AARHUS
UNIVERSITY
DEPARTMENT OF ENGINEERING

Architecture & Design of Embedded Real-Time Systems (TI-AREM)

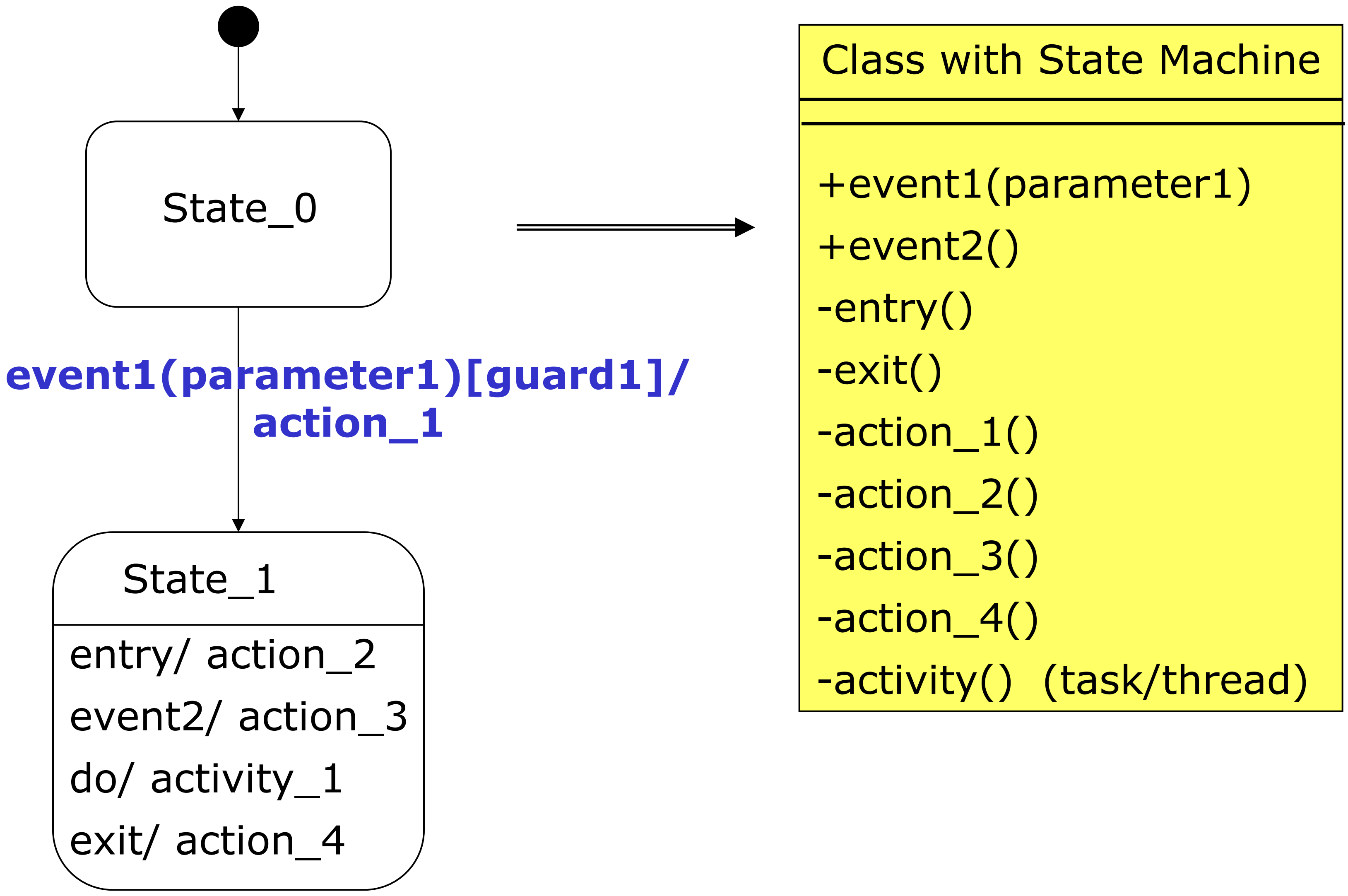
State Machine Implementation

Version: 29-1-2015

Agenda

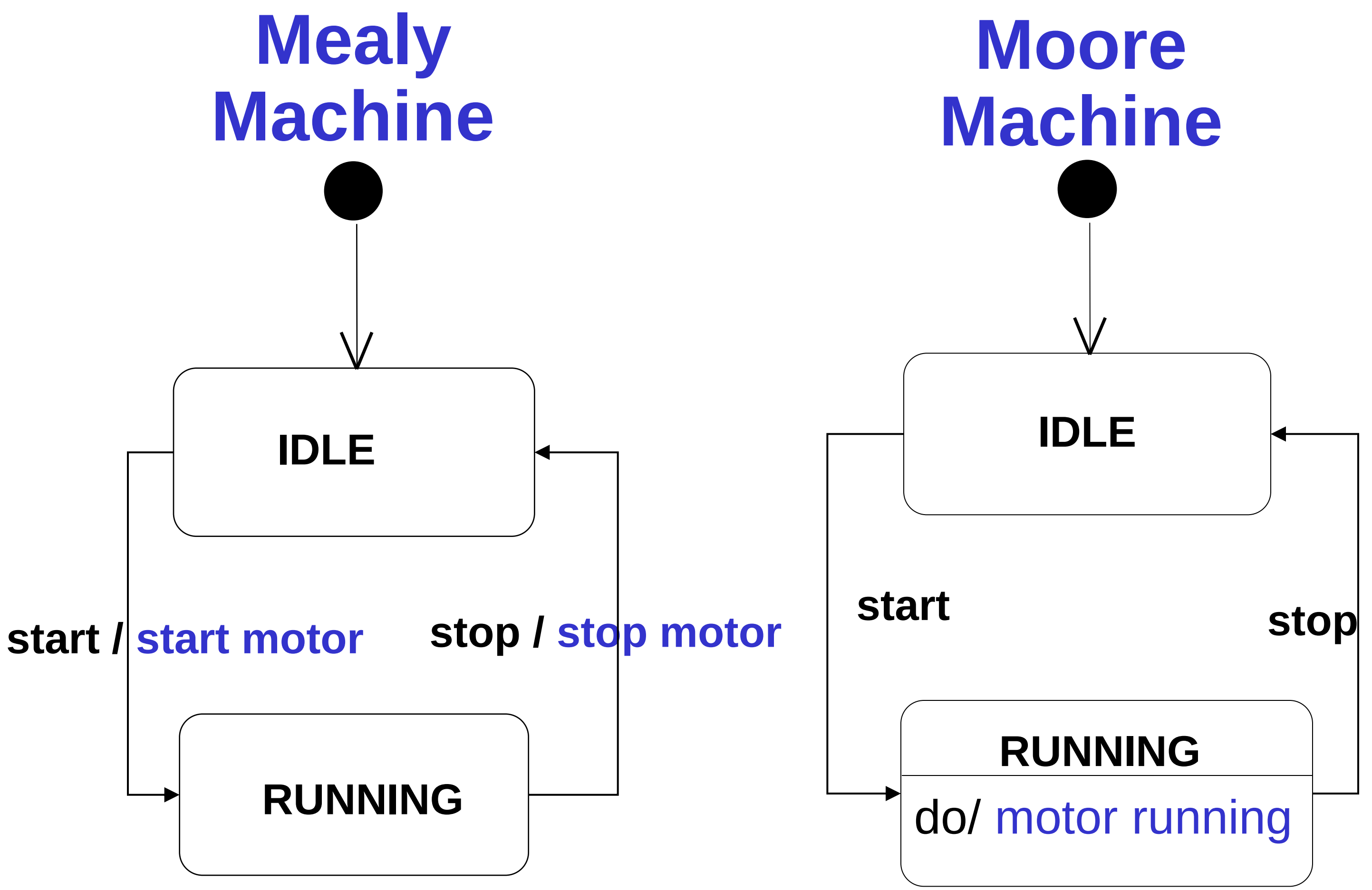
- STM notation and example
- Five state machine implementation techniques:
    1. Switch based
    2. Table driven implementation
    3. GoF State Pattern (separate slides)
- Case tool STM generation

UML State Diagram Notation

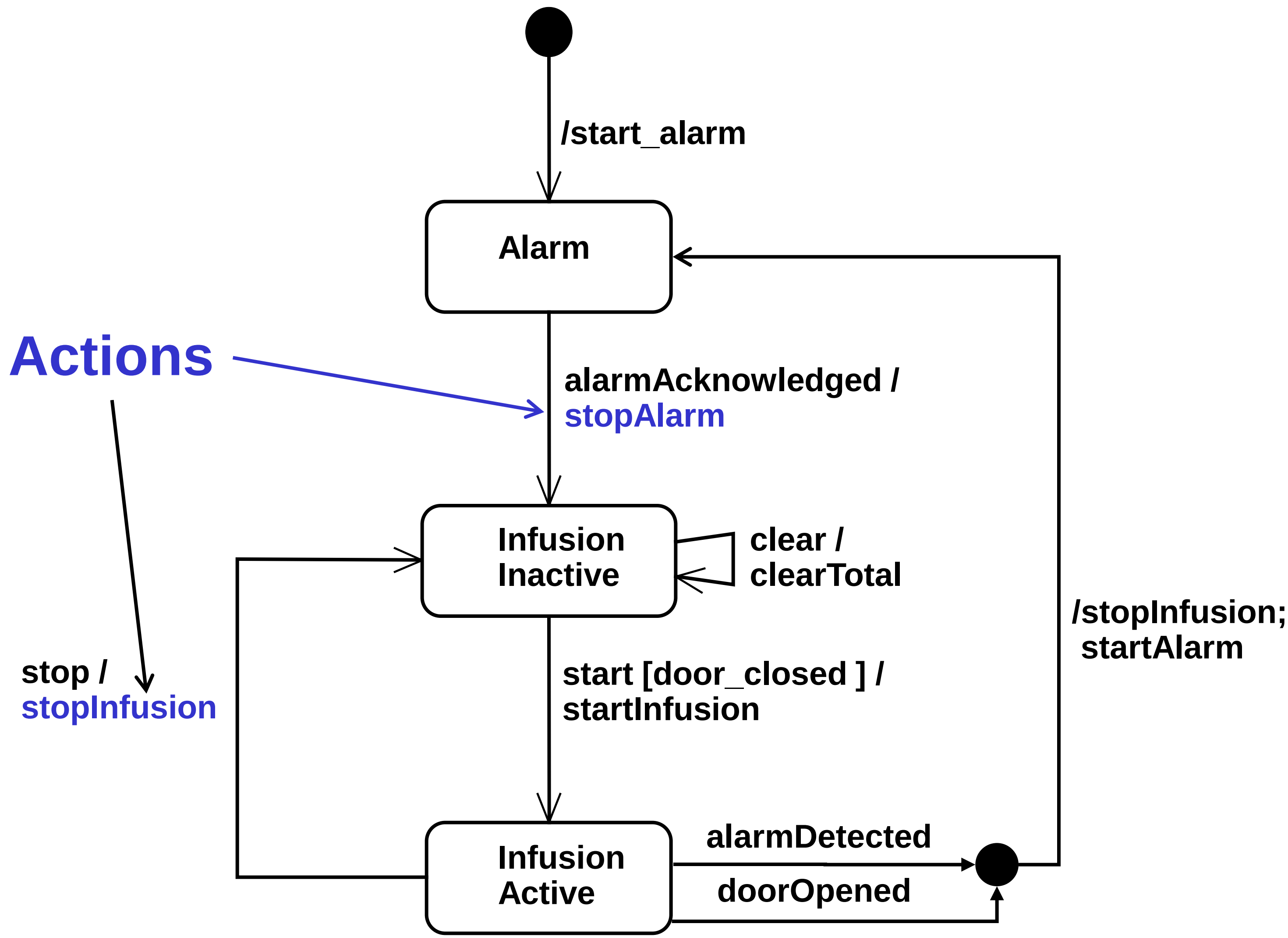


AARHUS UNIVERSITY
DEPARTMENT OF ENGINEERING

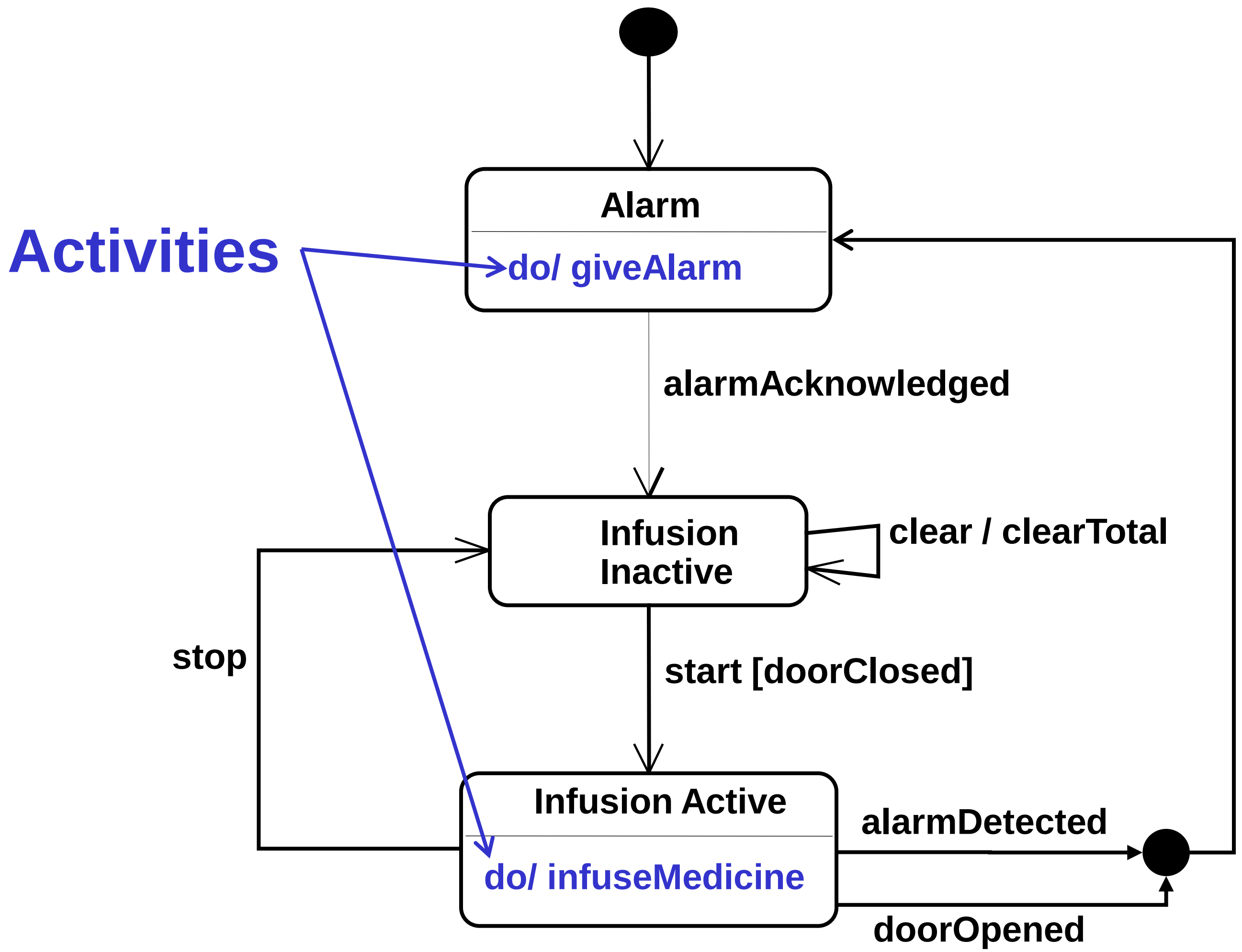
Two State Machine Types



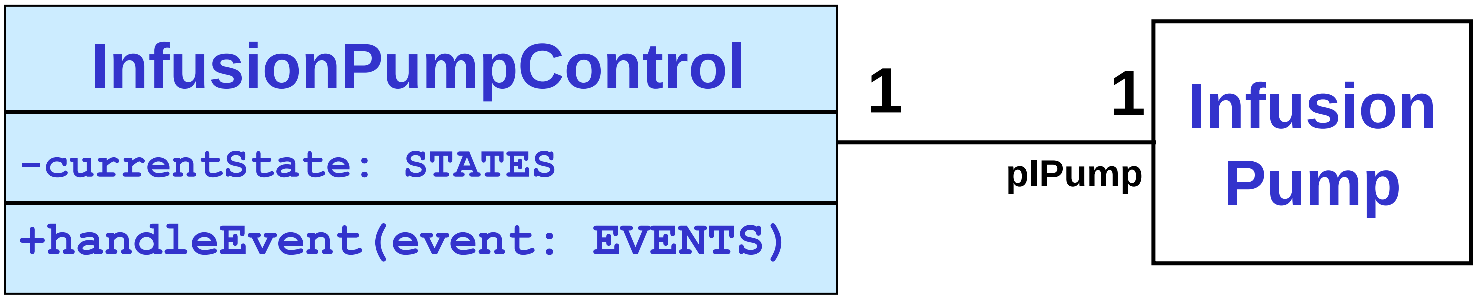
Example: Infusion Pump as a Mealy Machine



Example: Infusion Pump as a Moore Machine



1. Switch Based Implementation (1)




```
main()    // Mealy implementation
{
   InfusionPump infPump;
   InfusionPumpControl iPumpControl(&infPump);
   EVENTS event;

   while(FOREVER)
   {
       event = readEvent();
      iPumpControl.handleEvent(event);
   }
}
```

Switch Based Implementation (2)

```
class InfusionPumpControl
{
  public: // constructor
        InfusionPumpControl(InfusionPump* pIP)
        {    pIPump= pIP;
             currentState= ALARM;
             startAlarm();
        }
        void handleEvent(Events event);
  private:
        STATES currentState;
        InfusionPump* pIPump;
        startAlarm();
        stopAlarm();
        clearTotal();
}
```

Switch Based Implementation (3)

```
void InfusionPumpControl::handleEvent(Events event)
{
  switch (currentState)
  {
    case ALARM:
      if (event == ALARM_ACKNOWLEDGED)
      {
        stopAlarm();
        currentState = INFUSION_INACTIVE;
      }
    break;

    case INFUSION_INACTIVE:
      if (event == CLEAR)
        clearTotal();
      else if ((event == START) && doorClosed())
      { // Guarded transition
        pIPump->startInfusion();
        currentState = INFUSION_ACTIVE;
      }
    break;
```

Discussion of Switch-based Implementation

- Advantages:
  - Very easy and straightforward to implement
  - Easy to implement guarded transitions and entry/exit actions
- Disadvantages:
  - Problem with event parameters
  - The logic and the behavior are tightly interconnected
  - For larger state machines:
    - the nested switch/case statements becomes very difficult to read and modify
- **Recommendation:**
  - Use it only, for small and simple state machines

2. State-Event Table Implementation

	Event1	Event2	Event3	....
State1	**action1** **State2**	-	**action5** **State3**	
State2	-	**action3** **-**	**action6** **State1**	
State3	**action2** **State1**	**action4** **State2**	-	

New state

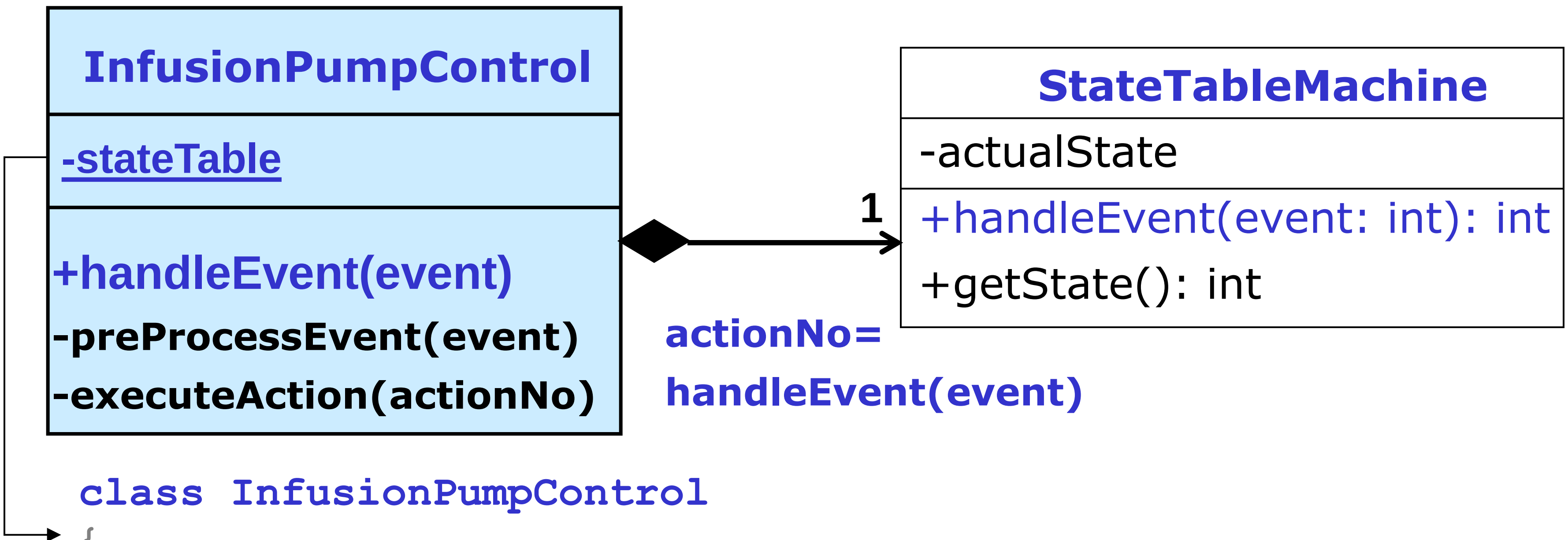
Infusion Pump State-Event Table

Notice

	start AND doorClosed	stop	alarm Acknowledged	alarmDetected OR doorOpened	clear
ALARM	**X**	**X**	stopAlarm **INFUSION INACTIVE**	**X**	**X**
INFUSION INACTIVE	start infusion **INFUSION ACTIVE**	**X**	**X**	**X**	clear Total **X**
INFUSION ACTIVE	**X**	stop Infusion **INFUSION INACTIVE**	**X**	startAlarm, stopInfusion **ALARM**	**X**

X represents don't cares

State-Event Table Implementation (1)




```
class InfusionPumpControl
{
    static int[ ] stateTable =
      {
        /* EVENTS            E1       E2       E3       E4       E5       */
        /* ALARM    0 */   NO,NO,  NO,NO,  A1,1,   NO,NO,  NO,NO,
        /* INF_INA  1 */   A2, 2,  NO,NO,  NO,NO,  NO,NO,  A3,NO,
        /* INF_ACT  2 */   NO,NO,  A4, 1,  NO,NO,  A5, 0,  NO,NO
      };
```

State-Event Table Implementation (2)

```
main()
 {
   int eventNo, actionNo;
   InfusionPumpControl iPumpControl;
   while ((eventNo=getEvent())!= END_PROGRAM)
       iPumpControl.handleEvent(eventNo);
}
```

State-Event Table Implementation (3)

```
InfusionPumpControl::InfusionPumpControl()
{
     pSTM = new stateTableMachine(
                    stateTable,NoOfEvents,
                    ALARM);
     startAlarm(); // call initial action
}

InfusionPumpControl::handleEvent(int eventNo)
{
     int pEvent= preProcessEvent(eventNo);
     int actionNo= pSTM->handleEvent(pEvent);
     executeAction(actionNo);
}
```

State-Event Table Implementation (4)

```
StateTableMachine::StateTableMachine(int[] _states,
                    int maxEvent, int initState)
{
    nofEvents = maxEvent;  states = _states;
    actualState = initState;
}

int StateTableMachine:: handleEvent(int eventNo)
{
  int index = ((actualState * nofEvents) + eventNo) * 2;

  // look for a state transition
  if (states[index+1] != State.NO)
    actualState = states[index+1];

  // return action
  return (states[index]);
}

int StateTableMachine:: getState()
{return actualState;}
```

State-Event Table Implementation (5)

```
int InfusionPumpControl::preProcessEvent(int inEvent)
{
    switch (inEvent)
    {
      case START:
           if (doorClosed())     // GUARDED TRANSITION
                return E1; break;
      case STOP: return E2; break;
      case ALARM_ACKNOWLEDGED: return E3; break;
      case ALARM_DETECTED:
      case DOOR_OPENED: return E4; break;
      case CLEAR: return E5; break;
      default: return NO_EVENT;
    }
    return NO_EVENT;
}
```

State-Event Table Implementation (6)

```
void InfusionPumpController::executeAction(int actionNo)
  {
    switch (actionNo)
    {
      case NO: cout   << "NO action defined in ″;
           break;
      case A1: cout << "A1: stopAlarm; new ";
           break;
      case A2: cout << "A2: startInfusion; new ";
           break;
      case A3: cout << "A3: clearTotal; new ";
           break;
      case A4: cout << ″A4: stopInfusion; new ";
           break;
      case A5: cout << ″A5: startAlarm, stopInfusion; new ";
           break;
      default: cout << "undefined action in ″;
    }
    cout << "state = " << pSTM->getState());
  }
```

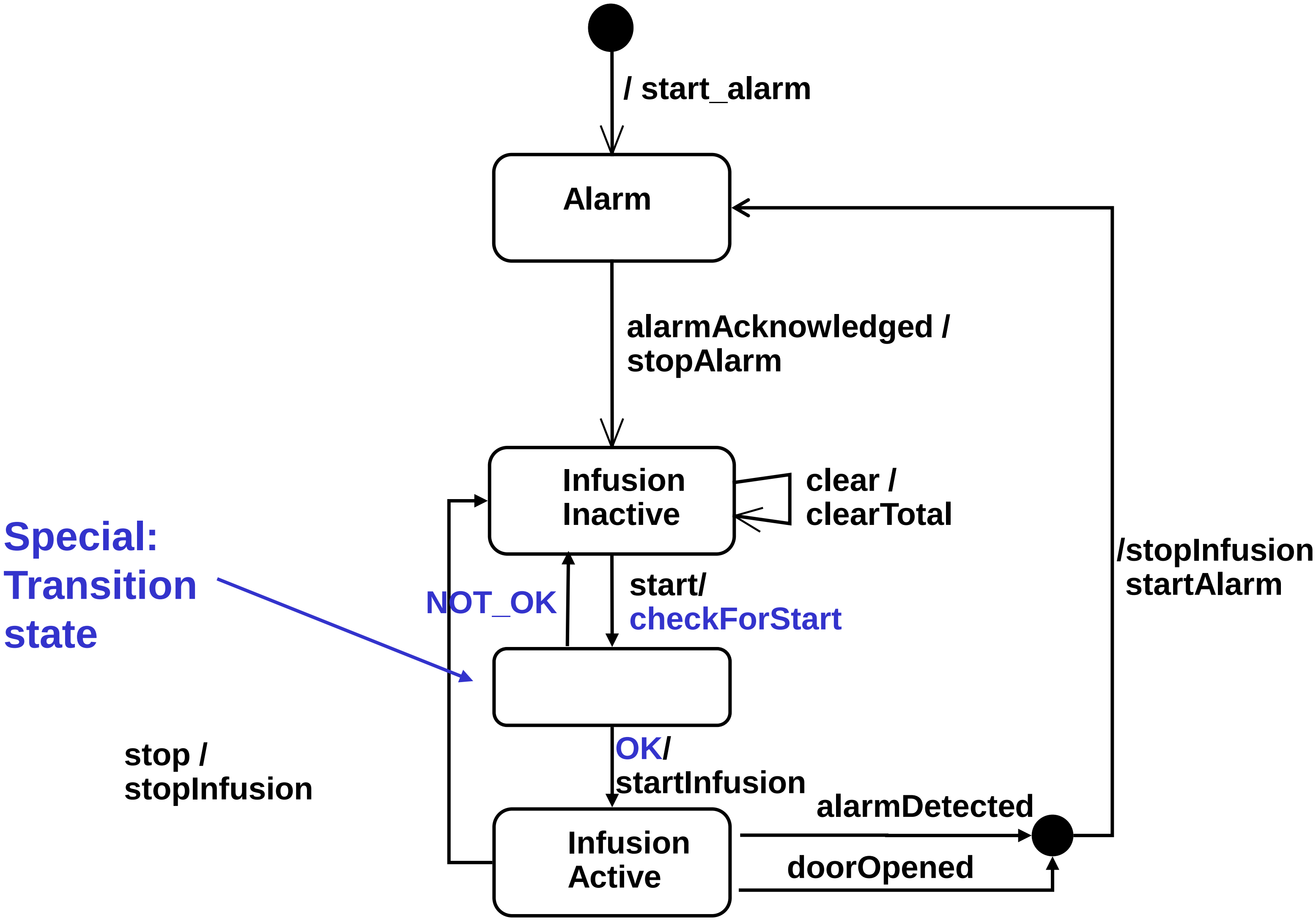
State-Event Table Implementation (7)

Example with guarded transition

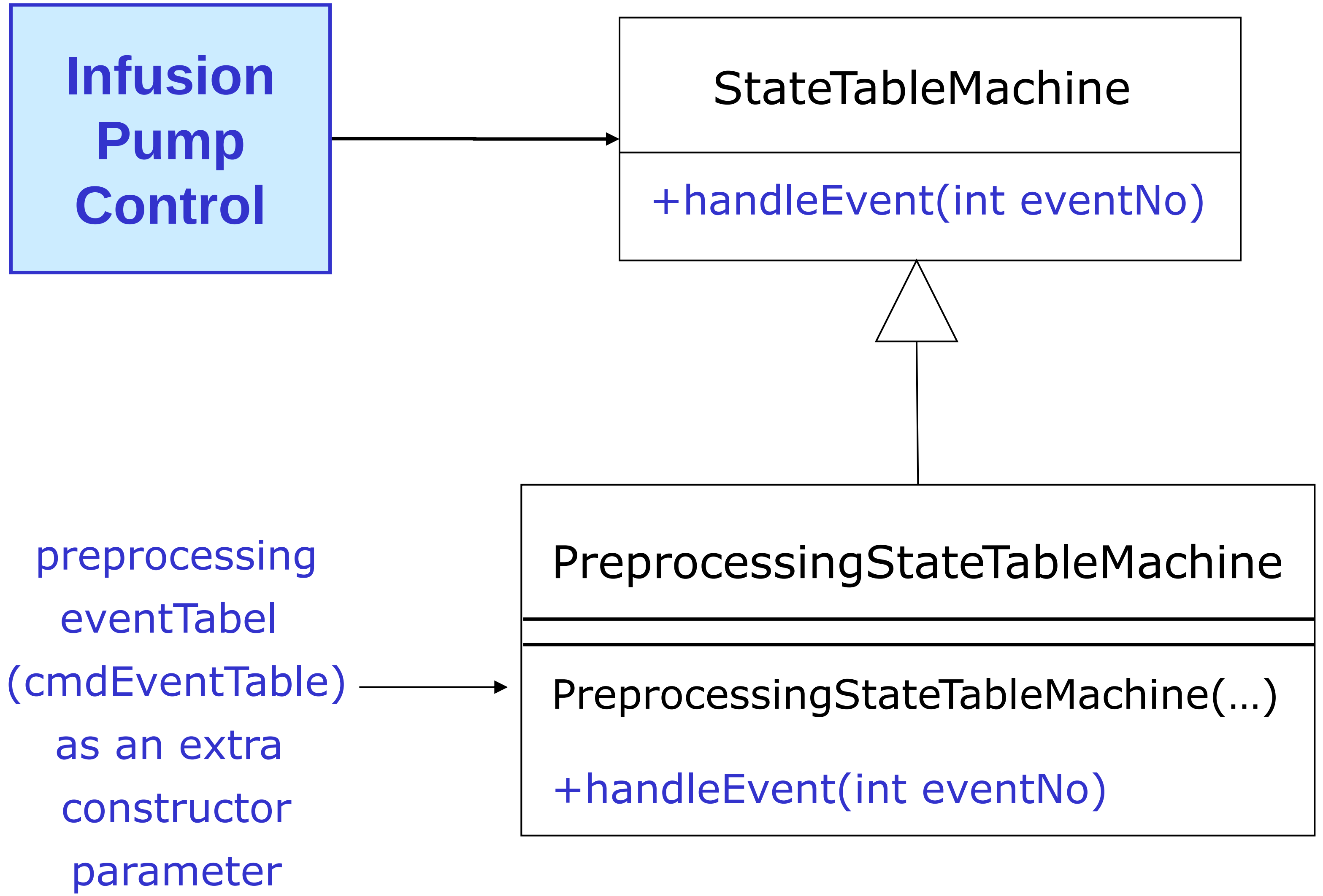
internal events

	start	stop	alarm Acknowled.	alarm Det.+xx	clear	OK	NOT_ OK
ALARM	X	X	stopAlarm INFUSION INACTIVE	X	X	X	X
INFUSION INACTIVE	check_ for_start TRANSIT. STATE	X	X	X	clear Total X	X	X
TRANSI- TION- STATE	X	X	X	X	X	start Infusion INFUSION ACTIVE	X INFUSION INACTIVE
INFUSION ACTIVE	X	stop Infusion INFUSION INACTIVE	X	stop Infusion startAl. ALARM	X	X	X

State-Event Table Implementation (8)



Example with a Specialized State M. (1)



Example with a Specialized State M. (2)

```
static int[] cmdEventTable =
  {
    START,                E0,
    STOP,                 E1,
    ALARM_ACKNOWLEDGED,   E2,
    ALARM_DETECTED,       E3,
    DOOR_OPENED,          E3,
    CLEAR,                E4,
    OK,                   E5,
    NOT_OK,               E6,
    END_OF_TABLE,         NO_EVENT,
  };

  InfusionPumpControl::InfusionPumpControl()
  {
   // call of constructor with cmdEventTable
    pSTM = new PreProcessingStateTableMachine(
        stateTable,NoOfEvents,ALARM,cmdEventTable);
  }
```

Example with a Specialized State M. (3)

```
int PreProcessingStateTableMachine::
            handleEvent(int actualEventNo)
{
    int i=0;
    while ((cmdEvent[i] != END_OF_TABLE) &&
          (cmdEvent[i] != actualEventNo))
       i += 2;
    if (cmdEvent[i] != END_OF_TABLE)
       return StateTableMachine::handleEvent(
                                  cmdEvent[i+1]);
    else
       return NO;  // NO= end of table
  }
```

Example with a Specialized State M. (4)

```
int InfusionPumpControl::executeAction(int actionNo)
  {
    switch (actionNo)
    {

      . . .
      case A6:          // check_for_start
        return (doorClosed() ? OK : NOT_OK); break;
     default:
    }
    return NO_EVENT;
  }

void InfusionPumpControl:: handleExternalEvent(int inEvent)
  {
    while (inEvent != NO_EVENT)
      inEvent = executeAction(pSTM->handleEvent(inEvent));
  }
```

Discussion of State-Event Table Implementation

- Disadvantages
  – The events should be preprocessed, to consecutive constants, before table lookup
  – Guarded transitions requires extra transition state plus internal events
  – Difficult to implement hierarchic state machines
- Advantages
  – if the state machine is specified as a state-event table:
    - Easy to implement and verify
  – The StateTableMachine can be a general utility class
- **Recommendations:**
  – Use it for larger and flat state machines, with no or only few guarded transition

3. State Pattern (GoF) Implementation

See separate
GoF State Pattern Slide Presentation

Case tool STM Generations

- A few tools can automatically generate code for state machines:
  - Visual State (IAR)
  - Rhapsody (IBM)
    - Code generation for two different strategies supported (switch based or state pattern based)

Summary

- Presented an example and different state machine implementations
  - Switch based
  - Table based
  - State Pattern based (separate slides)
  - Tool generated implementation