ILLINOIS INSTITUTE OF TECHNOLOGY

Transforming Lives. Inventing the Future.
www.iit.edu
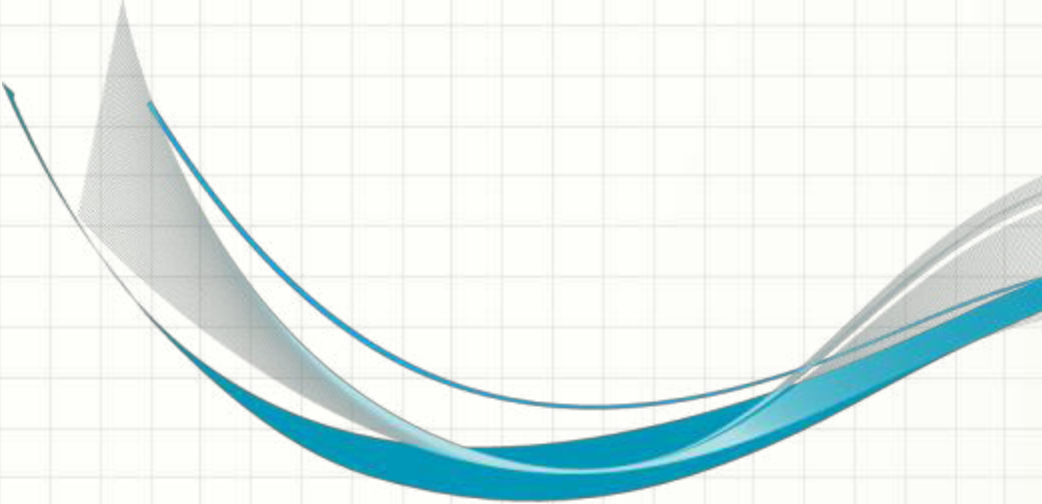
SOFTWARE ENGINEERING
CS 487

Dennis Hood

Computer Science

Summer '19

Homework #5

- Design the means of communication for the fish tank control system
    1. Describe each instance of communication:
        - Include both communication out from the controller and in to the controller
        - Explain the purpose of each
        - Describe any "parameters" which will be passed
    2. Define each instance as the command of your control system's "language"
        - Use only 0's and 1's (binary language)
        - Minimize the length of each command
        - Be sure there is no ambiguity (commands must be unique)
        - Include error messages/codes where necessary
- Please submit by 8/4/19

Lesson Overview

- Service-Oriented Architecture and Embedded Systems
- Reading:
  - Ch. 18 – Service-oriented Software Engineering
  - Ch. 19 – Systems Engineering
  - Ch. 20 – Systems of Systems
- Objectives
  - Discuss the benefits of service/provider relationships
  - Examine the issues involved in designing and implementing such architectures
  - Analyze embedded systems issues including:
    - CCI (computer-computer interfaces)
   - Exception handling
    - Timing, etc.

Lecture 9
Service-Oriented Architecture and Embedded Software

Service-Oriented Architecture (SOA)

- Based on the notion of cooperating entities
  - Entities offer to perform services
  - Information can be made available in a controlled and managed way
- Defined interfaces must be published
- Allows for the creation of configurable, distributed, platform-independent systems

Service-Oriented Architecture
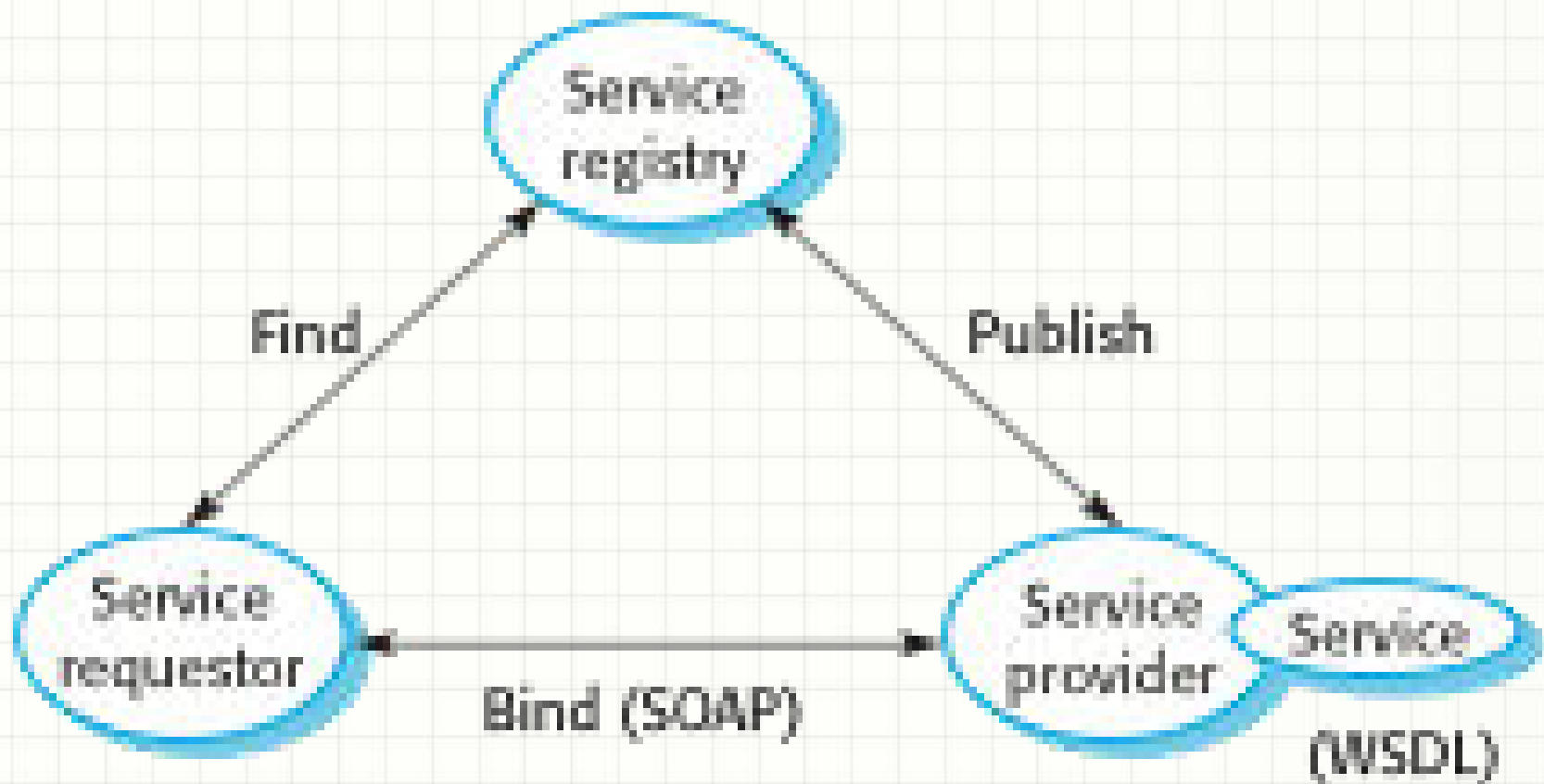
Web Services

- A web service is an instance of a more general notion of a service:

  > "an act or performance offered by one party to another. Although the process may be tied to a physical product, their performance is essentially intangible and does not normally result in ownership of any of the factors of production."

- The provision of the service is independent of the application using the service

- Service providers can develop specialized services and offer these to a range of service users from different organizations

SOA-related Standards

- SOAP – SOA Protocol
  - Message interchange
  - Supports communication between services
- WSDL – Web Services Definition Language
  - Service interface definition
  - Establishes definition for service operations and bindings
- WS-BPEL – Web Services Business Process Execution Language
  - Workflow language

Benefits of SOA

- Services can be provided locally or outsourced to external providers
- Services are language-independent
- Investment in legacy systems can be preserved
- Inter-organizational computing is facilitated through simplified information exchange
- Reuse

Ex.: Car Information System
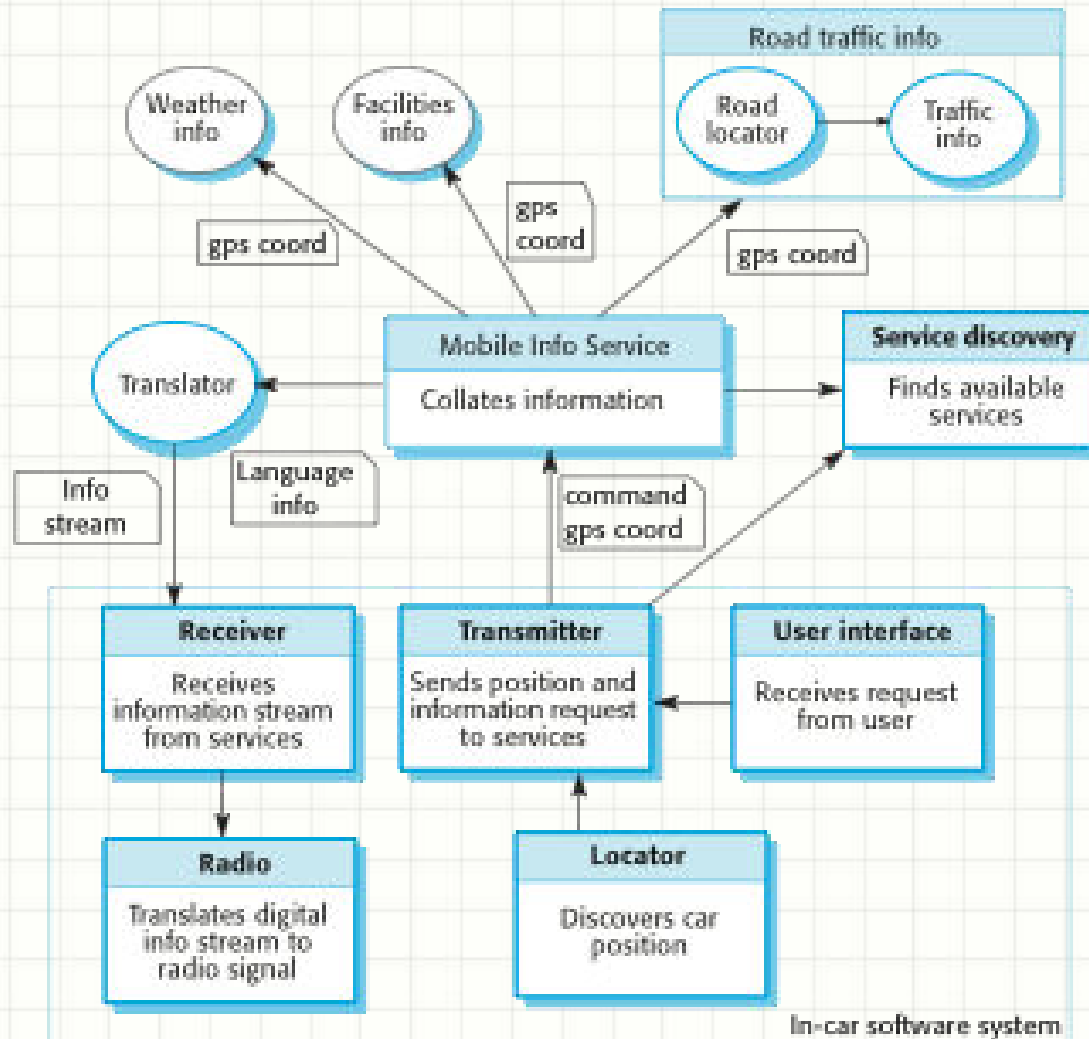
Ex.: Car Info Systems (cont.)

- It is not necessary to decide when the system is programmed or deployed what service provider should be used or what specific services should be accessed
  - As the car moves around, the in-car software uses the service discovery service to find the most appropriate information service and binds to that
  - Because of the use of a translation service, it can move across borders and therefore make local information available to people who don't speak the local language

Reusable Services

- Once established, a service can be used by any number of "systems"
  - Well-defined interface
  - Consistent, reliable performance
  - Independent and loosely coupled
- WSDL specification
  - What – the interface description
  - How – details of how to communicate
  - Where – the location of the implementation

WSDL Specification
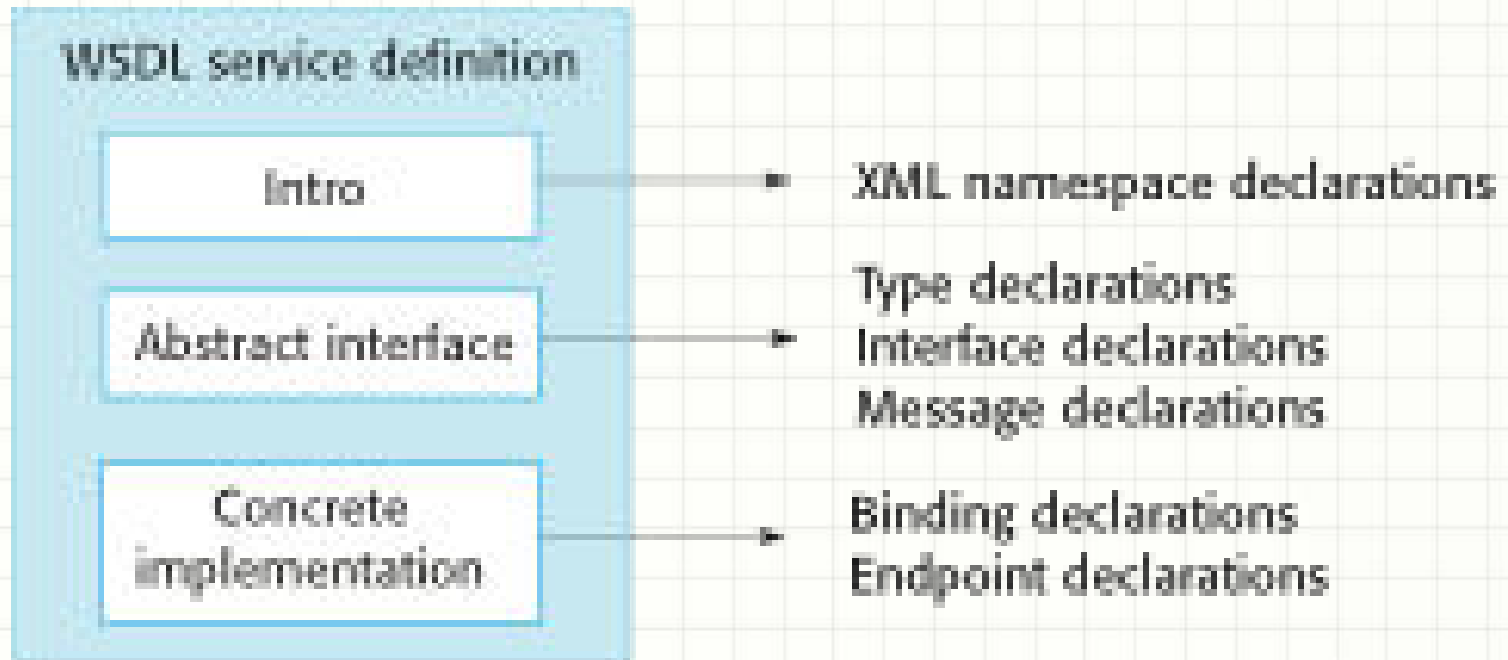
Service Types

- Utility services
  - General functionality
  - e.g., currency conversion
- Business services
  - Associated with a specific business function
  - e.g., student registration
- Coordination of process services
  - Support more general business processes
  - e.g., procurement process management

Service Engineering

- The process of developing services for reuse in service-oriented applications
- The service has to be designed as a reusable abstraction that can be used in different systems
- Generally useful functionality associated with that abstraction must be designed and the service must be robust and reliable
- The service must be documented so that it can be discovered and understood by potential users

Service Engineering
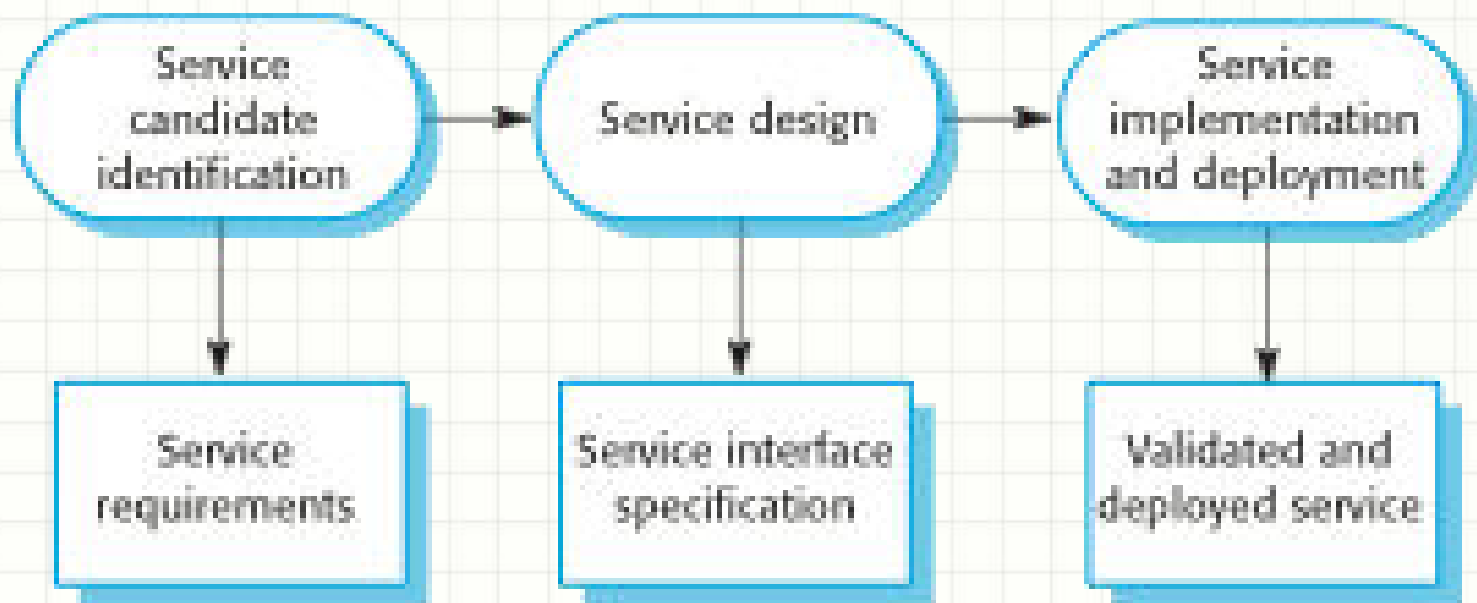
Service Testing Considerations

- Control or even access to external services is determined by the service provider
- An application may not always use the same service each time it is executed
- Performance may differ under varying loads – loads which can vary greatly depending on the number and frequency of service requests
- Testing of exception handling may depend on proper failure of dependent services

Embedded Systems

- Characteristics
  - Software embedded in hardware
  - Minimal user interface / interaction
  - Real-time response to environmental change
  - A product of the environment
- Performance
  - Proper performance means a correct response within an acceptable time
  - Availability is expected
  - Safety and reliability issues are prominent in design decisions

Figure 20.1 Stimuli and responses for a burglar alarm system

Stimulus	Response
Single sensor positive	Initiate alarm; turn on lights around site of positive sensor.
Two or more sensors positive	Initiate alarm; turn on lights around sites of positive sensors; call police with location of suspected break-in.
Voltage drop of between 10% and 20%	Switch to battery backup; run power supply test.
Voltage drop of more than 20%	Switch to battery backup; initiate alarm; call police; run power supply test.
Power supply failure	Call service technician.
Sensor failure	Call service technician.
Console panic button positive	Initiate alarm; turn on lights around console; call police.
Clear alarms	Switch off all active alarms; switch off all lights that have been switched on.

Design Considerations

- Platform selection
  - Timing, power, environment, OS, etc.
- Stimuli response
  - Periodic vs. aperiodic stimuli
- Timing requirements
- Process architecture
  - Aggregate stimuli and response processes
- Algorithms to support computations
- Data architecture
- Process scheduling

Process Coordination

- Processes in a real-time system have to be coordinated and share information

- Process coordination mechanisms ensure mutual exclusion to shared resources

- When one process is modifying a shared resource, other processes should not be able to change that resource

- When designing the information exchange between processes, you have to take into account the fact that these processes may be running at different speeds

Real-time System Modelling

- The effect of a stimulus in a real-time system may trigger a transition from one state to another

- State models are therefore often used to describe embedded real-time systems

- UML state diagrams may be used to show states and state transitions in a real-time system

Figure 20.5 State machine model of a petrol (gas) pump
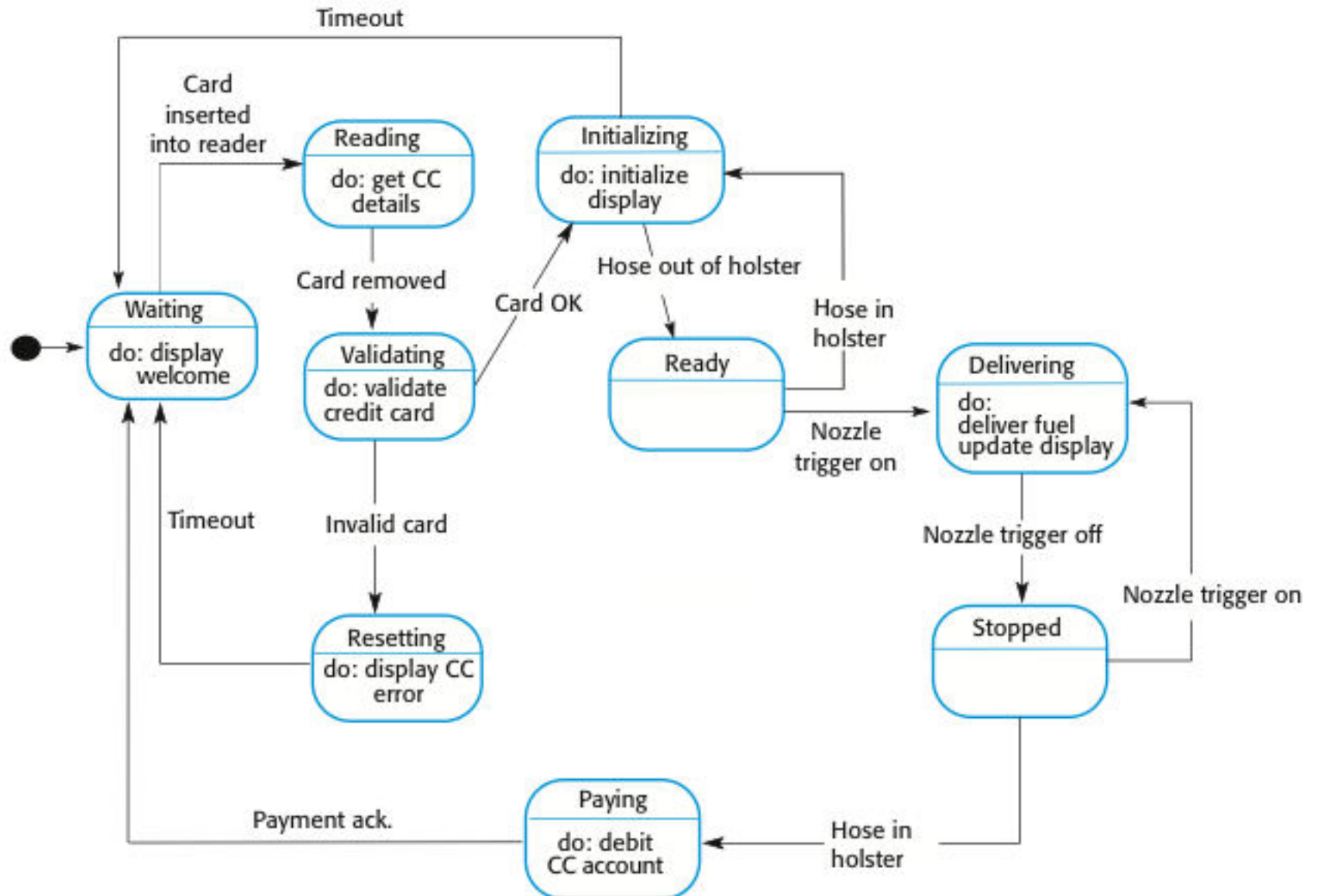
Real-time Design Patterns

- Observe and React
  - Sensors are monitored and status displayed
  - Sensor change initiates handler
- Environmental Control
  - Sensors monitor the environment and actuators adjust it
  - Sensor change initiates signals to actuators
- Process Pipeline
  - Data transformation is required before processing
  - Separate processors to handle concurrent transformations

Figure 20.11 Control system architecture for an anti-skid braking system
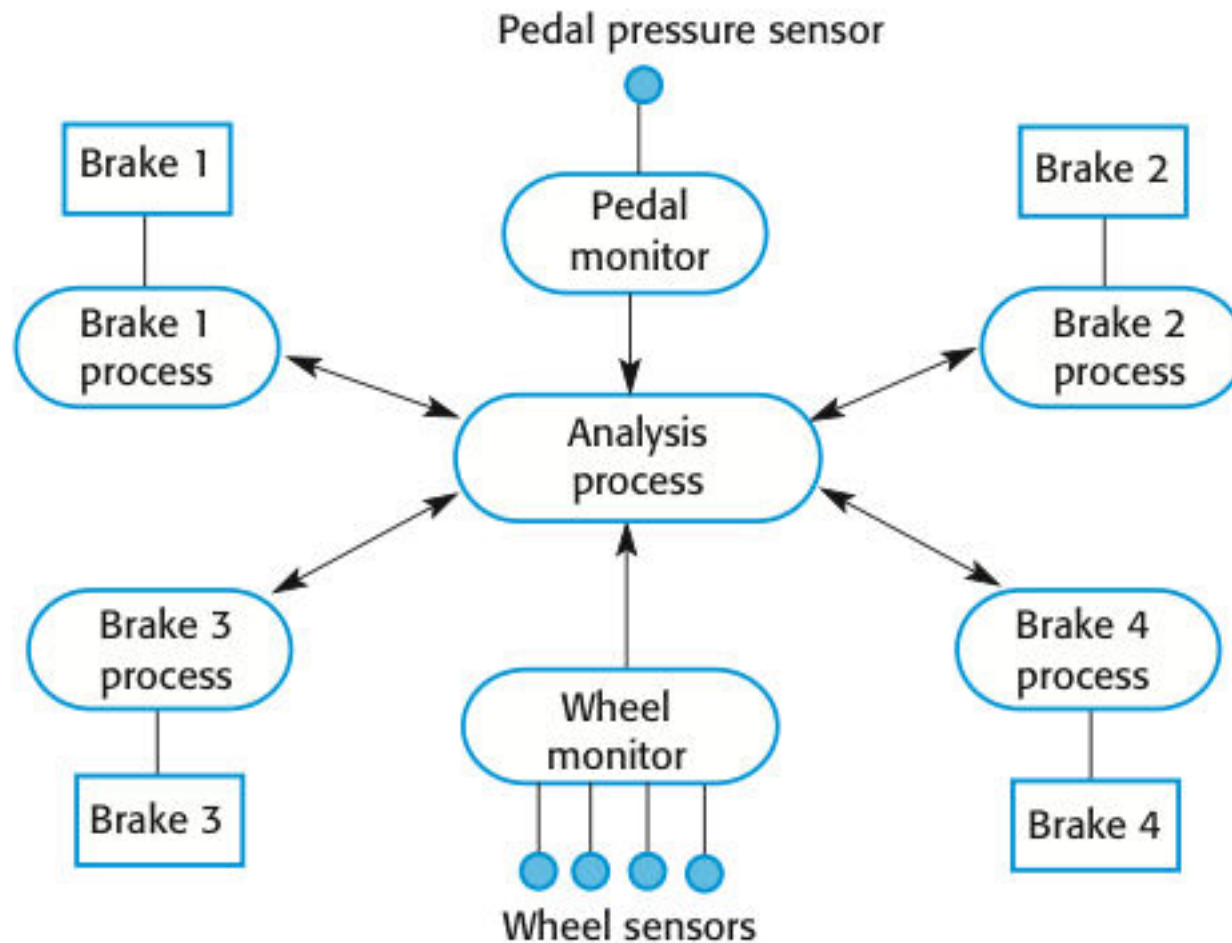
Timing Analysis

- The correctness of a real-time system depends on both the correctness of the outputs as well as the time at which they are produced
- Timing can be difficult with a mixture of periodic and aperiodic stimuli and responses
- Key factors in analyzing timing requirements:
  - Deadlines – the time by which stimuli must be processed
  - Frequency – the number of times per second that a process must execute
  - Execution time – the time required to process a stimulus and produce a response (consider both average and worst-case)

Real-time Operating Systems

- Real-time applications often require a more efficient and responsive real-time operating systems

- RTOS components:
  - A real-time clock for periodic events
  - An interrupt handler for aperiodic requests
  - A scheduler for examining and selecting processes for execution
  - A resource manager for allocating system resources such as memory
  - A dispatcher for starting the execution of processes

Figure 20.17 Components of a real-time operating system