Topic Coverage

1. Object Models
2. Event Ordering specifications
3. Simple Programs
4. Parallelism and Inheritance

Execution Models for Objects

Object or Class Definition

- **Member Variables**
- **Methods**
- **State Model**
- **Set of Events { (generated events), (received events)}**
  - **(event = message or method invocation)**

Execution Model

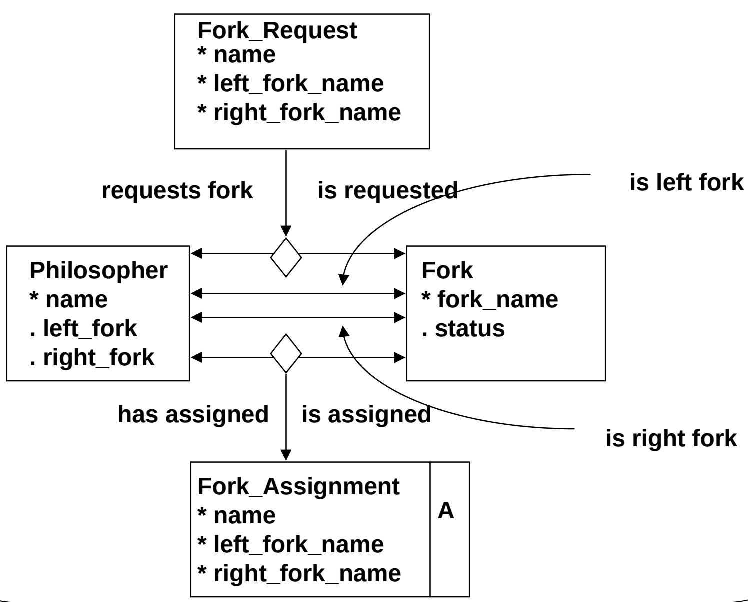
- **Each object instance has a state machine controlling the order of execution of its methods.**
- **A dynamic set of object instances interacting by sending events to one another in the order determined by the state models of the object instances.**

Object Models and Event-Sequenced Model of Programming

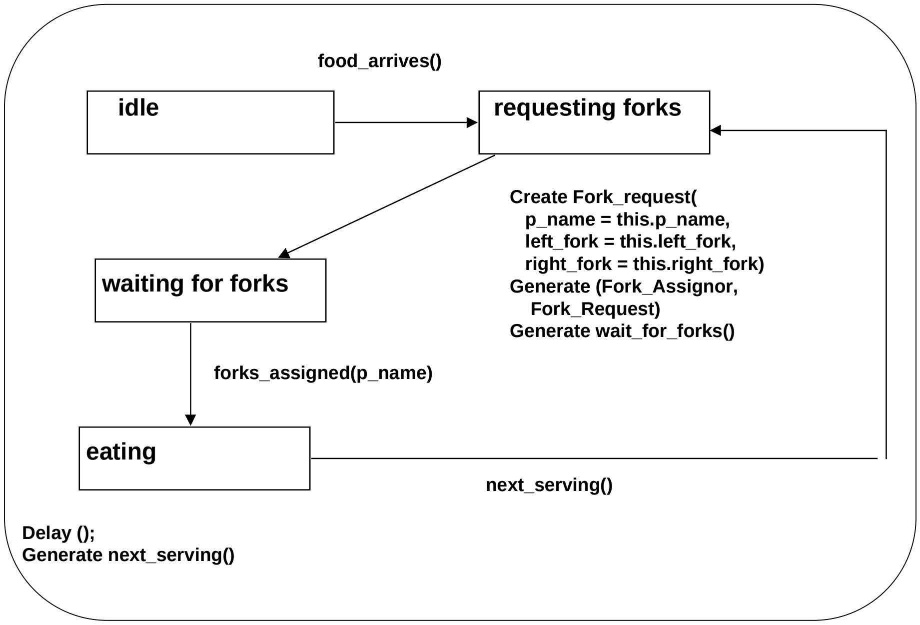
The Dining Philosophers Problem

There are five philosophers sitting at a round table. There are five forks on the table, one fork is placed between each pair of philosophers. Each philosopher can reach only the forks immediately to his/her left and right. The food to be served requires two forks to eat. Each philosopher will attempt to grab two forks and eat immediately when the food is brought to the table. Each philosopher will courteously put down both forks after one serving and immediately attempt to pick them back up again. It is desired to regulate the behavior of the philosophers so that each philosopher will obtain two forks and thus be able to eat within a reasonable time while at the same time maximizing the number of philosophers who are simultaneously dining.

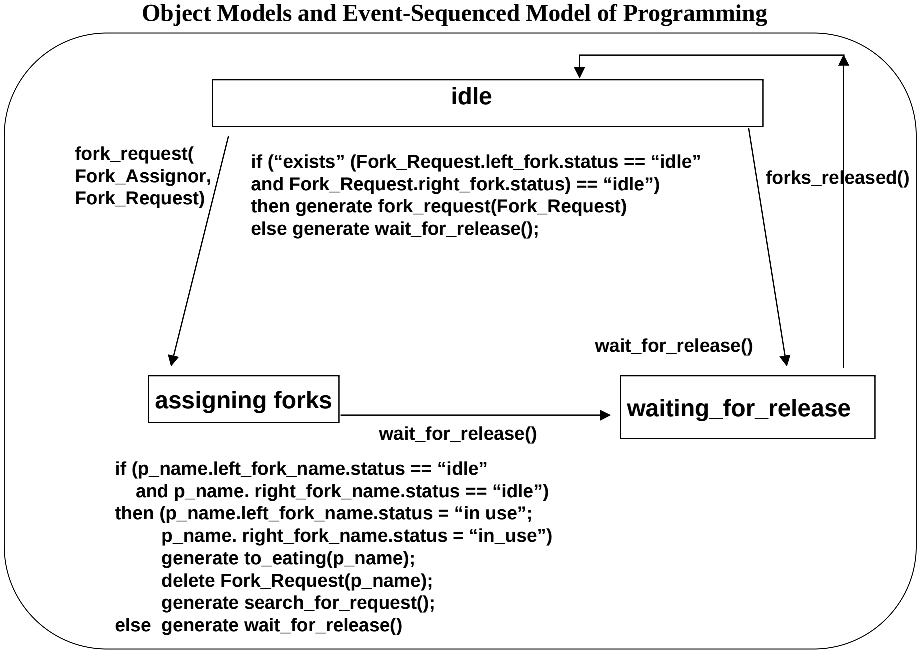
Object Models and Event-Sequenced Model of Programming



Object Models and Event-Sequenced Model of Programming



Object Models and Event-Sequenced Model of Programming



Execution Model - Continued

The execution of an instance of this family of programs where each object has been designed as a part of a system goes as follows:

1. **Create an initial set of object instances where each object is in its initial state.**
2. **Assign a process or thread to execute the state machine of each object instance**
3. **Begin the execution of this set of object instances.**
4. **The object instances will interact by sending events. The sequence of events is governed by the behaviors encoded into the state machines.**

How to Create Such Programs

1. **Download Rational's UML editor package.**

2. **Create an object.**

3. **Use the Java class template generator.**

4. **Insert an event checker as the first executable state of each method in a try/catch clause. If the event is not the event which is expected to initiate this method then throw an exception.**

Or

1. **Create a Shlaer/Mellor OOA model and use a code generator.**

Object Models and Event-Sequenced Model of Programming

Let us now consider an execution model for objects where either a state model has not been specified or the state models have not been designed together.

1. Specify an event ordering constraint expression to sequence the occurrence of each event. An event is the execution of a method or action.
2. Create an initial set of object instances where each object is in its initial state.
3. Assign a process or thread to execute the methods of each object instance
4. Begin the execution of this set of object instances.
5. Continue the execution by insuring that the methods or actions are executed in conformance to the specifications in the event ordering constraint expression.

Object Models and Event-Sequenced Model of Programming

Consider the following programs:

1. A simple Producer/Consumer program where the Producer and the Consumer interact through an infinite buffer.
2. The Consumer is slower than the Producer so that two copies of the Consumer program is needed to keep up with one producer.
3. The Producer and Consumer interact through a finite buffer of size N.

Object Models and Event-Sequenced Model of Programming

```
examprog1 ( )
{
    channel buf ;

    producer (buf) || consumer (buf) ;
}
```

(a) Concurrent program

```
producer (channel buf)
{
    while (TRUE)  {
        info = produce ( ) ;
        send (buf, info) ;
    }
}
```

```
consumer (channel buf)
{
    while (TRUE)  {
        info = receive (buf) ;
        consume (info) ;
    }
}
```

(b) Producer component

(c) Consumer component

The Infinite Buffer Producer-Consumer Problem

Object Models and Event-Sequenced Model of Programming

```
consumer1 (channel buf, sync)
{
    myTurn = myId % 2 ;
    while  (TRUE)  {
        if  (myTurn)  {
            info = receive (buf) ;
            consume (info) ;
            send (sync, ack) ;
            myTurn = FALSE ;
        } else  {
            ack = receive (sync) ;
            myTurn = TRUE ;
        }
    }
}
```

(a) Modified consumer program

```
examprog2 ( )
{
    channel buf ;
    channel sync ;

    producer (buf) ||
    consumer1 (buf, sync) ||
    consumer1 (buf, sync) ;
}
```

(b) Extended examprog1

The two Consumer Producer/Consumer Program

Object Models and Event-Sequenced Model of Programming

```
examprog3 ( )
{
    channel buf ;
    producer1 (buf) || consumer2 (buf) ;
}
```

(a) Modified concurrent program

```
producer1 (channel buf)
{
    while (TRUE)  {
        info = produce ( ) ;
        send (buf, info) ;
        count = count+1 ;
        if (count == n)  {
            ack = receive (buf) ;
            count = count - 1 ;
        }
    }
}
```

(b) Modified Producer

```
consumer2 (channel buf)
{
    while (TRUE)  {
        info = receive (buf) ;
        consume (info) ;
        send (buf, ack) ;
    }
}
```

(c) Modified consumer

The finite buffer Producer/Consumer Problem

Object Models and Event-Sequenced Model of Programming

Observations

1. We had to modify each of the Producer and Consumer programs because of their interactions with each other, NOT because the function that they are executing changed.

2. This is a result of the fact that the programs interleave specifications of interaction and computation.

3. Perhaps we should separate specification of interaction and computation!

4. Question: How can we separate specification of computation and interaction?

a) Introduce specifications for where interactions occur in programs.
b) Associate an event with every execution of a program or method.
c) Define a language for specification of the order in which events occur.

Object Models and Event-Sequenced Model of Programming

```
producer (buffer info)
{
  while (TRUE)  {
    info.produce ( ) ;
  }
}
```

(a) Representation of producer

```
consumer (buffer info)
{
  while (TRUE)  {
    info.consume ( ) ;
  }
}
```

(b) Representation of consumer

Representations of interacting programs

Object Models and Event-Sequenced Model of Programming

```
producer (buffer info ; prod)
{
  computation :
    while (TRUE)  {
      info.produce ( ) ;
    }
  interaction point :
    prod names info.produce ;
}
```

(a) Representation of producer

```
consumer (buffer info ; cons)
{
  computation :
    while (TRUE)  {
      info.consume ( ) ;
    }
  interaction point :
    prod names info.consume ;
}
```

(b) Representation of consumer

Let us add explicit specification of where the programs interact.

Object Models and Event-Sequenced Model of Programming

```
<evoce>          :: =        <primEvoce>
                        |   <nonPrimEvoce>
<primEvoce>      : : =      ( <eventId> ⧀ <eventId> )
<nonPrimEvoce>   : : =      <evoce> ⩒ <evoce>
                        |   <evoce> ⩑ <evoce>
                        |   forall var <varIter> in <eventSet>
                                 <evoce>
                        |   forall occ <varOcc> in <eventSet>
                                 <evoce>
                        |   exists var <varIter> in <eventSet>
                                 <evoce>
                        |   exists occ <varOcc> in <eventSet>
                                 <evoce>
                        |   (<evoce> )
```

BNF expression for event ordering constraint expressions

Object Models and Event-Sequenced Model of Programming

ConsExp1 = forall occ i in produce :
produce [i] ⧀ consume [i]

Event Ordering Expression for the Producer/Consumer Problem

SingleBuffer = forall occ *i* in produce :
(produce [*i*] ⧀ consume [*i*]) ⩑
forall occ *i* in consume :
(consume [*i*] ⧀ produce [*i*+1])

Event Ordering Expression for the Single Buffer Producer/Consumer Problem

Object Models and Event-Sequenced Model of Programming

examprog2 = (producer || consumer || consumer)
where
ConsExp2

(produce [2*i-1] < consume [i])
(produce [2*i] < consume2 [i]

Event ordering specification for the two Consumer Producer/Consumer problem.

ConsExp2 = forall occ *i* in produce :
(produce [2*i-1] < consume [i]) ∧
(produce [2*i] < consume2 [i])

Object Models and Event-Sequenced Model of Programming

Examprog3 = (producer || consumer)
where
ConsExp3

ConsExp3 = ConsExp1 ∧
forall occ i in consume:
(consume[i] < produce[i+N])

Event ordering specification for finite buffer Producer/Consumer problem.

Object Models and Event-Sequenced Model of Programming

What have we done?

1. State machines have been subsumed in the event ordering constraint expressions. (The evoce are expressions in a first order linear logic.)

2. The same Producer and Consumer programs are used in all of the three programs.

3. These programs can and have been compiled.

Object Models and Event-Sequenced Model of Programming

What is the benefit of this approach?

1. One version of a functional component is sufficient.

2. The interactions of the programs are explicitly visible.

3. Formal verification of corrrectness can be performed.

4. This model of execution can be used for debugging.
- **a) Write out the event ordering constraint expression.**
- **b) Trace the execution of the programs.**
- **c) Parse the trace with the event ordering constraint expression.**