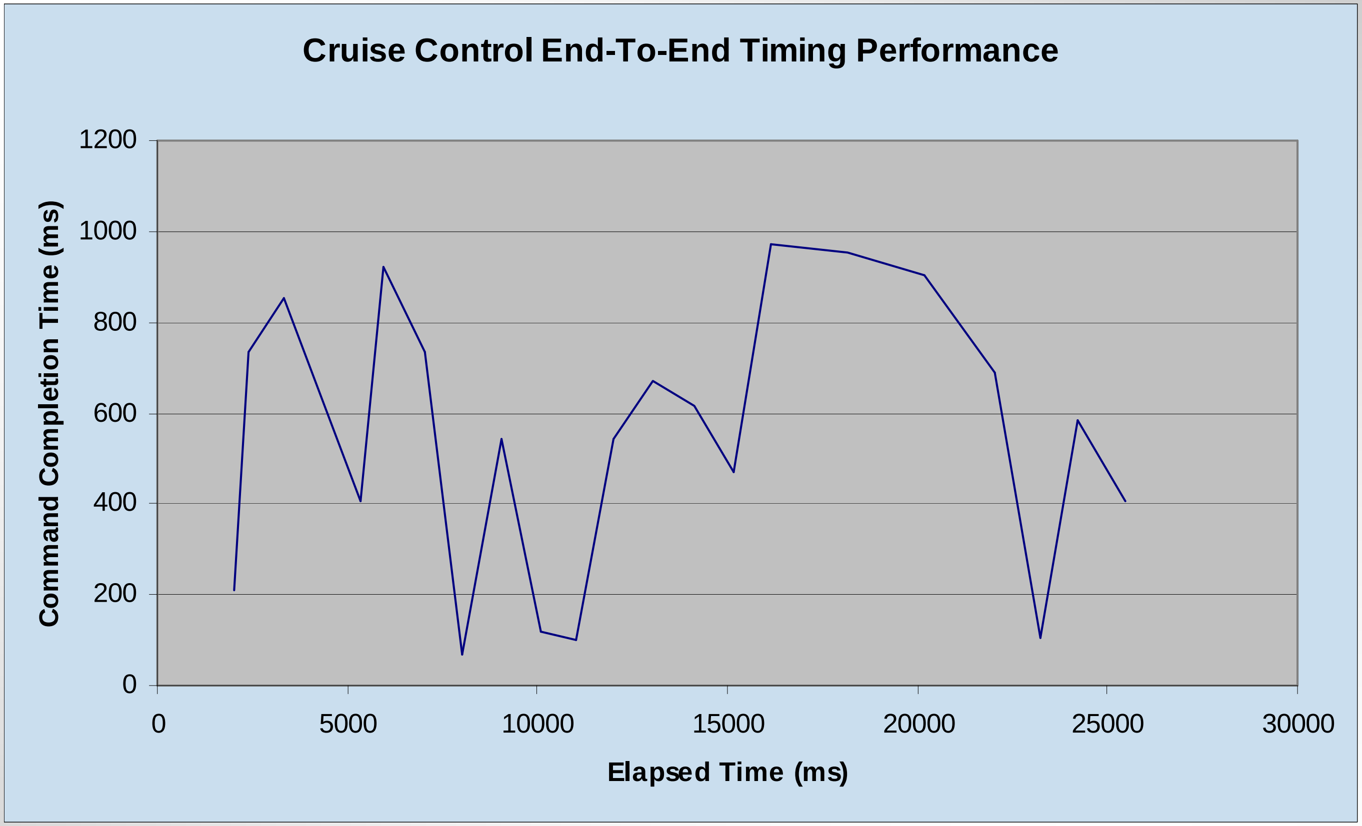


(a)



(b)

Example of Timing Analysis



Conclusions and Future Research

- Dynamic behavior of concurrent system represented using
  - CPN templates
    - Allow systematic, repeatable modeling of object behavioral patterns
    - Maintain structure and integrity of software architecture
- CPN analysis
  - Analyze concurrent behavior at design stage
  - Allows correction of fundamental design problems
- Areas for future work
  - Extend to support distributed environments
  - Investigate scalability to larger models
  - Automate translation to CPN model