Content(2)

- **Object-oriented Software Development Methodology**
  - Outline of Unified Process and Use-case Driven Approach
  - Elevator Control System:
    Problem Description and Use-case Model
  - Elevator Control System:
    Finding of Problem Domain Objects
  - Elevator Control System:
    Sub-System Design and Task Design
  - Elevator Control System:
    Performance Evaluation
- **Product Line Technology**
  - Feature modeling
- **Aspect Oriented Software Design**
- **Contribution of OOT in Software Engineering**
  - History of SE Technologies and Contribution of OOT in SE field

Elevator Control System

Koichiro Ochimizu

School of Information Science
Japan Advanced Institute of Science and Technology

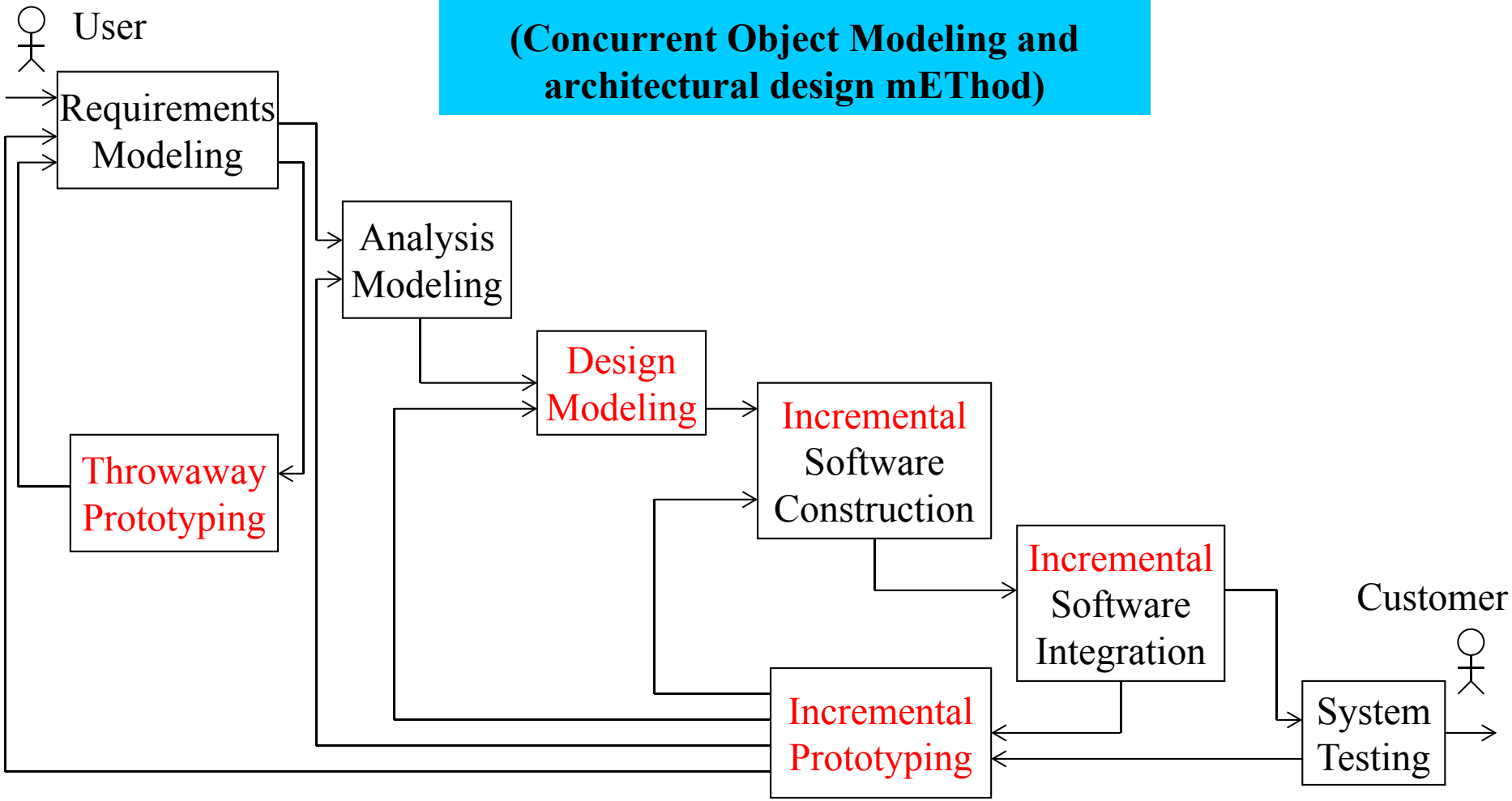
CASE STUDY

- *Elevator Control System*
- *Banking System*
- *Cruise Control and Monitoring System*
- *Distributed Factory Automation System*
- *Electronic Commerce System*

Hassan Gomaa, “Designing Concurrent, Distributed, and Real-Time Applications with UML”, Addison-Wesley, Object Technology Series, 2000.

COMET

(Concurrent Object Modeling and architectural design mEThod)



Activities of each Phase(1/2)

- **Requirements Modeling** functional requirements defined by actors and use cases
- **Analysis Modeling**

  Static model: structural relationships among problem domain classes depicted on class diagrams,

  Dynamic model: objects and their interactions depicted on either communication diagrams or sequence diagrams.

  The state-dependent aspects of the system are defined using hierarchical statecharts (finite state machines).
- **Design Modeling** The software architecture of the system is designed. The analysis model is mapped to an operational environment. Subsystem structuring criteria are provided.

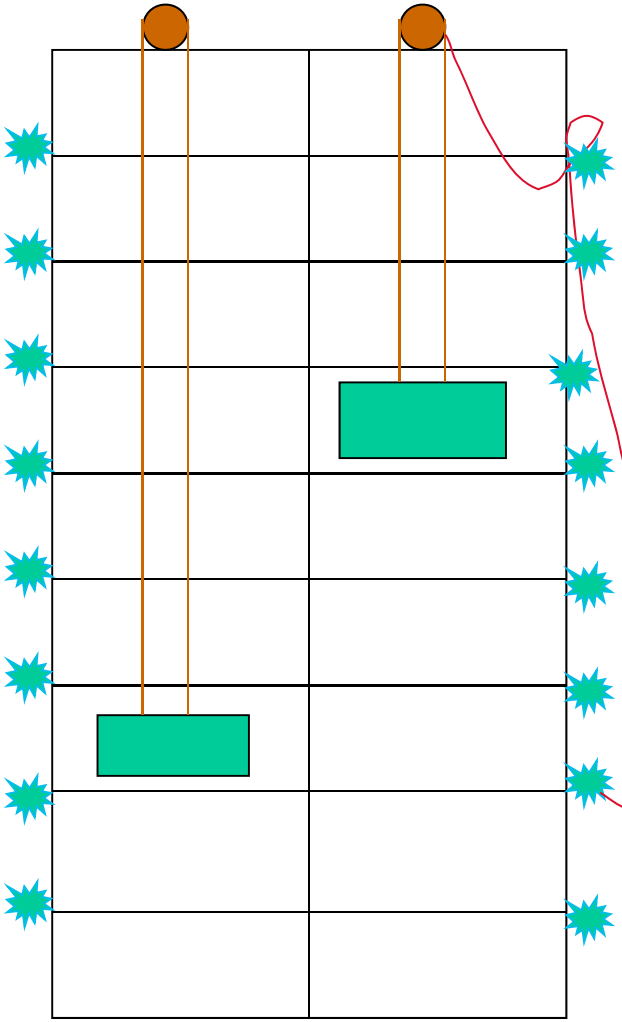
Activities of each Phase(2/2)

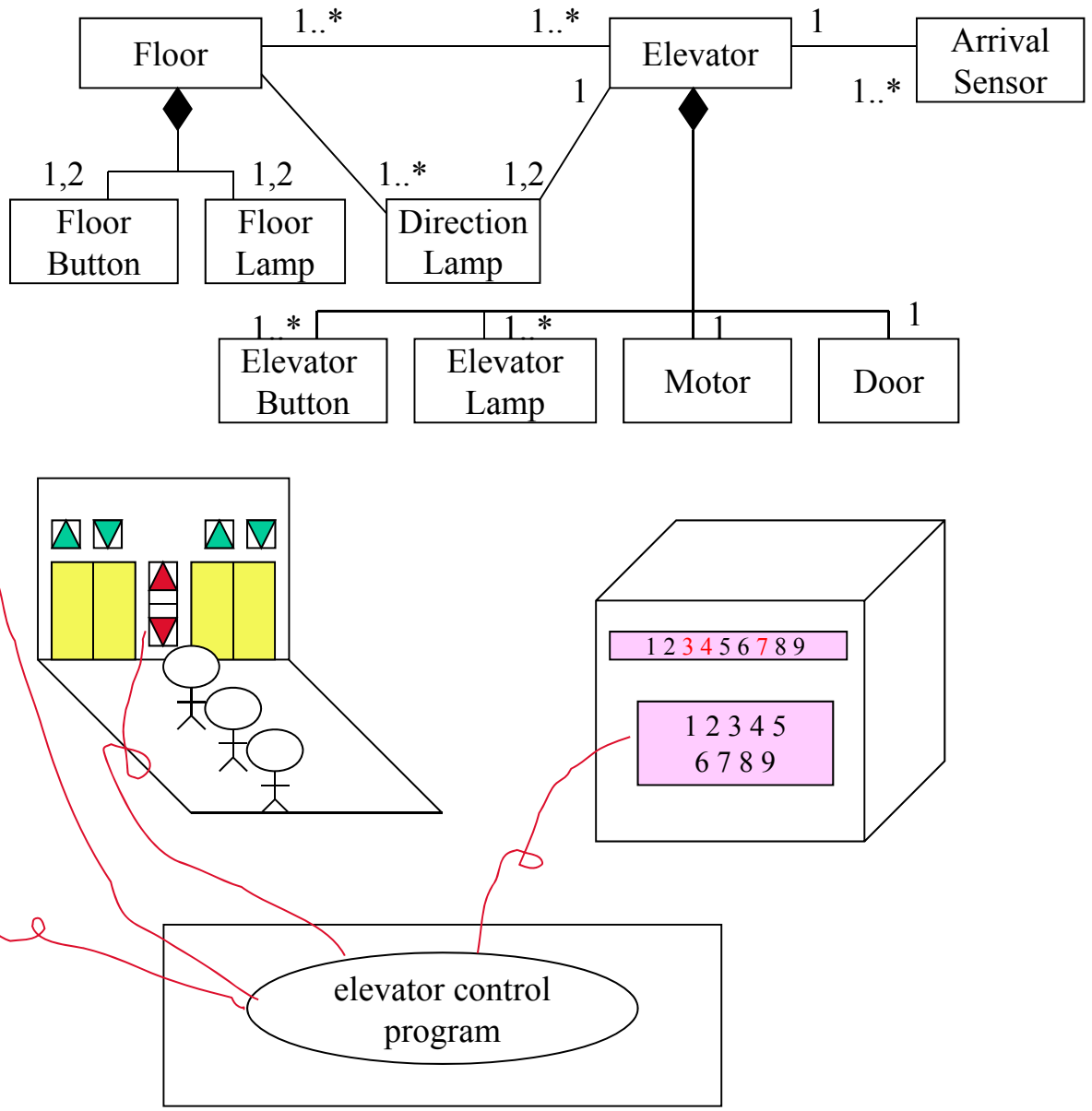
- **Incremental Software Construction** Selecting a subset of the system to be constructed for each increment. The subset is determined by choosing the use cases to be Included in this increment. Incremental software construction consists of the detailed design, coding, and unit testing of the classes in the subset.
- **Incremental Software Integration** the integration testing of each increment is performed. Integration test cases are developed for each use case. Interfaces between the objects that participates in each use case are tested.
- **System Testing** testing the system against Its functional requirements.

Problem Description

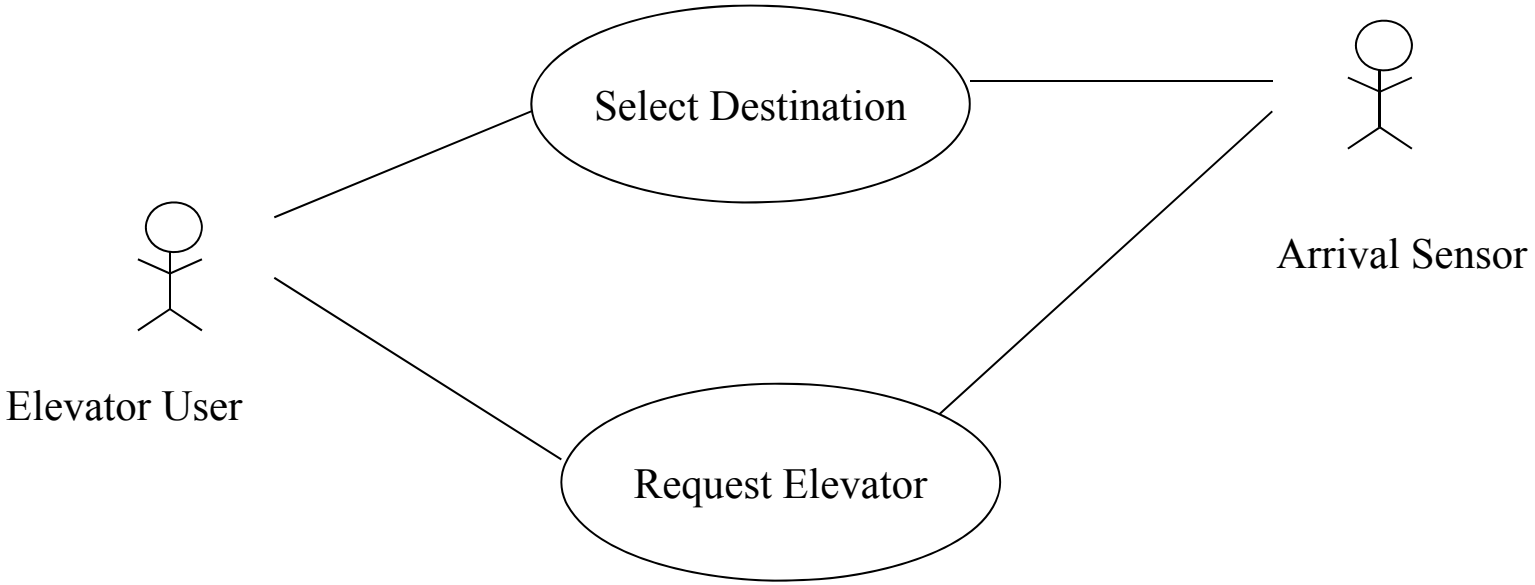
- Definition of Functional Requirements by Use Cases

Problem Description





Use Case Model



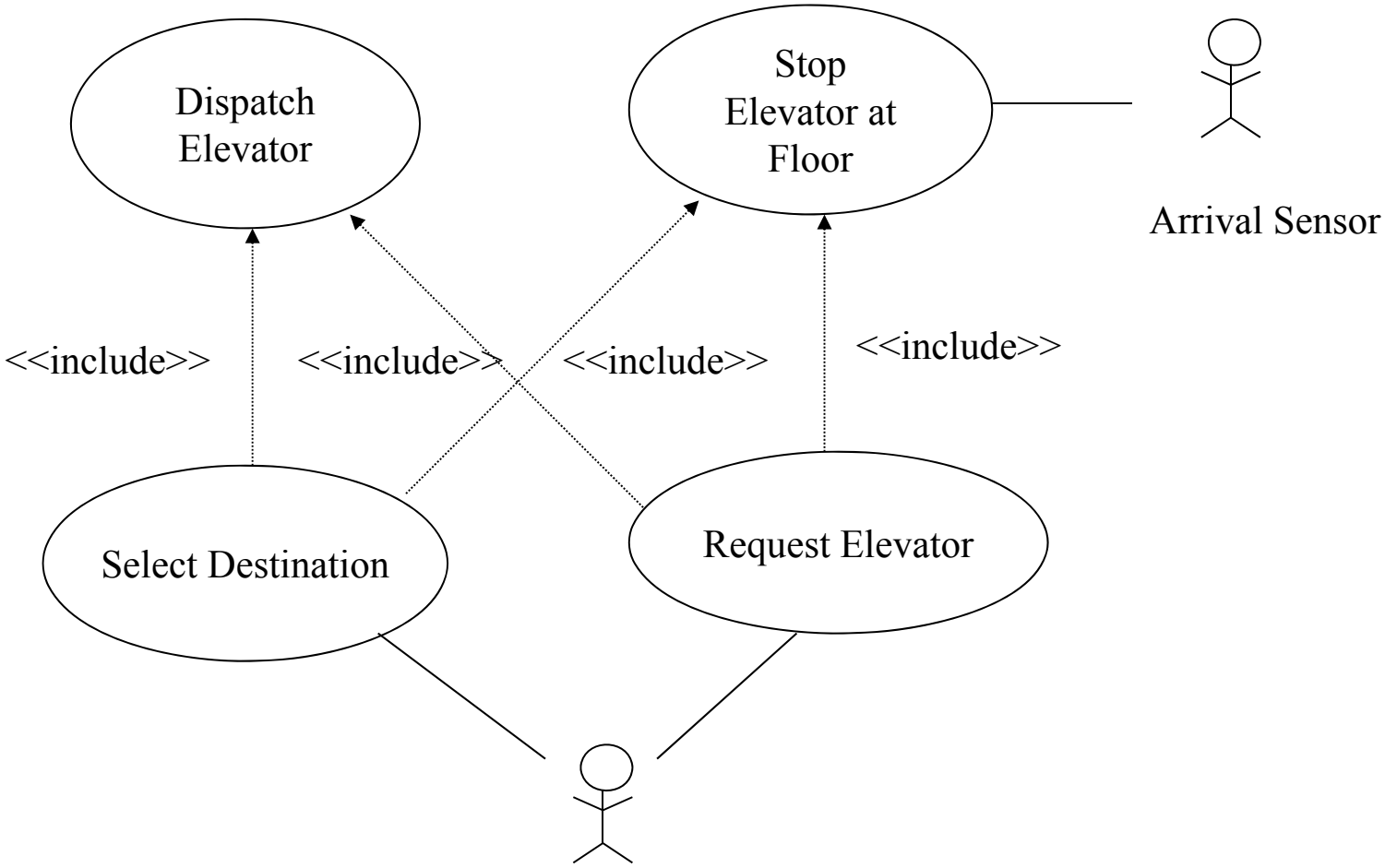
Use Case Description

- **Use Case name:** Select Destination
- **Summary:** The user in the elevator presses an up or down elevator button to select a destination floor to which move
- **Dependency**
- **Actor:** Elevator User (primary), Arrival Sensor
- **Precondition:** User in the elevator
- **Description:**
  - 1. User presses an up elevator button. The elevator button sensor sends the elevator button request to the system, identifying the destination floor the user wishes to visit.
  - 2. The new request is added to the list of floors to visit. If the elevator is stationary, The system determines in which direction the system should move in order to service the next request. The system commands the elevator door to close. When the door has closed, the system commands the motor to start moving the elevator, either up or down.
  - 3. As the elevator moves between floors, the arrival sensor detects that the elevator is approaching a floor and notifies the system. The system checks whether the elevator should stop at this floor. If so, the system commands the motor to stop. When the elevator has stopped, the system commands the elevator door to open.
  - If there are other outstanding requests, the elevator visits these floors on the way to the floor requested by the user. Eventually, the elevator arrives at the destination floor selected by the user.
- **Alternatives:**
  - 1. User presses down elevator button to move down. System response is the same as for the main sequence.
  - 2. If the elevator is at a floor and there is no new floor to move to, the elevator stays at the current floor, with the door open.
- **Postcondition:** Elevator has arrived at the destination floor selected by the user.

Use Case Description

- **Use Case name:** Request Elevator
- **Summary:** The user at a floor presses an up or down floor button to request an elevator.
- **Dependency**
- **Actor:** Elevator User (primary), Arrival Sensor
- **Precondition:** User is at a floor and wants to an elevator
- **Description:**
  - 1. User presses an up floor button. The floor button sensor sends the user request to the system, identifying the floor number.
  - 2. The system selects an elevator to visit this floor. The new request is added to the list of floors to visit. If the elevator is stationary, The system determines in which direction the system should move in order to service the next request. The system commands the elevator door to close. After the door has closed, the system commands the motor to start moving the elevator, either up or down.
  - 3. As the elevator moves between floors, the arrival sensor detects that the elevator is approaching a floor and notifies the system. The system checks whether the elevator should stop at this floor. If so, the system commands the motor to stop. When the elevator has stopped, the system commands the elevator door to open.
  - If there are other outstanding requests, the elevator visits these floors on the way to the floor requested by the user. Eventually, the elevator arrives at the floor in response to the user request.
- **Alternatives:**
  - 1. User presses floor button to move down. System response is the same as for the main sequence.
  - 2. If the elevator is at a floor and there is no new floor to move to, the elevator stays at the current floor, with the door open.
- **Postcondition:** Elevator has arrived at the floor in response to user request.

Use case Model



Use Case Description

- **Use Case name:** Select Destination
- **Summary:** The user in the elevator presses an up or down elevator button to select a destination floor to which move
- **Dependency**
- **Actor:** Elevator User
- **Precondition:** User in the elevator
- **Description:**
  - 1. User presses an up elevator button. The elevator button sensor sends the elevator button request to the system, identifying the destination floor the user wishes to visit.
  - 2. The new request is added to the list of floors to visit. If the elevator is stationary,

    Include Dispatch Elevator abstract use.
  - 3. Include Stop Elevator at Floor abstract use case.
  - If there are other outstanding requests, the elevator visits these floors on the way to the floor requested by the user, following the above sequence of dispatching and stopping. Eventually, the elevator arrives at the destination floor selected by the user.
- **Alternatives:**
  - 1. User presses down elevator button to move down. System response is the same as for the main sequence.
  - 2. If the elevator is at a floor and there is no new floor to move to, the elevator stays at the current floor, with the door open.
- **Postcondition:** Elevator has arrived at the destination floor selected by the user.

Use Case Description

- **Use Case name:** Request Elevator
- **Summary:** The user at a floor presses an up or down floor button to request an elevator.
- **Dependency**
- **Actor:** Elevator User
- **Precondition:** User is at a floor and wants to an elevator
- **Description:**
  - 1. User presses an up floor button. The floor button sensor sends the user request to the system, identifying the floor number.
  - 2. The system selects an elevator to visit this floor. The new request is added to the list of floors to visit. If the elevator is stationary, then include Dispatch Elevator abstract use case.
  - 3. Include Stop Elevator at Floor abstract use case.
  - If there are other outstanding requests, the elevator visits these floors on the way to the floor requested by the user following the above sequence of dispatching and stopping. Eventually, the elevator arrives at the floor in response to the user request.
- **Alternatives:**
  - 1. User presses floor button to move down. System response is the same as for the main sequence.
  - 2. If the elevator is at a floor and there is no new floor to move to, the elevator stays at the current floor, with the door open.
- **Postcondition:** Elevator has arrived at the floor in response to user request.

Use Case Description

- **Use Case name:** Stop Elevator at Floor abstract Use Case
- **Summary:**
- **Dependency**
- **Actor:** Arrival Sensor
- **Precondition:** Elevator is moving
- **Description:**
  - As the elevator moves between floors, the arrival sensor detects that the elevator is approaching a floor and notifies the system. The system checks whether the elevator should stop at this floor. If so, the system commands the motor to stop. When the elevator has stopped, the system commands the elevator door to open.
- **Alternatives:**
  - The elevator is not required to stop at this floor and so continues past the floor.
- **Postcondition:** Elevator has stopped at floor, with door open.

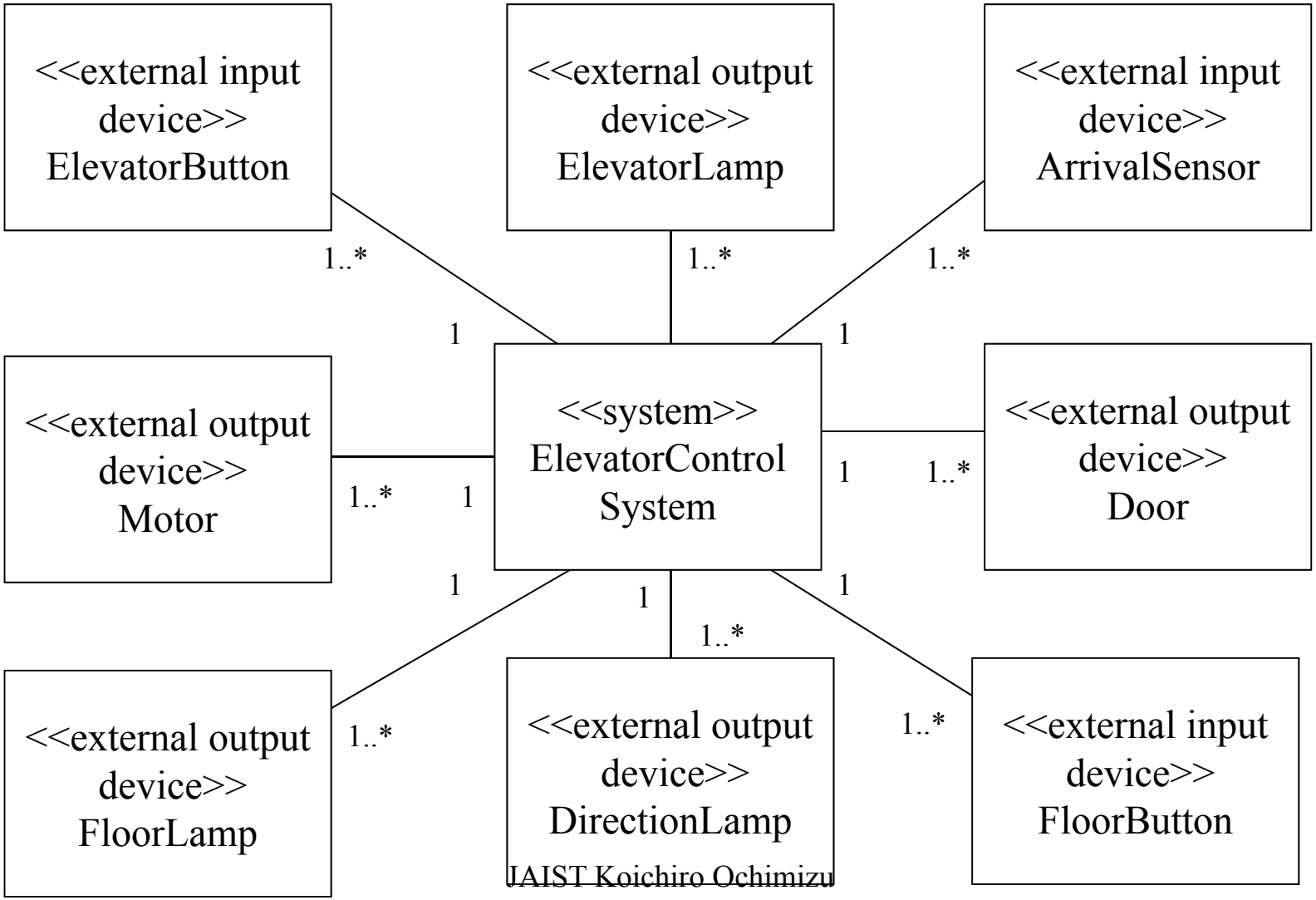
Use Case Description

- **Use Case name:** Dispatch Elevator abstract use case
- **Summary:**
- **Dependency**
- **Actor:**
- **Precondition:** Elevator is at a floor with the door open.
- **Description:**
  - The system determines in which direction the system should move in order to service the next request. The system commands the elevator door to close. After the door has closed, the system commands the motor to start moving the elevator, either up or down.
- **Alternatives:**
  - If the elevator is at a floor and there is no new floor to move to, the elevator stays at the current floor, with the door open.
- **Postcondition:** Elevator is moving in the commanded direction.

Content(2)

- **Object-oriented Software Development Methodology**
  - Outline of Unified Process and Use-case Driven Approach
  - Elevator Control System:
    Problem Description and Use-case Model
  - Elevator Control System:
    Finding of Problem Domain Objects
  - Elevator Control System:
    Sub-System Design and Task Design
  - Elevator Control System:
    Performance Evaluation
- **Product Line Technology**
  - Feature modeling
- **Aspect Oriented Software Design**
- **Contribution of OOT in Software Engineering**
  - History of SE Technologies and Contribution of OOT in SE field

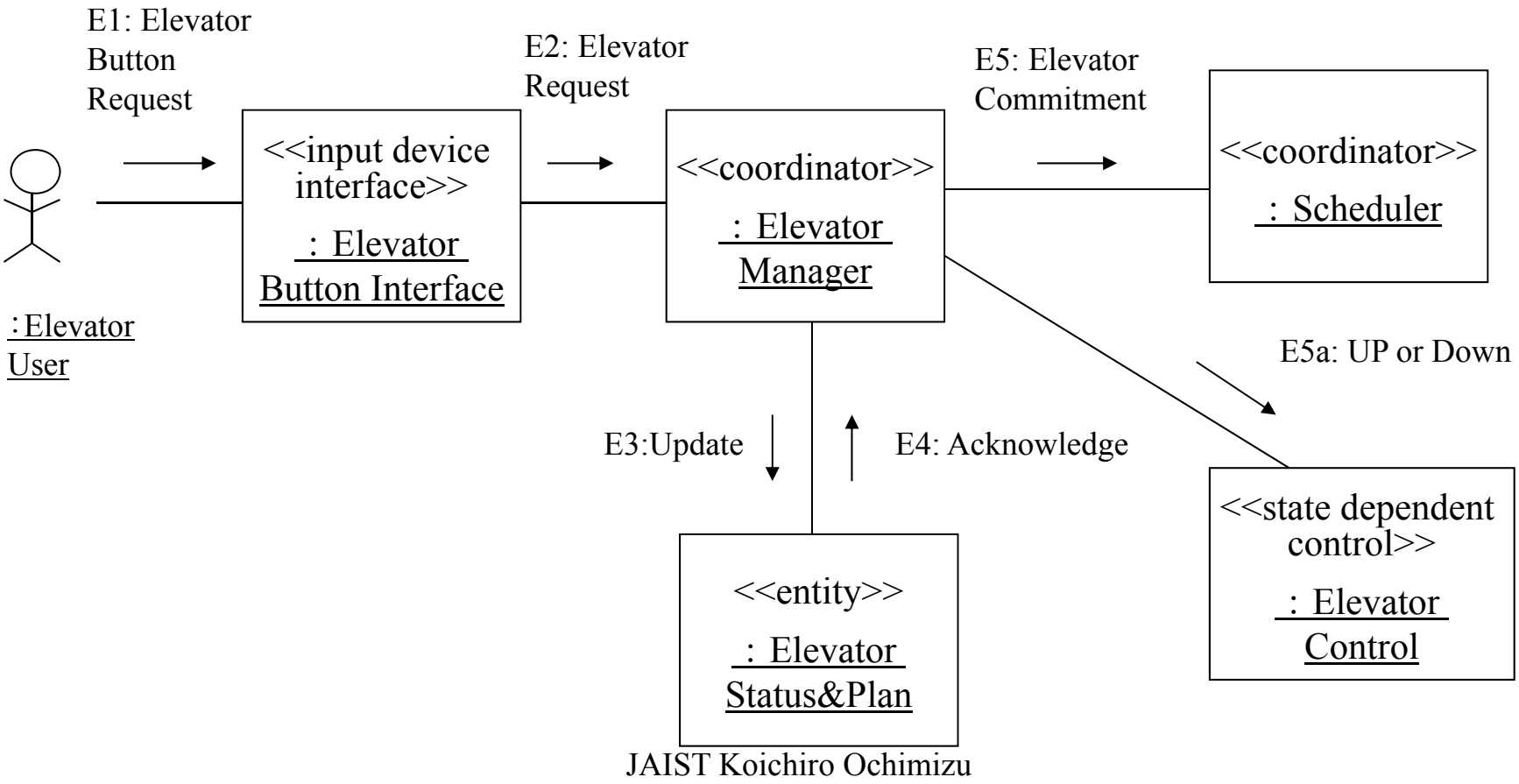
Context Class Diagram



Finding Analysis objects

- For every <<external input/output device>> object, there is a corresponding software device interface object.
- Each elevator has a state-dependent control object called Elevator Control, which control the elevator motor and door.
- Because requests for the elevator can come at any time, a decision is made to have a separate coordinator object, called the Elevator Manager, to receive all incoming requests for the elevator and to update the elevator plan.
- An entity object is needed for each elevator, which we call Elevator Status & Plan.

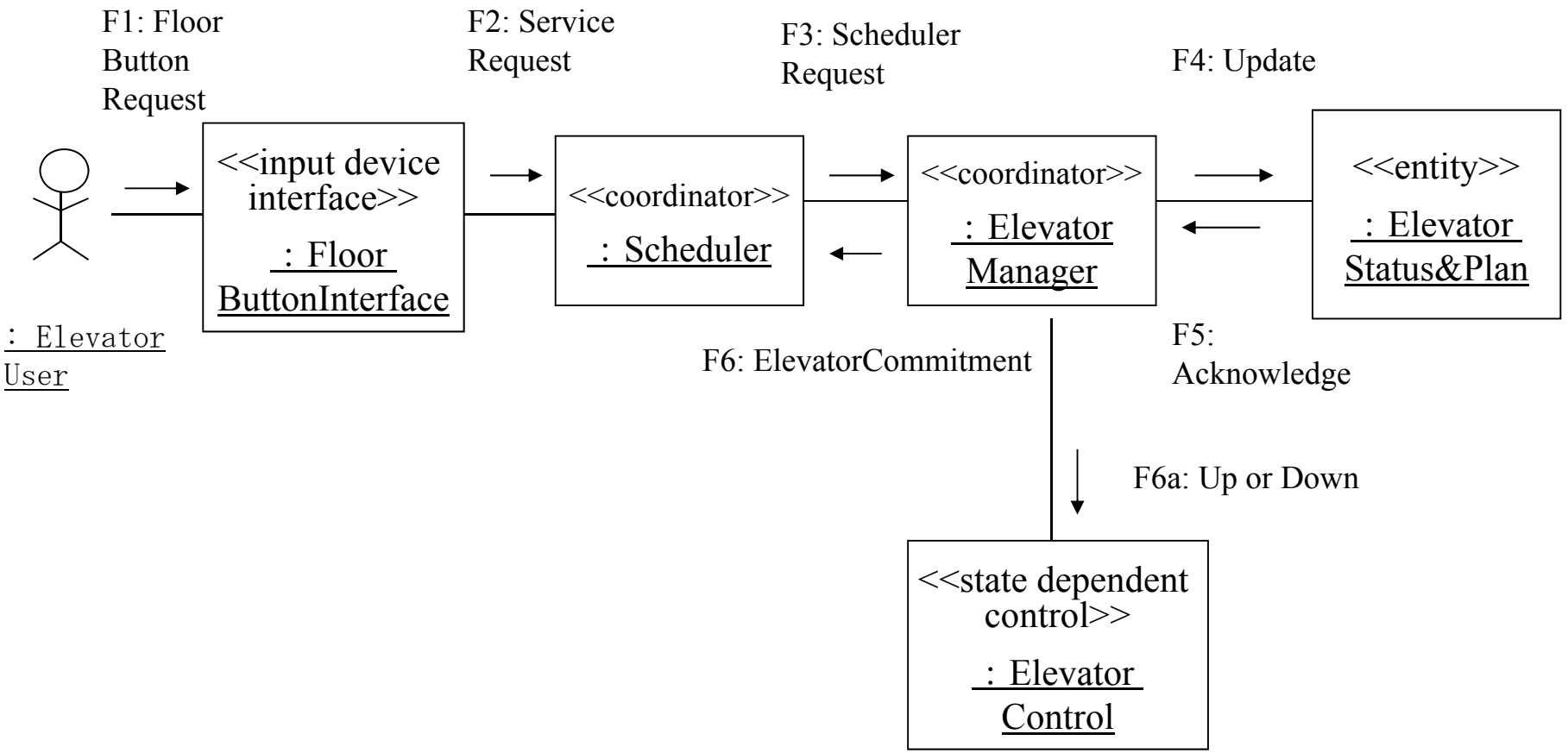
Collaboration Diagram for Select Destination use case



The message sequence description

- E1: The Elevator Button Request arrives at the Elevator Button Interface object.
- E2: The Elevator Button Interface object sends the Elevator Request to the Elevator Manager object.
- E3: The Elevator Manager sends the request to Elevator Status & Plan, which adds the request to the list of floors to be visited.
- E4: The elevator plan is updated. An acknowledgement is returned to the Elevator Manager object, which identifies whether the elevator is idle.
- E5: The Elevator Manager sends an Elevator Commitment message to the Scheduler, to inform it that this elevator is committed to visit the given floor.
- E5a: If the Elevator is idle, the Elevator Manager sends an Up (or Down) message to the Elevator Control object, directing it to move in the desired direction. This case is handled by the Dispatch Elevator use case.

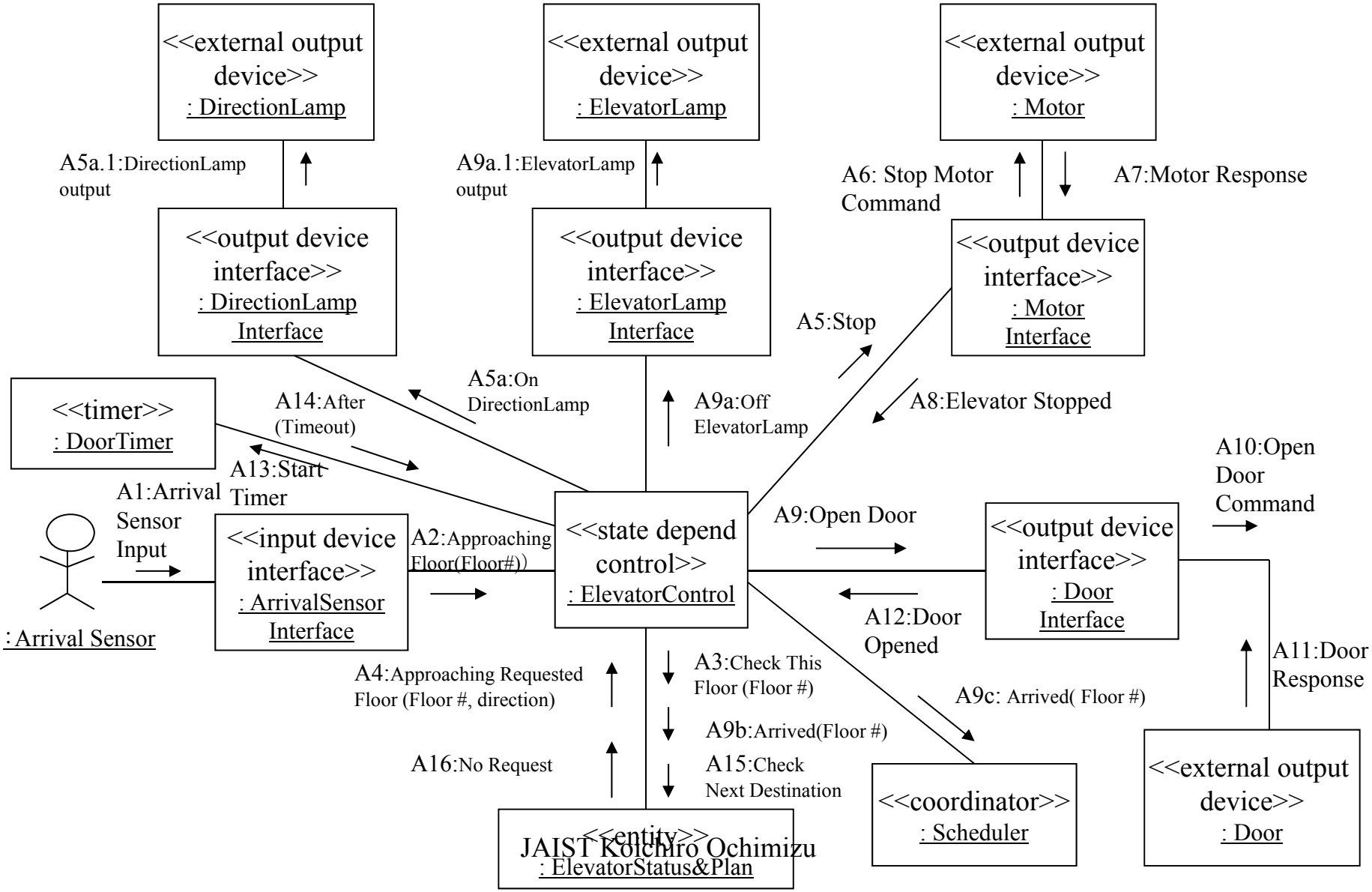
Collaboration Diagram for Request Elevator use case



The message sequence description

- F1: The Floor Button Request arrives at the Floor Button Interface object.
- F2: The Floor Button Interface object sends a Service Request to the Scheduler object.
- F3: The Scheduler object selects an elevator and sends a Scheduler Request to the Elevator Manager object in the selected Elevator composite object.
- F4: The Elevator Manager object sends an Update message to the Elevator Status & Plan to add the new request to the elevator plan of which floor it is to visit.
- F5: An acknowledgement is returned to the Elevator Manager object, which identifies whether the elevator is idle.
- F6: The Elevator Manager sends an Elevator Commitment message to the Scheduler.
- F6a: If the elevator is idle, the Elevator Manager sends an Up (or Down) message to the Elevator Control object, directing it to move in the desired direction. This case is handled by the Dispatch Elevator use case.

Collaboration Diagram for Stop Elevator use case



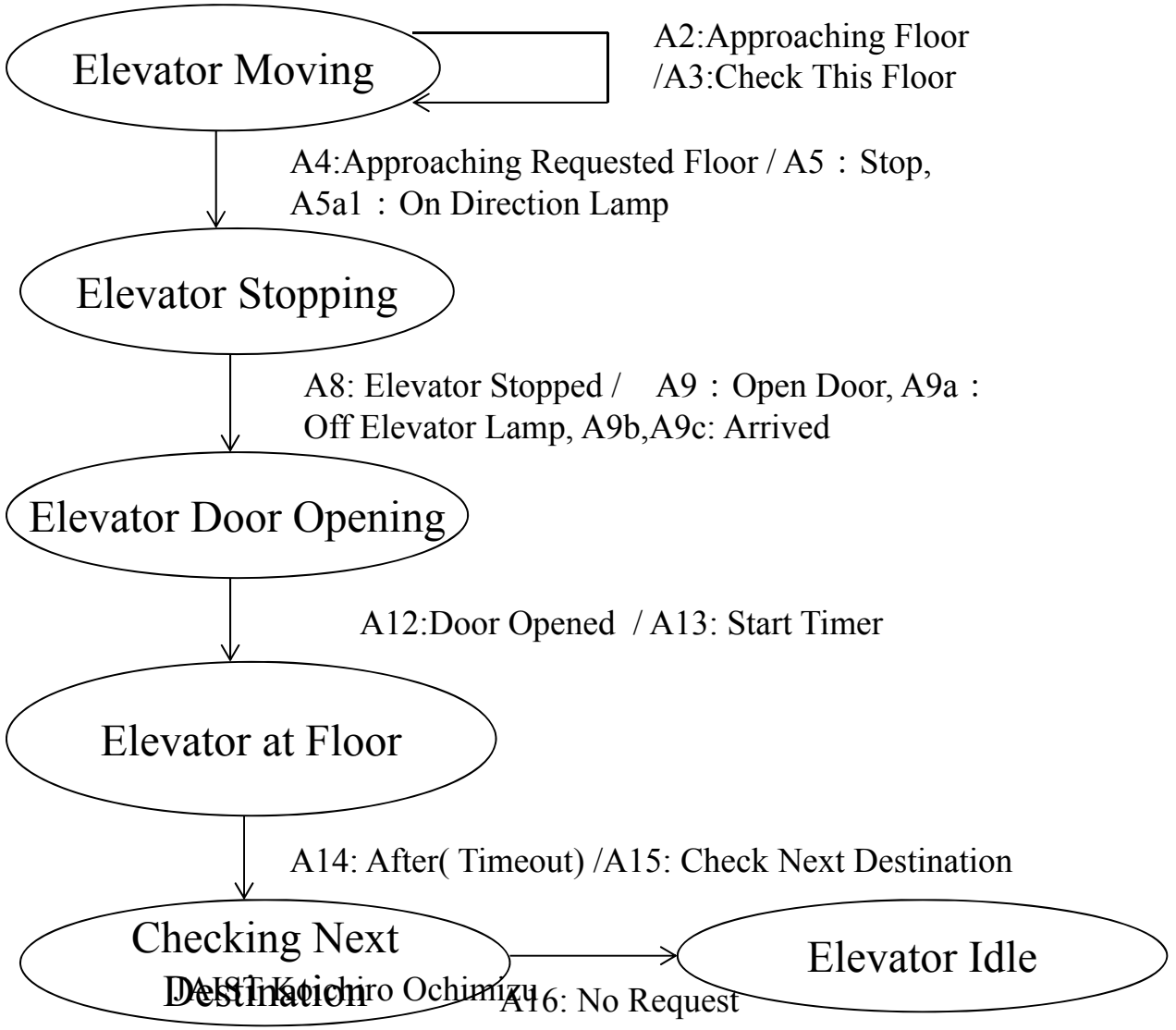
The message sequence description

- A1: The Arrival Sensor Interface object receives an input from the arrival sensor external entity.
- A2: The Arrival Sensor Interface object sends the floor number in the Approaching Floor message to the Elevator Control object.
- A3: The Elevator Control object sends a Check This Floor message to the Elevator Status & Plan object, which checks whether the floor at which the elevator is arriving is one where it should stop.
- A4: As the elevator is arriving at a requested floor, the Elevator Status & Plan object sends the Approaching Requested Floor message to the Elevator Control object. The message contains the floor number and the future direction. On receiving this message, Elevator Control transitions from Elevator Moving state to Elevator Stopping state.
- A5: As a result of the transition to Elevator Stopping state, the Elevator Control object commands the Motor Interface object to Stop.
- A5a(parallel sequence): Elevator Control sends an On Direction Lamp (with up or down as a parameter) to the Direction Lamp Interface object, which switches on the real-world direction lamp(A5a.1).
- A6: The Motor Interface object sends the Stop Motor Command to the real-world motor.
- A7: The Motor Interface object receives the Motor Response.
- A8: Motor Interface object sends an Elevator Stopped message to the Elevator Control object, which then transitions to Elevator Door opening state.

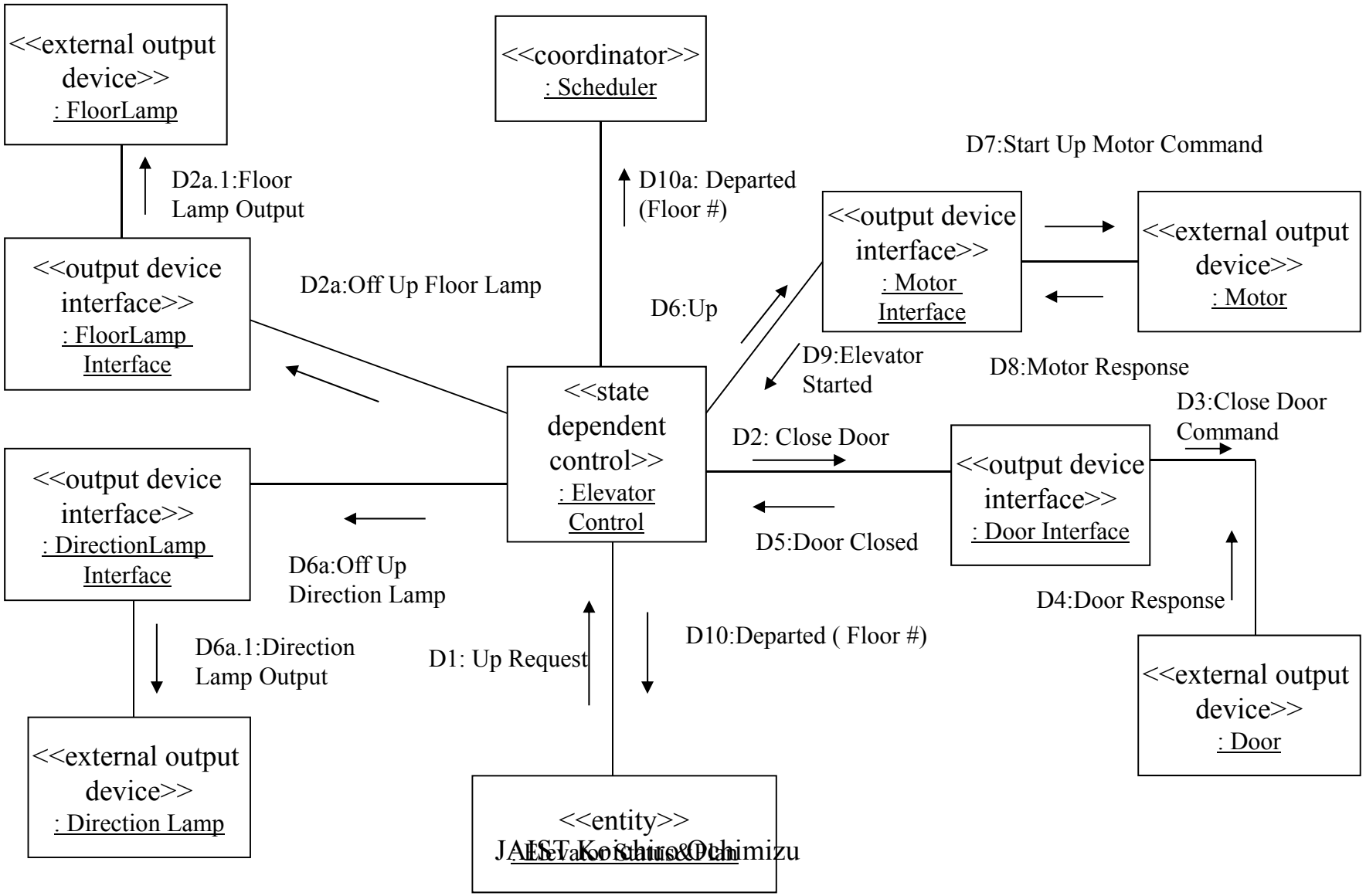
The message sequence description

- A9: On transitioning to Elevator Door Opening state, the Elevator Control object sends the Door Interface object a command to Open Door.
- A9a(parallel sequence because there are four actions associated with the state transition): The Elevator Control object sends an Off Elevator Lamp message to the Elevator Lamp Interface object, which then sends an Elevator Lamp Output to the external lamp to switch it off(A9a.1).The Elevator Control object sends the Arrived message to both the Elevator Status & Plan object(A9b, third parallel sequence) and the Scheduler object(A9c, Fourth parallel sequence).
- A10: The Door Interface object sends the Open Door Command to the real world door.
- A11: The Door Interface object receives the Door Response.
- A12: The Door Interface object sends a Door Opened message to the Elevator Control object, which then transitions to Elevator at Floor
- A13: The Elevator Control object starts a timer.
- A14: A timer event is generated after a period of time equal to timeout, causing the Elevator Control object to transition to Checking Next Destination state..
- A15: As a result of the transition, Elevator Control sends a Check Next Destination message to the Elevator Status & Plan object. The objective is to determine the next destination just prior to departure, in case there has been a recent update to the plan. If the elevator does not have any outstanding requests, it transitions to Elevator Idle state (event A16). Otherwise, use the Dispatch Elevator use

Stop Elevator at Floor use case: statechart for Elevator Control



Collaboration diagram for Dispatch Elevator use case



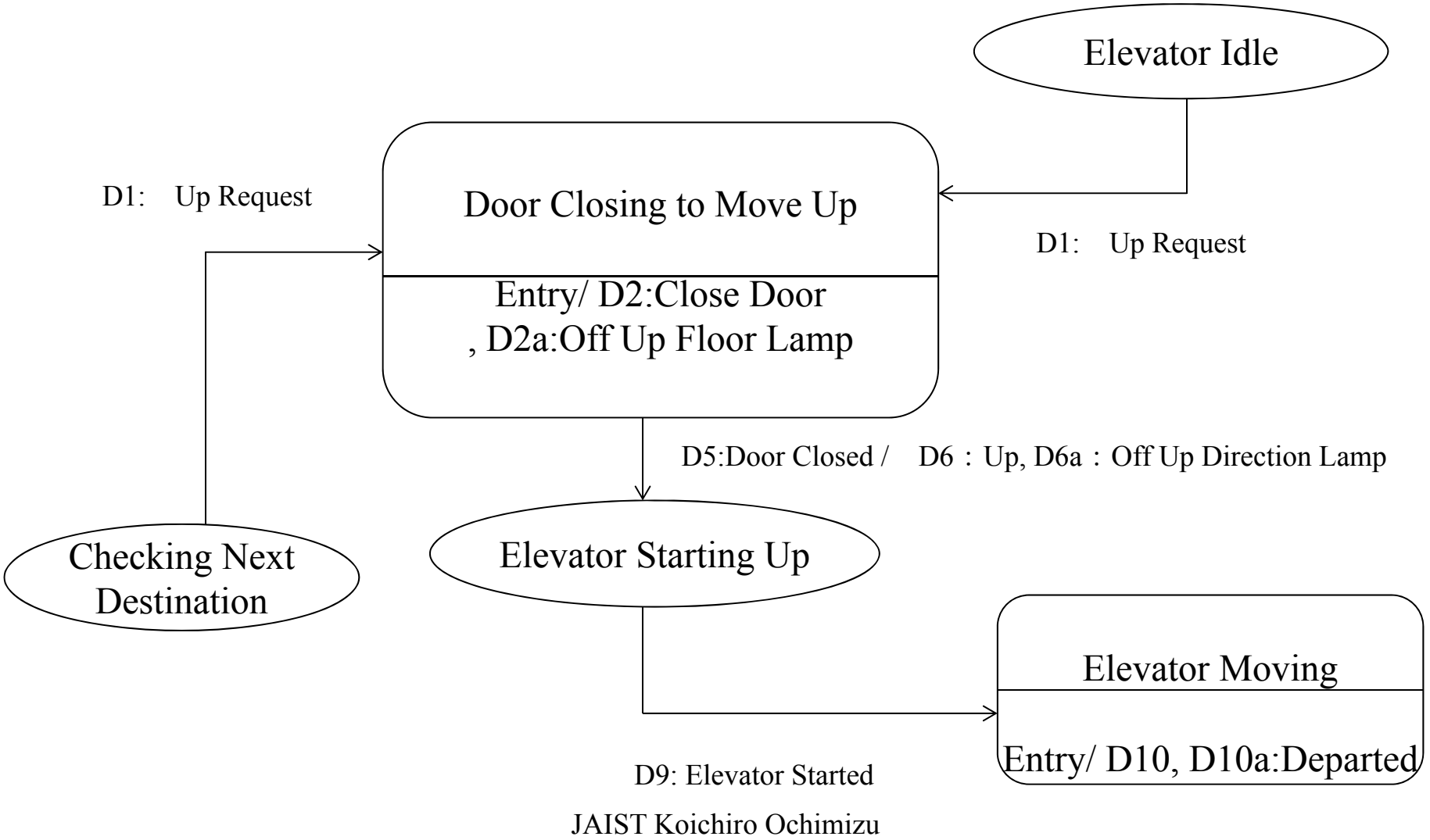
The message sequence description

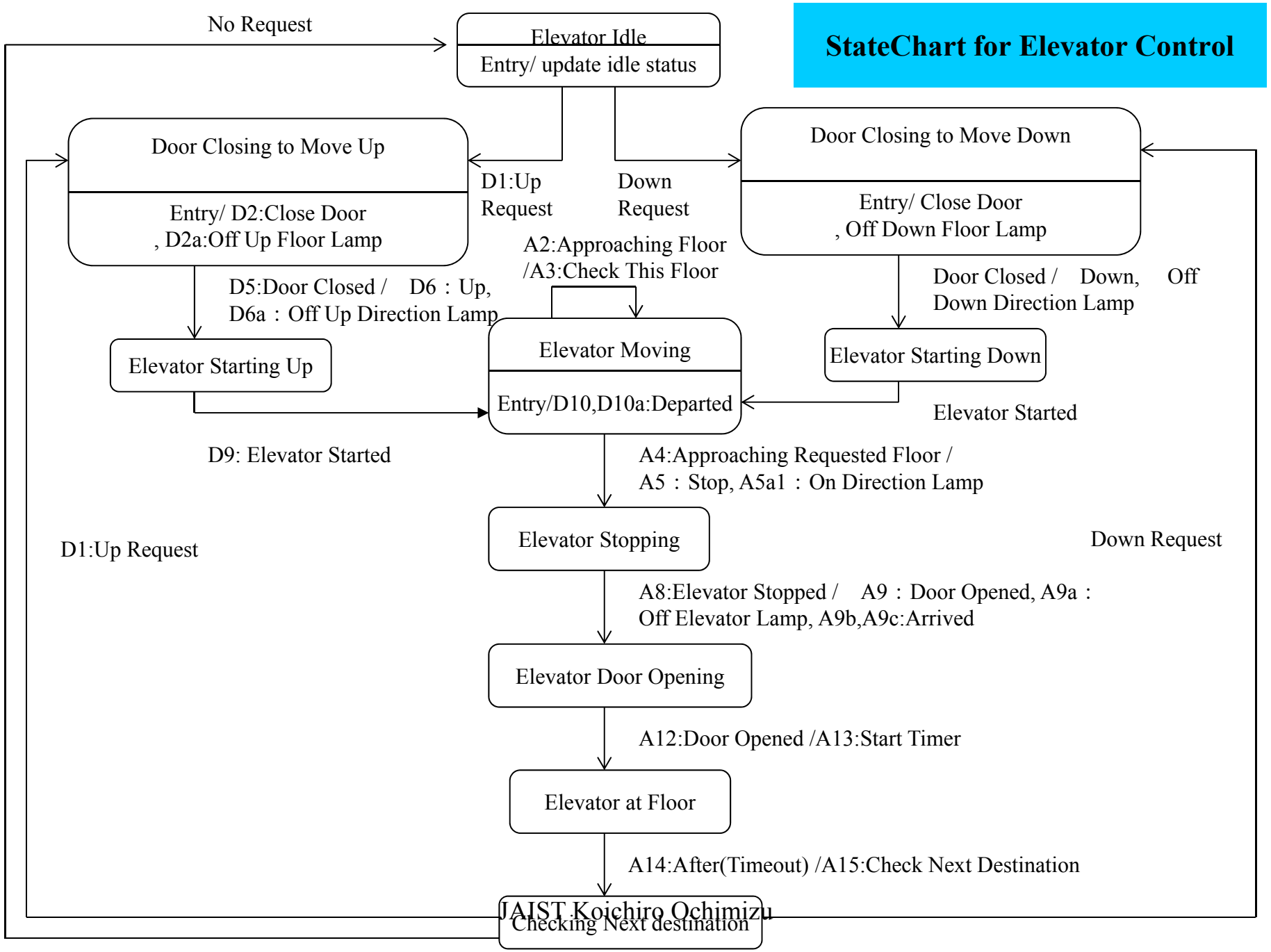
- Starting preconditions are different for Dispatch Elevator.
  - Stop Elevator at Floor is the first case. On entering Checking Next Destination state, Elevator Control sends a Check Next Destination message to Elevator Status & Plan . Elevator Status & Plan sends an Up Request (or Down Request) message to Elevator Control, informing it of the direction in which to move.
  - Elevator Control object is in Elevator Idle state is the second case. Elevator Manager receives a message from either the Scheduler or the Elevator Button Interface with a request for the elevator to visit the floor. Elevator Manager sends a message to Elevator Status & Plan to update the plan. If the elevator is busy servicing a request, Elevator Status & Plan returns an Acknowledgement message with a null parameter. On the other hand, if the elevator is idle, Elevator Status & Plan returns an Acknowledgement message with an up (or down) parameter.
- D1: {Source object} sends Elevator Control an Up Request message. Elevator Control transitions to Door Closing to Move Up state.
- D2: As a result of this state transition, there are two concurrent outputs events. Elevator Control sends a Close Door command to Door Interface. On the statechart, the Close door event (as well as one other output event) is shown as an entry action, because the Up Request event can arrive from either the Elevator Idle state or the Checking Next Destination state.
- D2a(parallel sequence): Elevator Control sends an Off Up Floor Lamp to the Floor Lamp Interface object, which switches off the real-world

The message sequence description

- D3: Door Interface sends a Close Door Command to the real – world door.
- D4: The real-world door sends a Door Response when the door is closed.
- D5: The Door Interface sends a Door Closed message to Elevator Control, which transitions to Elevator Starting Up state.
- D6: Elevator Control sends an Up Command to the Motor Interface object.
- D6a: Elevator Control sends an Off Up Direction Lamp request to the Direction Lamp Interface object, which switches off the direction lamp.
- D7: The Motor Interface object sends the Start Up Motor Command to the real-world motor.
- D8: The real-world motor sends a Motor Response when the elevator has started moving upward.
- D9: The Motor Interface object sends an Elevator Started message to Elevator Control, which transitions to Elevator Moving state.
- D10: Elevator Control sends a Departed message to both the Elevator Status & Plan and Scheduler object.

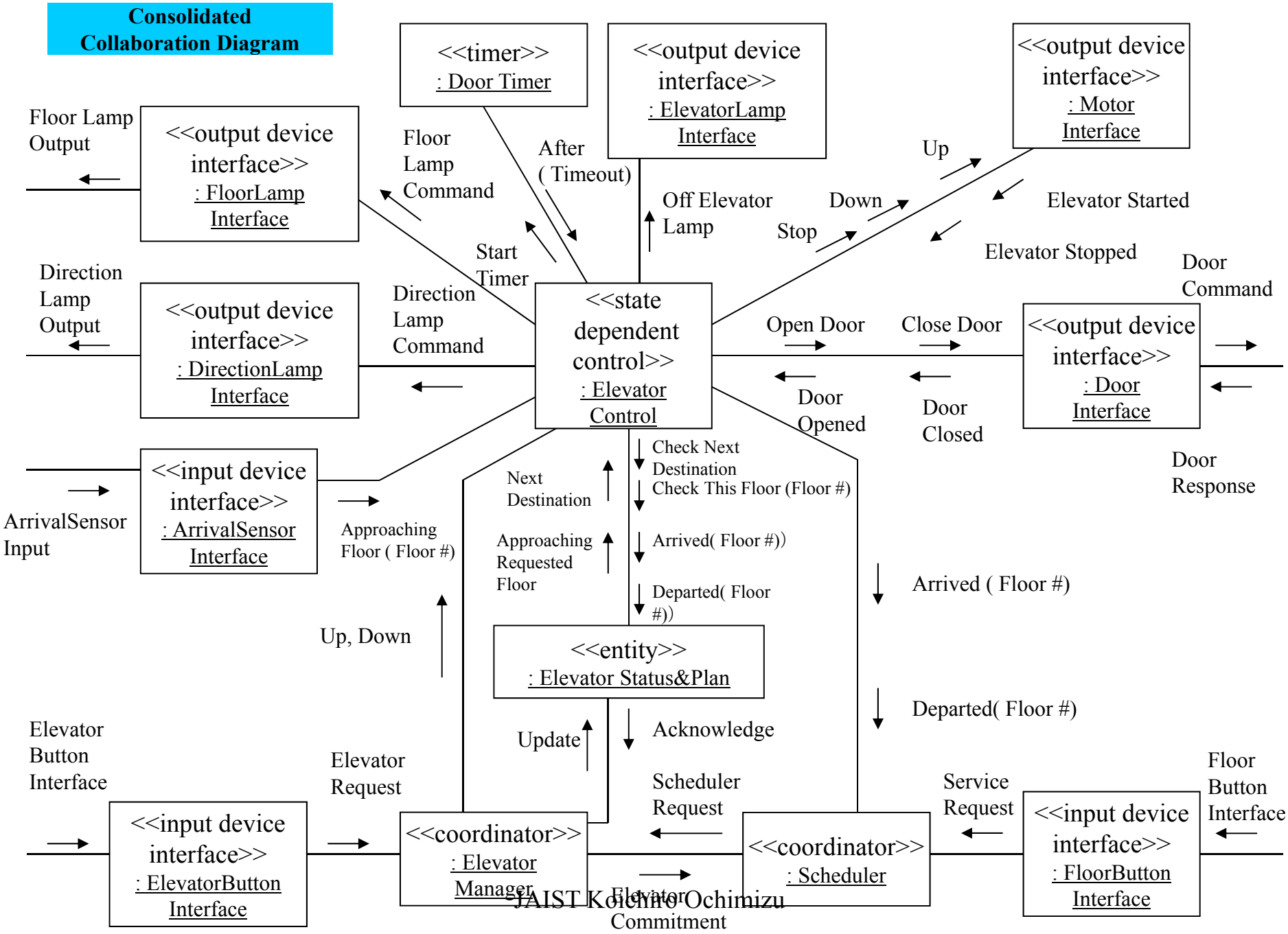
Dispatch Elevator use case: Statechart for Elevator Control




StateChart for Elevator Control
No Request
Elevator Idle
Entry/ update idle status
Door Closing to Move Up
Entry/ D2:Close Door
, D2a:Off Up Floor Lamp
D1:Up
Request
Down
Request
Door Closing to Move Down
Entry/ Close Door
, Off Down Floor Lamp
A2:Approaching Floor
/A3:Check This Floor
D5:Door Closed / D6 : Up,
D6a : Off Up Direction Lamp
Door Closed / Down, Off
Down Direction Lamp
Elevator Starting Up
Elevator Moving
Entry/D10,D10a:Departed
Elevator Starting Down
Elevator Started
D9: Elevator Started
A4:Approaching Requested Floor /
A5 : Stop, A5a1 : On Direction Lamp
Elevator Stopping
D1:Up Request
Down Request
A8:Elevator Stopped / A9 : Door Opened, A9a :
Off Elevator Lamp, A9b,A9c:Arrived
Elevator Door Opening
A12:Door Opened /A13:Start Timer
Elevator at Floor
A14:After(Timeout) /A15:Check Next Destination
Checking Next destination
JAIST Koichiro Ochimizu

Consolidated Collaboration Diagram

- Consolidated Collaboration Diagram shows all the objects that participate In the use cases and all the interactions between these objects.


Consolidated Collaboration Diagram
<<timer>>
: Door Timer
<<output device interface>>
: ElevatorLamp Interface
<<output device interface>>
: Motor Interface
Floor Lamp Output
<<output device interface>>
: FloorLamp Interface
Floor Lamp Command
After (Timeout)
Off Elevator Lamp
Up
Down
Stop
Elevator Started
Elevator Stopped
Start Timer
Direction Lamp Output
<<output device interface>>
: DirectionLamp Interface
Direction Lamp Command
<<state dependent control>>
: Elevator Control
Open Door
Close Door
<<output device interface>>
: Door Interface
Door Command
Door Opened
Door Closed
Door Response
<<input device interface>>
: ArrivalSensor Interface
ArrivalSensor Input
Approaching Floor (Floor #)
Next Destination
Check Next Destination
Check This Floor (Floor #)
Approaching Requested Floor
Arrived(Floor #))
Departed(Floor #))
Arrived (Floor #)
Up, Down
<<entity>>
: Elevator Status&Plan
Departed(Floor #)
Update
Acknowledge
Elevator Button Interface
Elevator Request
Scheduler Request
Service Request
Floor Button Interface
<<input device interface>>
: ElevatorButton Interface
<<coordinator>>
: Elevator Manager
<<coordinator>>
: Scheduler
<<input device interface>>
: FloorButton Interface
Elevator Commitment
JAIST Koichiro Ochimizu

Content(2)

- **Object-oriented Software Development Methodology**
  - Outline of Unified Process and Use-case Driven Approach
  - Elevator Control System:
    Problem Description and Use-case Model
  - Elevator Control System:
    Finding of Problem Domain Objects
  - Elevator Control System:
    Sub-System Design and Task Design
  - Elevator Control System:
    Performance Evaluation
- **Product Line Technology**
  - Feature modeling
- **Aspect Oriented Software Design**
- **Contribution of OOT in Software Engineering**
  - History of SE Technologies and Contribution of OOT in SE field

Subsystem Structuring

- Structuring Criterion: Principles ”high coupling within a subsystem and low coupling between subsystems”
  - Aggregate/composite object: Geographical location : If two objects could potentially be physically separated in different locations, they should be in different subsystems to reduce communication cost
  - Clients and servers must be in separate subsystems
  - User interface objects are usually clients
  - A control objects and all the entity and interface objects it directly controls should all be part of one subsystem

Type of subsystems for concurrent, real-time, or distributed application domains

- **<<Control>>**

  The subsystem receives its inputs from the external environment and generates outputs to the external, usually without any human intervention. It includes at least one state-dependent control object. In some case, some Inputs data might be gathered by some other subsystem (s).
- **<<Coordinator>>**

  In cases with more than one control subsystem, it is sometimes necessary to have a coordinator subsystem that coordinates the control subsystems.
- **<<Data collection>>**

  A data collection subsystem collects data from the external environment.
- **<<Data analysis>>**

  A data analysis subsystem analyzes data and provides reports and/or displays for data collected by another subsystem.

Type of subsystems for concurrent, real-time, or distributed application domains

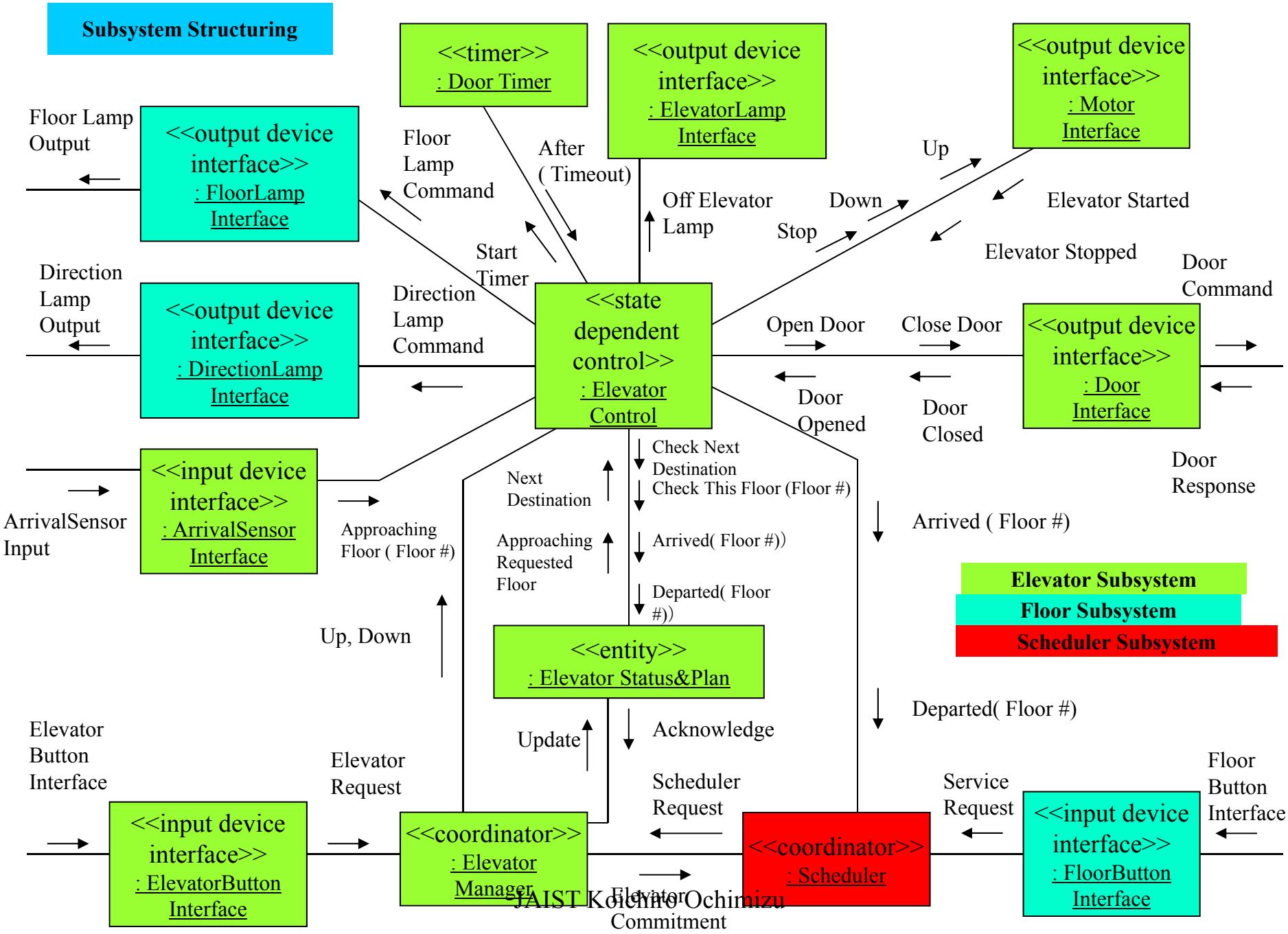
- **<<Server>>**

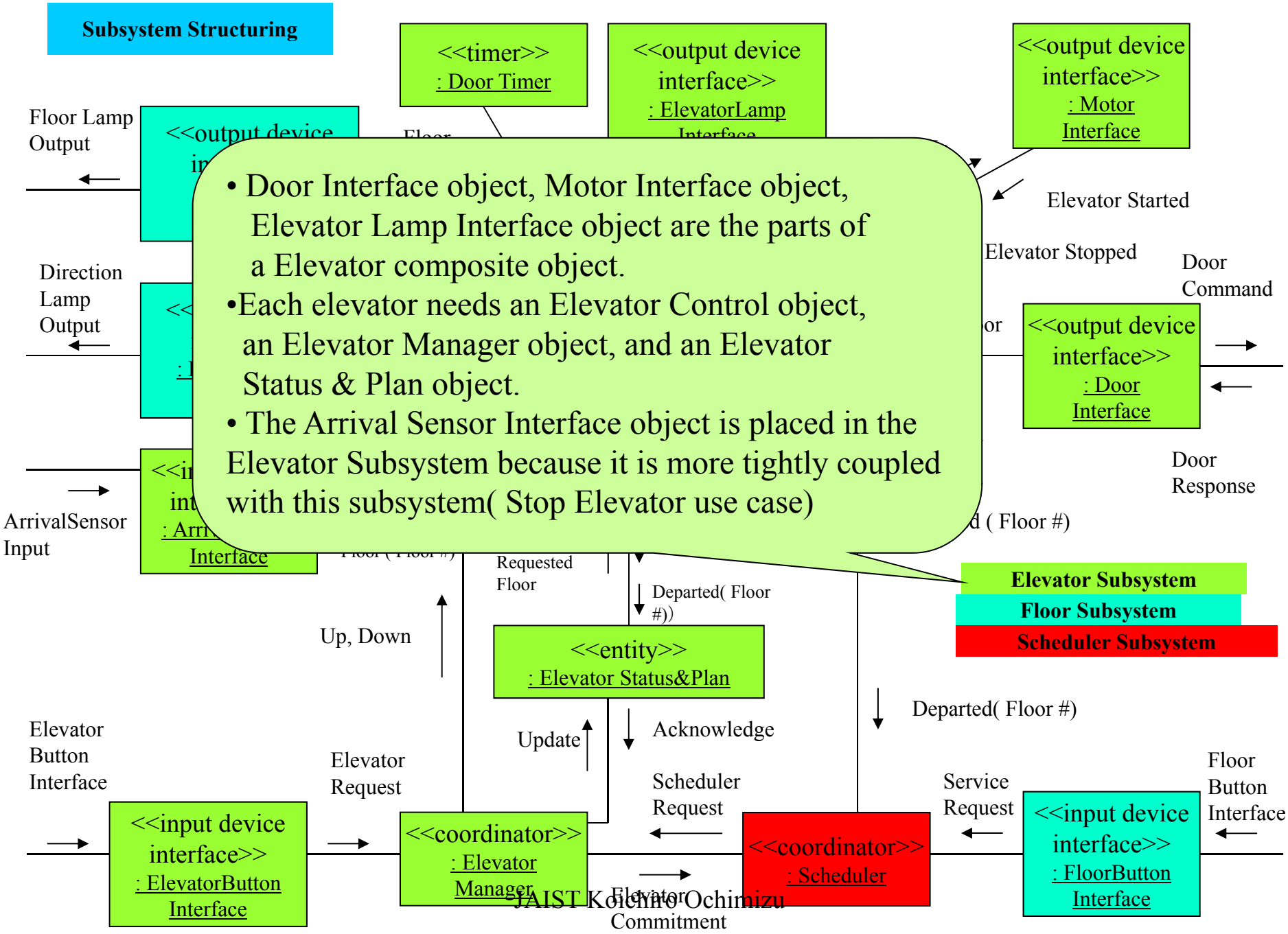
  A server subsystem provides a service for other subsystems. In the simplest case, a server object could consist of a single entity object.
- **<<User interface>>**

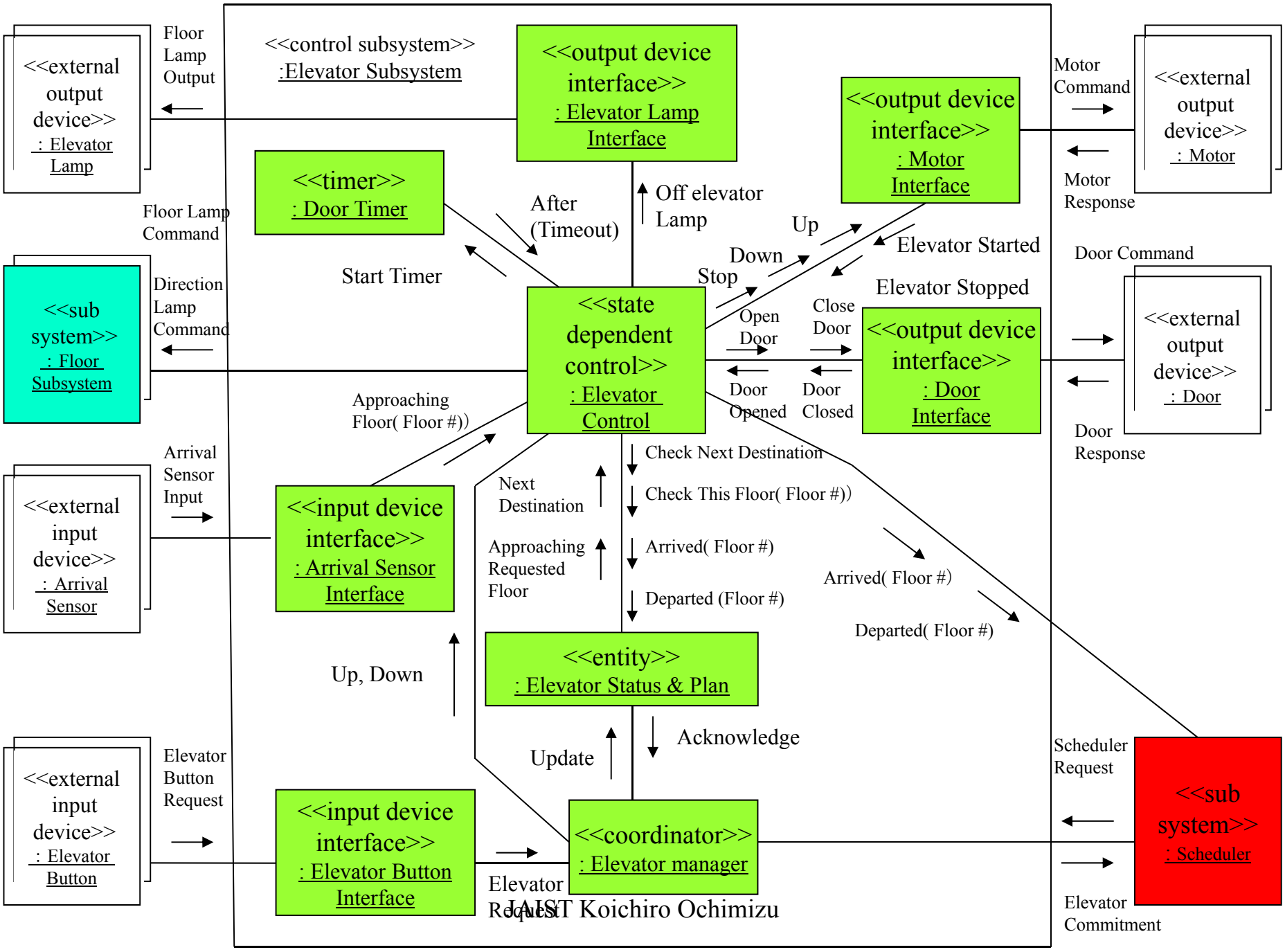
  A user interface subsystem provides the user interface and acts as a client. There may be more than one user interface subsystems. A user interface subsystem is usually a composite object that is composed of several simpler user interface objects.
- **<<I/O subsystem>>**

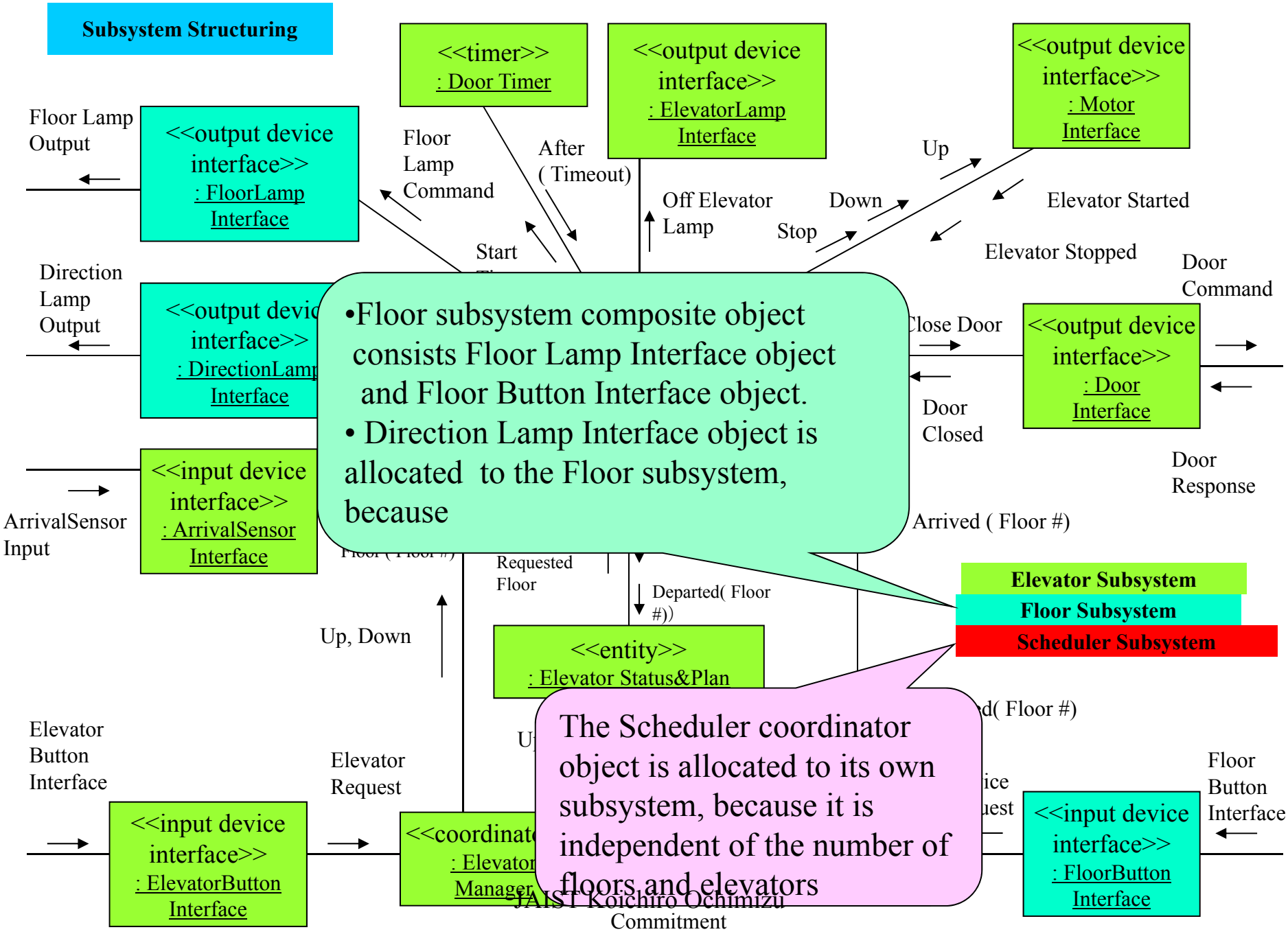
  In some systems, grouping al the device interface classes into an I/O subsystem might be useful, because developing device interface classes is a specialized skill.
- **<<System services>>**

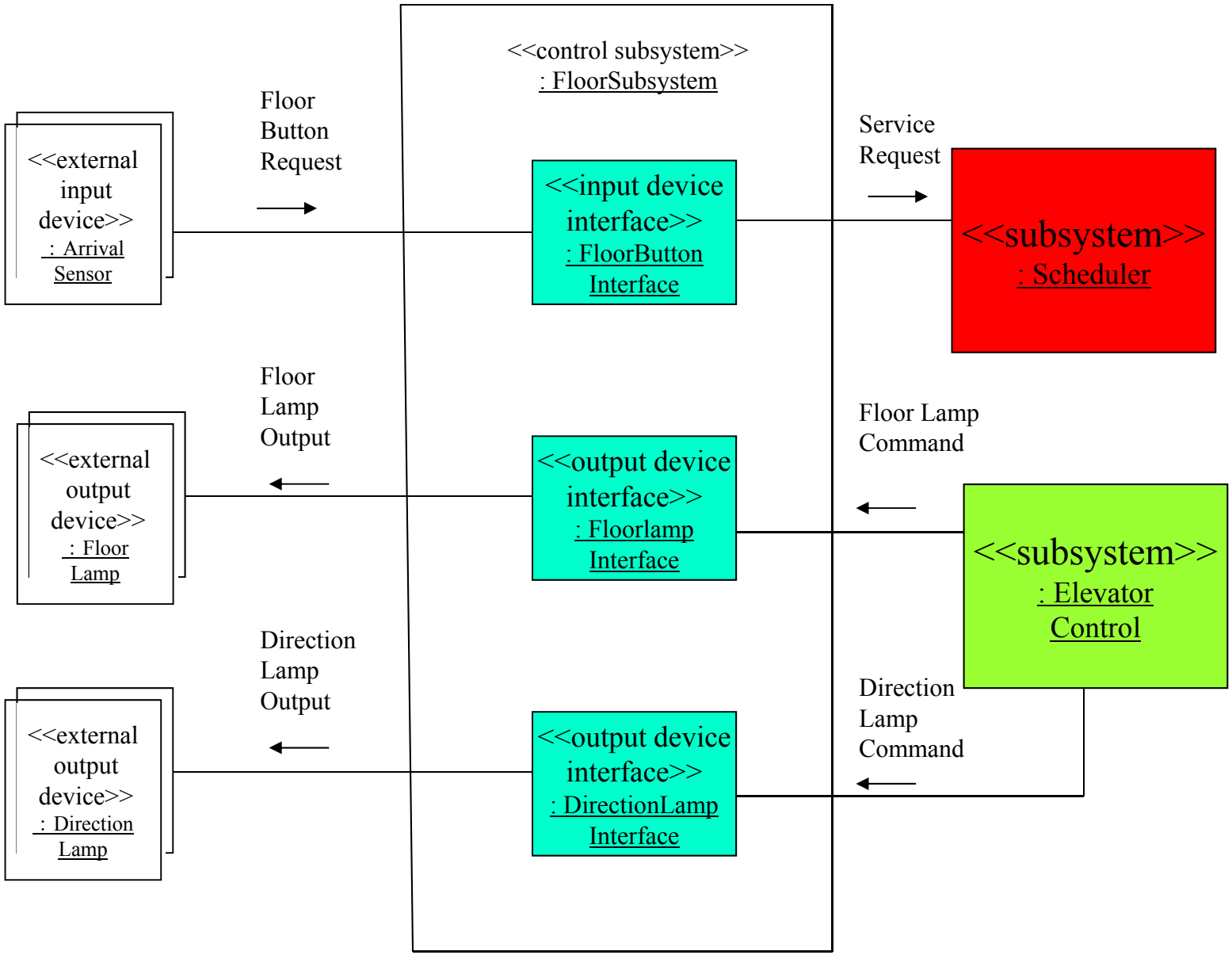
  Certain services are not problem domain-specific but provide system-level services, such as file management and network communication management.


Subsystem Structuring
<<timer>>
: Door Timer
<<output device interface>>
: ElevatorLamp Interface
<<output device interface>>
: Motor Interface
Floor Lamp Output
<<output device interface>>
: FloorLamp Interface
Floor Lamp Command
After (Timeout)
Off Elevator Lamp
Up
Down
Stop
Elevator Started
Elevator Stopped
Start Timer
Direction Lamp Output
<<output device interface>>
: DirectionLamp Interface
Direction Lamp Command
<<state dependent control>>
: Elevator Control
Open Door
Close Door
<<output device interface>>
: Door Interface
Door Command
Door Opened
Door Closed
Door Response
<<input device interface>>
: ArrivalSensor Interface
ArrivalSensor Input
Approaching Floor (Floor #)
Next Destination
Approaching Requested Floor
Check Next Destination
Check This Floor (Floor #)
Arrived(Floor #))
Departed(Floor #))
Arrived (Floor #)
Elevator Subsystem
Floor Subsystem
Scheduler Subsystem
Up, Down
<<entity>>
: Elevator Status&Plan
Departed(Floor #)
Elevator Button Interface
Update
Acknowledge
Elevator Request
Scheduler Request
Service Request
Floor Button Interface
<<input device interface>>
: ElevatorButton Interface
<<coordinator>>
: Elevator Manager
<<coordinator>>
: Scheduler
<<input device interface>>
: FloorButton Interface
Elevator Commitment
JAIST Koichiro Ochimizu

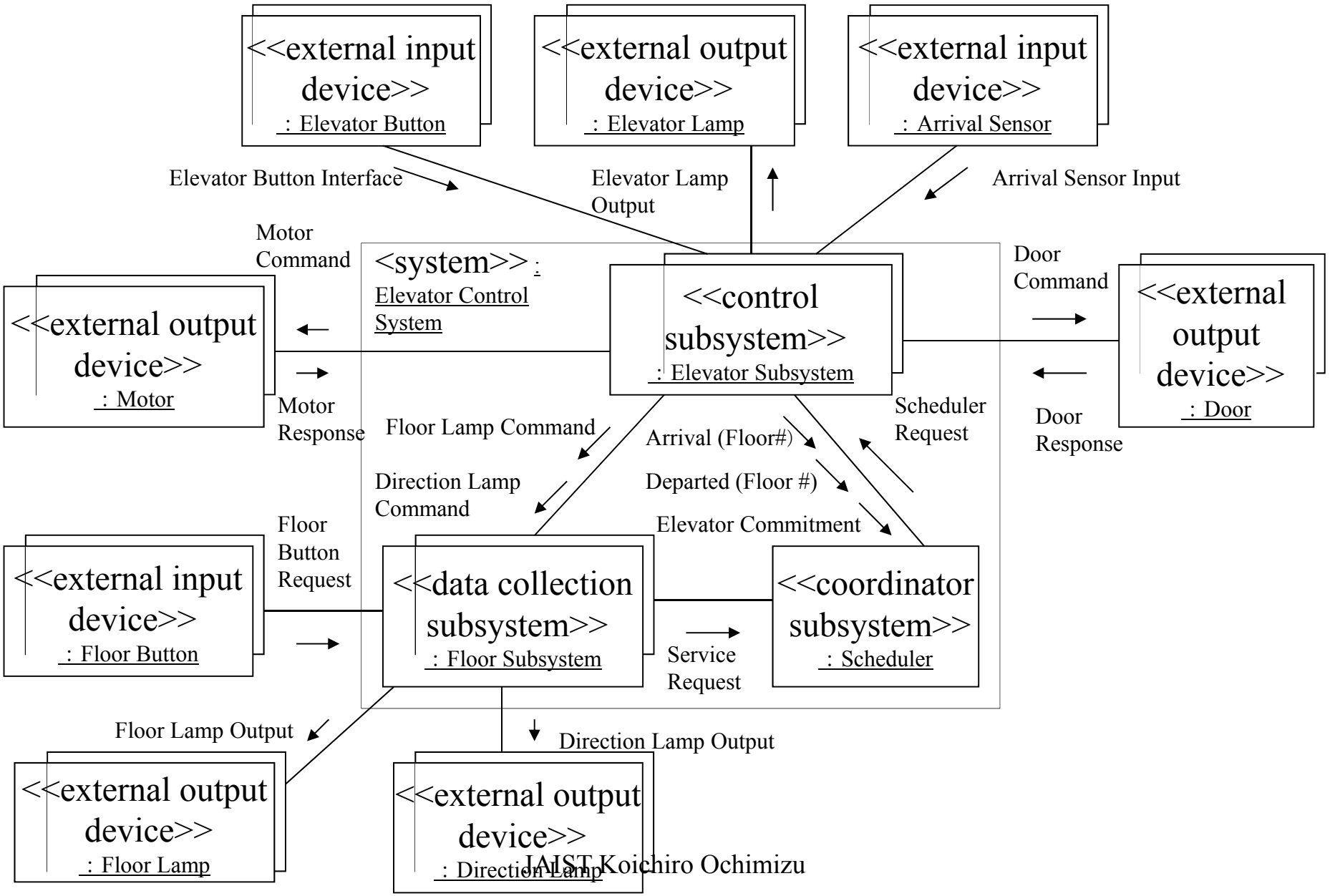

Subsystem Structuring
<<timer>>
: Door Timer
<<output device interface>>
: ElevatorLamp Interface
<<output device interface>>
: Motor Interface
Floor Lamp Output
Direction Lamp Output
Elevator Started
Elevator Stopped
Door Command
<<output device interface>>
: Door Interface
Door Response
ArrivalSensor Input
: Arri Interface
Requested Floor
Departed(Floor #))
Up, Down
<<entity>>
: Elevator Status&Plan
Elevator Subsystem
Floor Subsystem
Scheduler Subsystem
Departed(Floor #)
Elevator Button Interface
Update
Acknowledge
Elevator Request
Scheduler Request
Service Request
Floor Button Interface
<<input device interface>>
: ElevatorButton Interface
<<coordinator>>
: Elevator Manager
<<coordinator>>
: Scheduler
<<input device interface>>
: FloorButton Interface
Elevator Commitment
• Door Interface object, Motor Interface object, Elevator Lamp Interface object are the parts of a Elevator composite object.
•Each elevator needs an Elevator Control object, an Elevator Manager object, and an Elevator Status & Plan object.
• The Arrival Sensor Interface object is placed in the Elevator Subsystem because it is more tightly coupled with this subsystem(Stop Elevator use case)


<<external output device>>
: Elevator Lamp
Floor Lamp Output
<<control subsystem>>
:Elevator Subsystem
<<output device interface>>
: Elevator Lamp Interface
<<output device interface>>
: Motor Interface
Motor Command
Motor Response
<<external output device>>
: Motor
<<timer>>
: Door Timer
After (Timeout)
Off elevator Lamp
Floor Lamp Command
Start Timer
Up
Down
Stop
Elevator Started
Elevator Stopped
Door Command
<<sub system>>
: Floor Subsystem
Direction Lamp Command
<<state dependent control>>
: Elevator Control
Open Door
Close Door
Door Opened
Door Closed
<<output device interface>>
: Door Interface
<<external output device>>
: Door
Door Response
Approaching Floor(Floor #))
Arrival Sensor Input
<<external input device>>
: Arrival Sensor
<<input device interface>>
: Arrival Sensor Interface
Check Next Destination
Next Destination
Check This Floor(Floor #))
Approaching Requested Floor
Arrived(Floor #)
Departed (Floor #)
Arrived(Floor #)
Departed(Floor #)
Up, Down
<<entity>>
: Elevator Status & Plan
Update
Acknowledge
Scheduler Request
<<external input device>>
: Elevator Button
Elevator Button Request
<<input device interface>>
: Elevator Button Interface
<<coordinator>>
: Elevator manager
<<sub system>>
: Scheduler
Elevator Request
Elevator Commitment
JAIST Koichiro Ochimizu

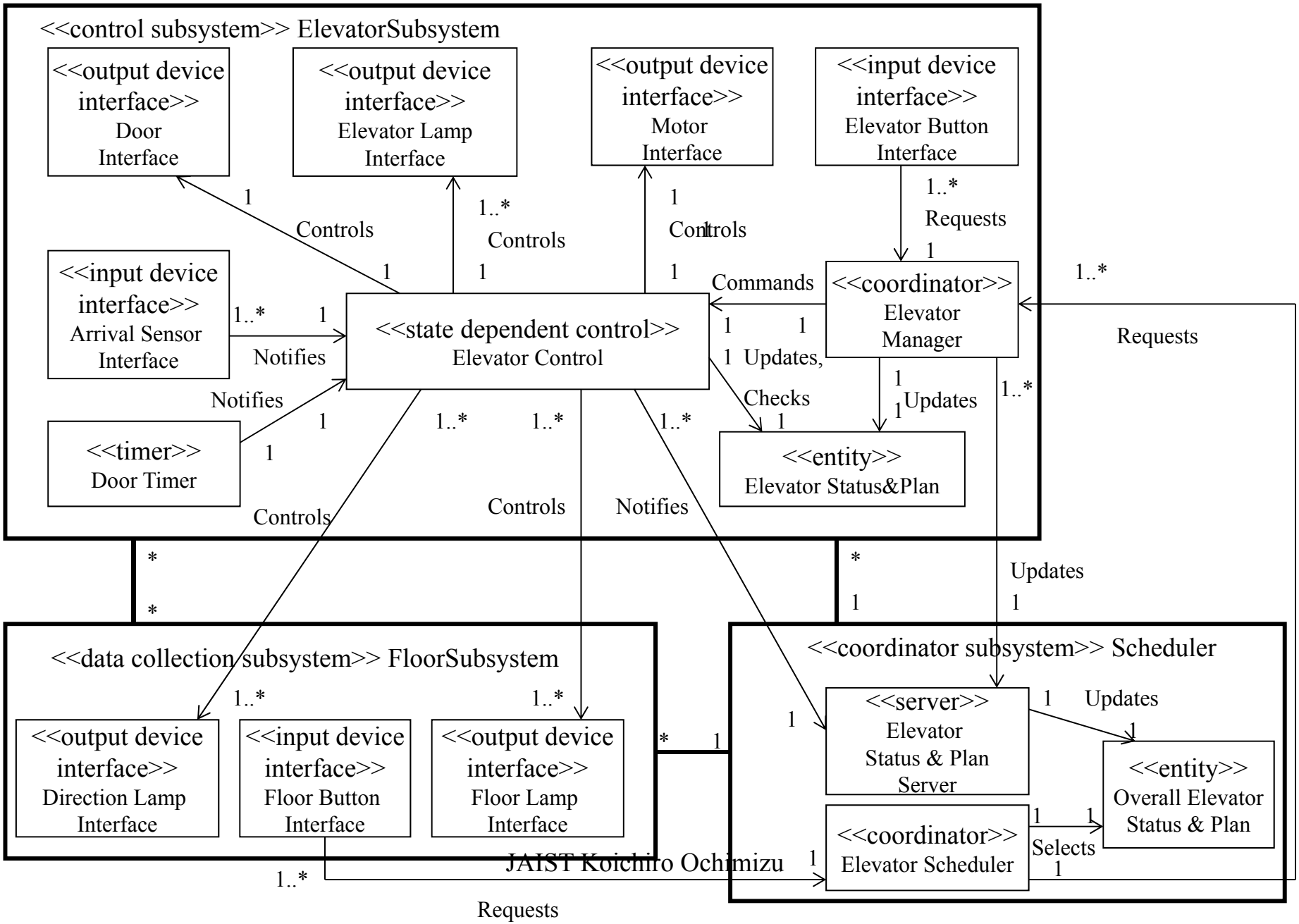

Subsystem Structuring
<<timer>>
: Door Timer
<<output device interface>>
: ElevatorLamp Interface
<<output device interface>>
: Motor Interface
Floor Lamp Output
<<output device interface>>
: FloorLamp Interface
Floor Lamp Command
After (Timeout)
Off Elevator Lamp
Up
Down
Stop
Elevator Started
Elevator Stopped
Start
Direction Lamp Output
<<output devic
interface>>
: DirectionLamp Interface
•Floor subsystem composite object consists Floor Lamp Interface object and Floor Button Interface object.
• Direction Lamp Interface object is allocated to the Floor subsystem, because
Close Door
Door Closed
<<output device interface>>
: Door Interface
Door Command
Door Response
<<input device interface>>
: ArrivalSensor Interface
ArrivalSensor Input
Arrived (Floor #)
Requested Floor
Departed(Floor #))
Elevator Subsystem
Floor Subsystem
Scheduler Subsystem
Up, Down
<<entity>>
: Elevator Status&Plan
Elevator Button Interface
Elevator Request
<<input device interface>>
: ElevatorButton Interface
<<coordinat
: Elevator Manager
The Scheduler coordinator object is allocated to its own subsystem, because it is independent of the number of floors and elevators
Commitment
<<input device interface>>
: FloorButton Interface
Floor Button Interface


<<control subsystem>>
: FloorSubsystem
<<external input device>>
: Arrival Sensor
Floor Button Request
<<input device interface>>
: FloorButton Interface
Service Request
<<subsystem>>
: Scheduler
<<external output device>>
: Floor Lamp
Floor Lamp Output
<<output device interface>>
: Floorlamp Interface
Floor Lamp Command
<<subsystem>>
: Elevator Control
<<external output device>>
: Direction Lamp
Direction Lamp Output
<<output device interface>>
: DirectionLamp Interface
Direction Lamp Command

Subsystems


<<external input device>>
: Elevator Button
<<external output device>>
: Elevator Lamp
<<external input device>>
: Arrival Sensor
Elevator Button Interface
Elevator Lamp Output
Arrival Sensor Input
Motor Command
<system>> : Elevator Control System
<<control subsystem>>
: Elevator Subsystem
Door Command
<<external output device>>
: Motor
Motor Response
<<external output device>>
: Door
Door Response
Scheduler Request
Floor Lamp Command
Arrival (Floor#)
Direction Lamp Command
Departed (Floor #)
Floor Button Request
Elevator Commitment
<<external input device>>
: Floor Button
<<data collection subsystem>>
: Floor Subsystem
<<coordinator subsystem>>
: Scheduler
Service Request
Floor Lamp Output
Direction Lamp Output
<<external output device>>
: Floor Lamp
<<external output device>>
: Direction Lamp

Refined Static Model (Class Diagram) for Elevator Control System



Task Structuring

- Design task structure and task interface by applying the following task structuring criteria to problem domain objects recognized as an consolidated collaboration diagram.
  - **I/O task structuring criteria**

    Criteria to decide whether each device interface object is an active object or not, considering the properties: interrupt-driven, polling, communication, discrete data or analog data
  - **Internal task structuring criteria**

    Criteria to decide whether each internal object is an active object or not, considering the properties: period, asynchronous, control, UI.
  - **Task priority criteria**

    Criteria to decide whether each internal object is an active object or not, considering the properties: time-critical, computation (CPU bound).
  - **Task clustering criteria**

    Criteria to group active objects selected by the above criteria, considering properties: time, sequencing, control, mutual exclusion..

Criteria to make the objects (in a consolidated collaboration diagram) active objects

Criteria to group the active objects to reduce the number of tasks

Task Structuring Criteria (Outline)

- **I/O task structuring criteria**
  - Asynchronous I/O Device Interface tasks
  - Periodic I/O Device Interface tasks
  - Passive I/O Device Interface tasks
  - Resource Monitor tasks
- **Internal task structuring criteria**
  - Periodic Tasks
  - Asynchronous tasks
  - Control tasks
  - User Interface tasks
- **Task priority criteria**
  - Time-Critical tasks
  - Non-Time-Critical Computationally Intensive tasks
- **Task clustering criteria**
  - Temporal Clustering
  - Sequential Clustering
  - Control Clustering
  - Mutually Exclusive Clustering

I/O Task Structuring Criteria

Necessary to determine the hardware characteristics of the I/O device that interface to the system, and the nature of the data being input to the system to these devices.

- Asynchronous (active) I/O devices:

  For each asynchronous I/O device, an asynchronous I/O device interface task is needed not to miss an interrupt.

- Periodic I/O Device Interface Tasks:

  If passive input (or output) devices are polled (or addressed) periodically by a timer, a periodic I/O device interface task is needed.

- Passive I/O Devices Interface Tasks:

  For passive I/O devices that do not need to be polled, passive I/O devices interface tasks are needed when it is considered desirable to overlap computation with I/O.

- Resource Monitor Task:

  An input or output device that receives requests from multiple sources should have a resource monitor task to coordinate these requests, even if the device is passive. A resource monitor task has to sequence these requests so as to maintain data integrity and ensure that no data is corrupted or lost.

Internal Task Structuring Criteria

- **Periodic Tasks**

  An activity that needs to be executed periodically (i.e. at regular, equally spaced intervals of time) is structured as a separate periodic task. The task is activated by a timer event, performs the periodic activity.
- **Asynchronous Tasks**

  The demand-driven(the arrival of internal messages or events) activities are typically handled by means of asynchronous tasks.
- **Control Tasks**

  A task that executes a sequential state-chart is referred to as a control task.
- **User Interface Tasks**

  A user typically performs a set of sequential operations, this can be handled by a use Interface task.

Task Priority Criteria

Task priority criteria take into account priority considerations in task structuring, in particular, high- and low-priority tasks are considered.

- **Time-Critical Tasks**

  A time-critical task is a task that needs to meet a hard deadline. Such a task needs to run at a high priority.

- **Non-Time-Critical Computationally Intensive Tasks**

  A non-time-critical computationally intensive task may run as a low-priority task consuming spare CPU cycles. A low-priority computationally intensive task executing as a background task that is preempted by higher-priority foreground tasks has its origin in early multiprogramming systems and is typically supported by most modern operating systems.

Task Clustering Criteria(1/3)

Reduce the number of tasks

- **Temporal Clustering**

  Certain candidate tasks may be activated by the same event, e.g. a timer event. If there is no sequential dependency between the candidate tasks, they may be grouped into the same task, based on the temporal clustering. Some tradeoffs need to be considered.

  – If some candidate task is more time critical than the others, the task should not be combined.
  – If two candidate tasks could be executed on separate processors, they should not be combined.
  – Preference should be given in temporal clustering to tasks that are functionally related and likely to be of equal importance from a scheduling viewpoint.
  – Two tasks with different periods may not be clustered.

Task Clustering Criteria(2/3)

Reduce the number of tasks

- **Sequential Clustering**

  The first candidate task is triggered by an asynchronous or periodic event and the other are then executed sequentially after it. These sequentially dependent candidate tasks may be grouped. But,

  – If the last candidate task in a sequence does not send an inter-task message, this terminates the group of tasks to be considered for sequential clustering.

  – If the next candidate task in the sequence also receives inputs from another source and therefore can be activated by receiving input from that source, this candidate task should be left as a separate task.

  – If the next candidate task in sequence is of a lower priority, they should be kept as separate task.

Task Clustering Criteria(3/3)

Reduce the number of tasks

- **Control Clustering**

  A control object, which executes a sequential state-chart, is mapped to a control task.

  – The actions activated during state transition are executed within the thread of control of the control object.

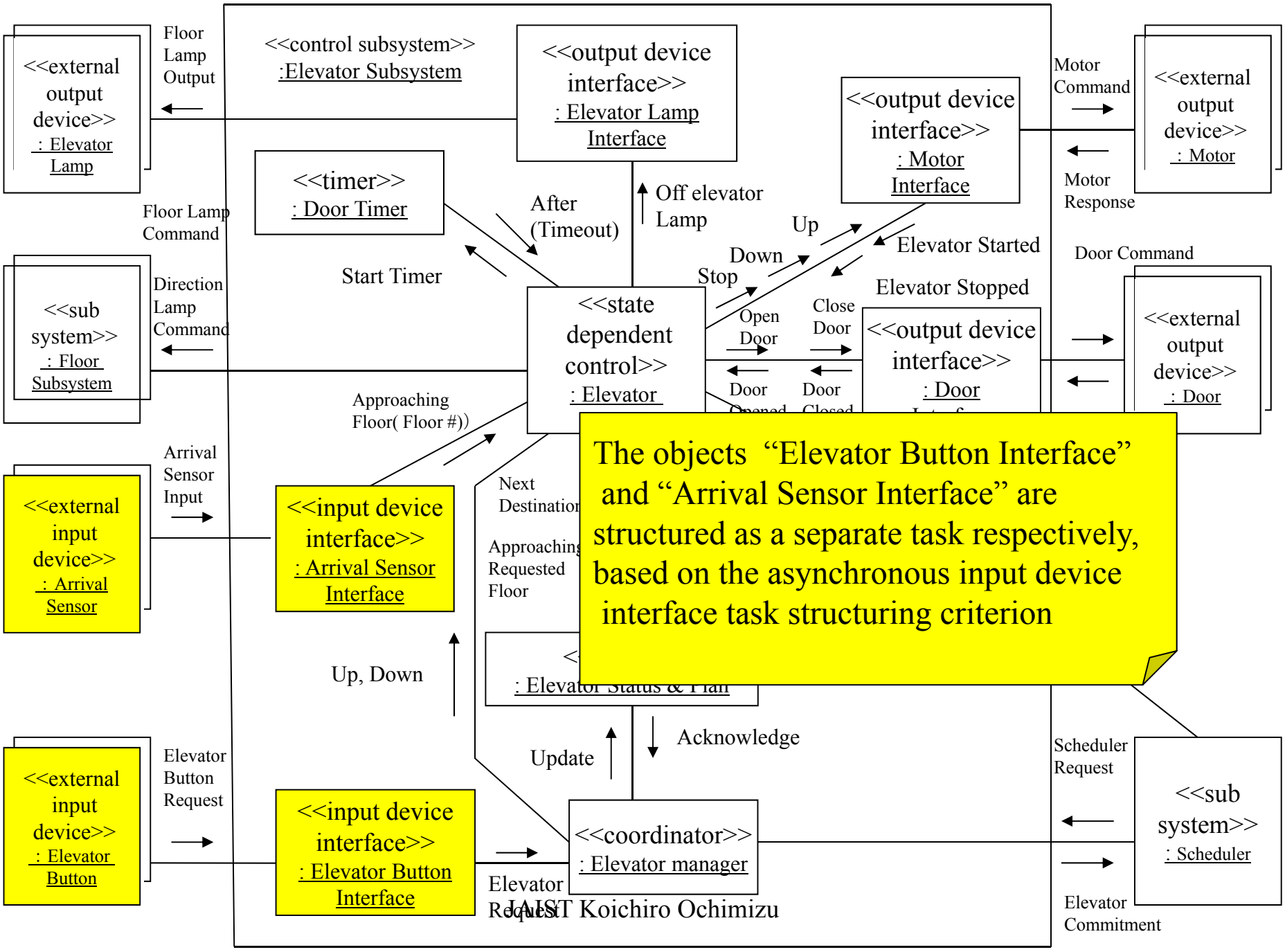
  – The activity should be structured as a separate task.

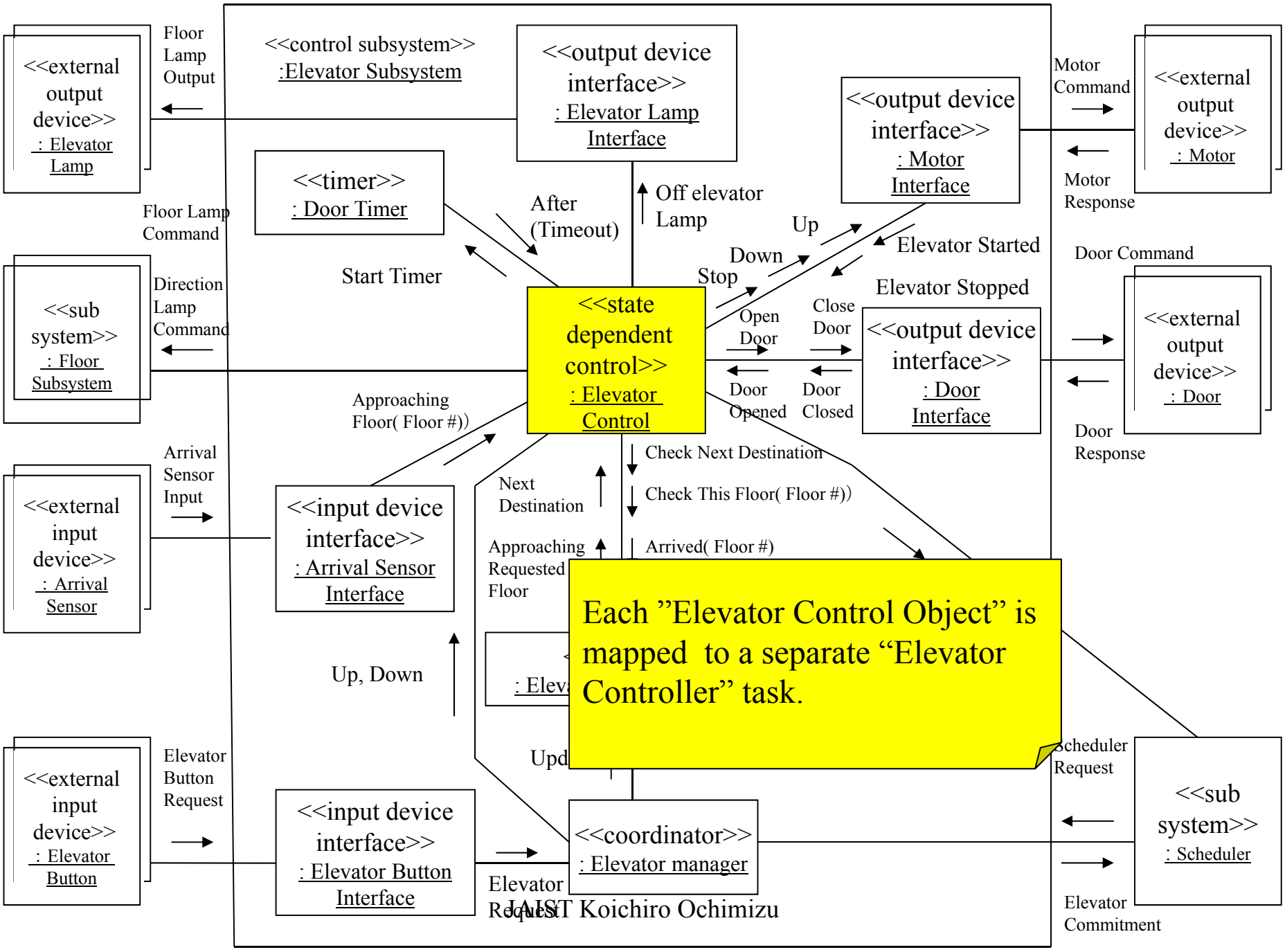
- **Mutually Exclusive Clustering**

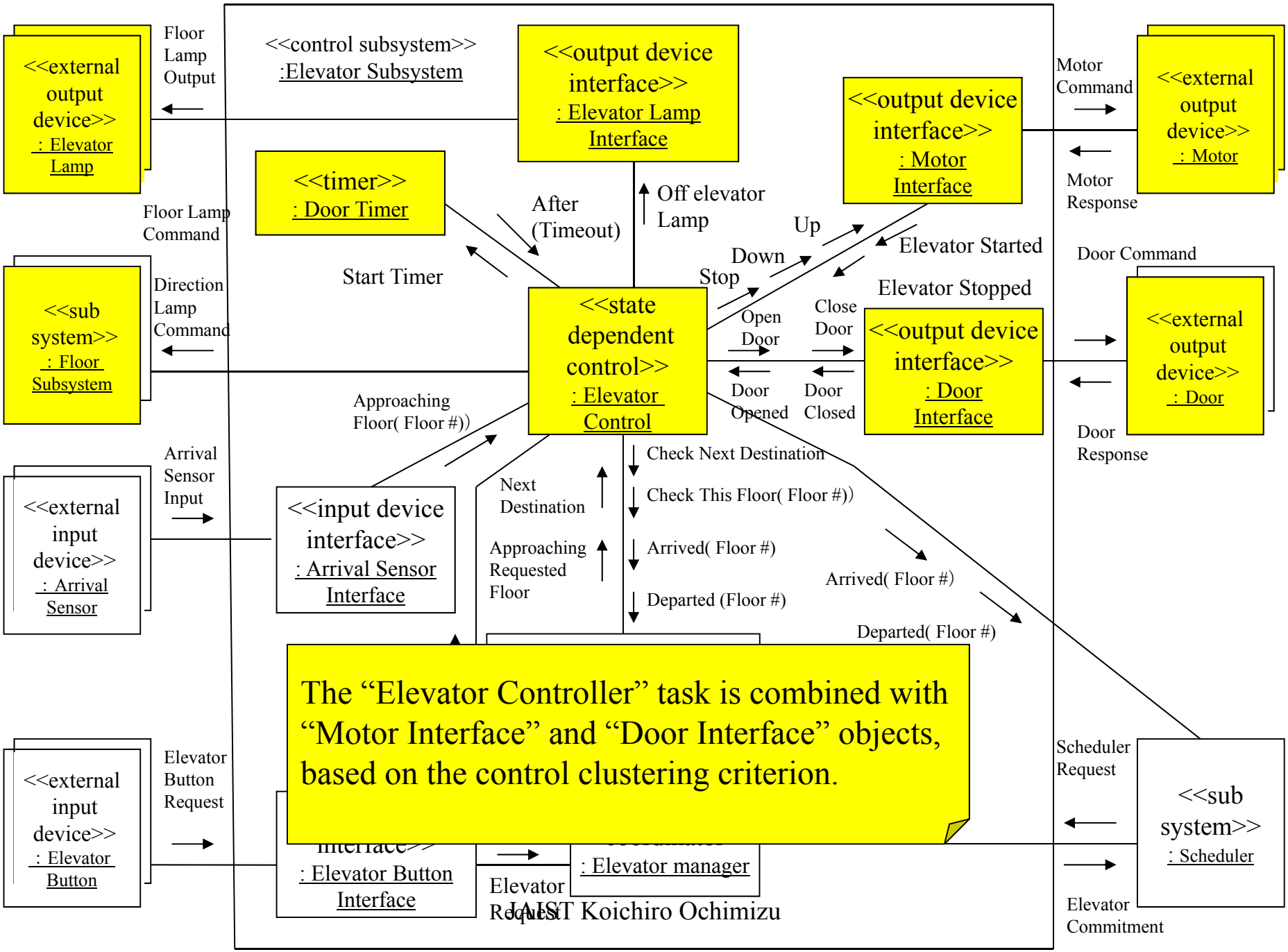
  Mutually exclusive tasks may be clustered.

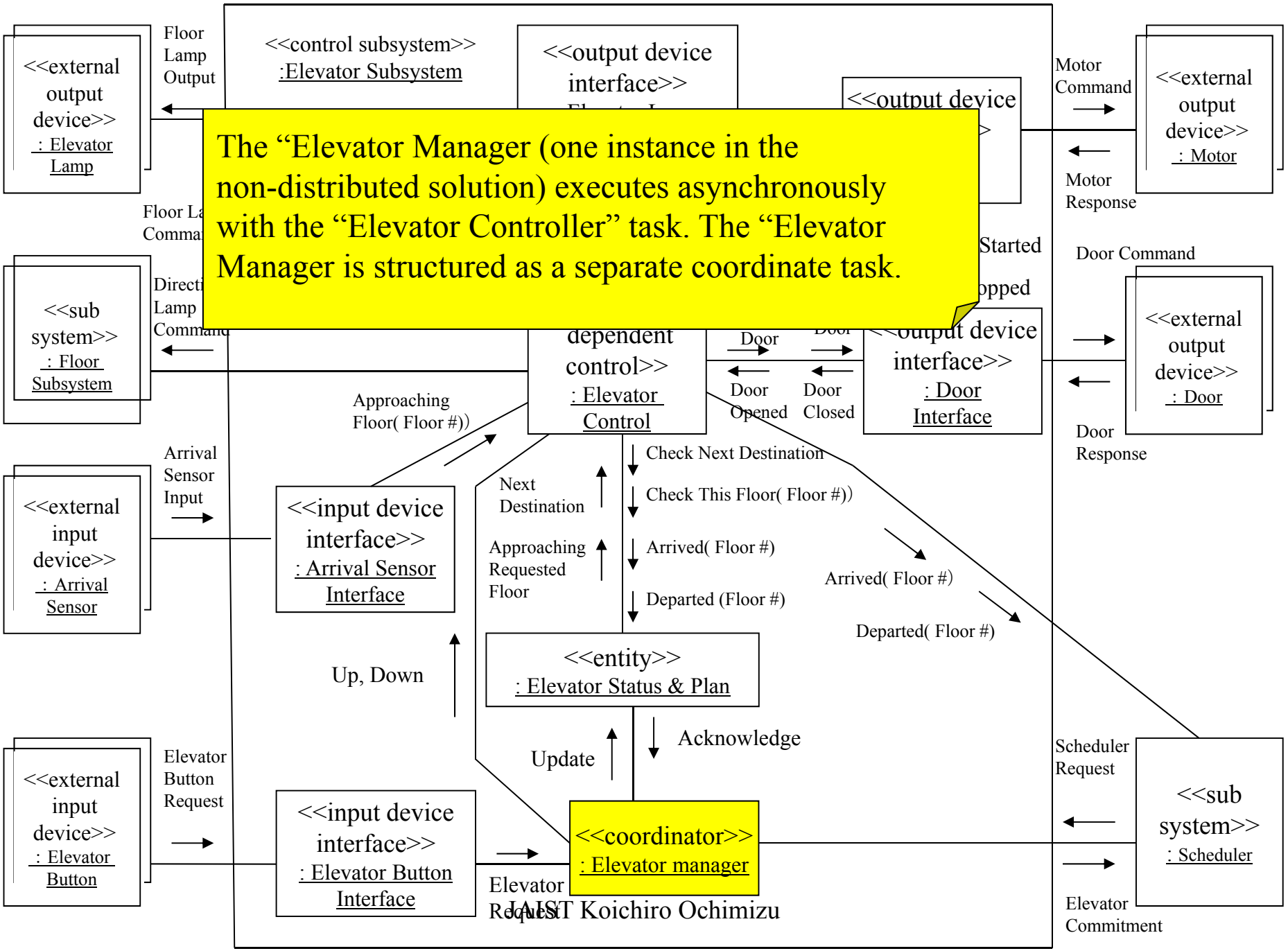
Non-Distributed Solution

- The Elevator Control System is mapped to a single CPU or tightly coupled multiprocessor configuration
- The Elevator Status & Plan entity object is accessible to all elevators as well as scheduler, so that one centralized repository of data can be used


<<external output device>> : Elevator Lamp
Floor Lamp Output
<<control subsystem>> :Elevator Subsystem
<<output device interface>> : Elevator Lamp Interface
<<output device interface>> : Motor Interface
Motor Command
Motor Response
<<external output device>> : Motor
<<timer>> : Door Timer
After (Timeout)
Off elevator Lamp
Floor Lamp Command
Start Timer
Up
Down
Stop
Elevator Started
Elevator Stopped
Door Command
<<sub system>> : Floor Subsystem
Direction Lamp Command
<<state dependent control>> : Elevator
Open Door
Close Door
<<output device interface>> : Door
Door Opened
Door Closed
<<external output device>> : Door
Approaching Floor(Floor #))
Arrival Sensor Input
<<external input device>> : Arrival Sensor
<<input device interface>> : Arrival Sensor Interface
Next Destination
Approaching Requested Floor
: Elevator Status & Plan
Up, Down
Update
Acknowledge
Scheduler Request
<<external input device>> : Elevator Button
Elevator Button Request
<<input device interface>> : Elevator Button Interface
<<coordinator>> : Elevator manager
<<sub system>> : Scheduler
Elevator Request
Elevator Commitment
The objects "Elevator Button Interface" and "Arrival Sensor Interface" are structured as a separate task respectively, based on the asynchronous input device interface task structuring criterion
JAIST Koichiro Ochimizu


<<external output device>> : Elevator Lamp
Floor Lamp Output
<<control subsystem>> :Elevator Subsystem
<<output device interface>> : Elevator Lamp Interface
<<output device interface>> : Motor Interface
Motor Command
<<external output device>> : Motor
Motor Response
<<timer>> : Door Timer
After (Timeout)
Off elevator Lamp
Up
Down
Stop
Elevator Started
Elevator Stopped
Floor Lamp Command
Start Timer
<<sub system>> : Floor Subsystem
Direction Lamp Command
<<state dependent control>> : Elevator Control
Open Door
Close Door
Door Opened
Door Closed
<<output device interface>> : Door Interface
Door Command
<<external output device>> : Door
Door Response
Approaching Floor(Floor #))
Check Next Destination
Check This Floor(Floor #))
Arrived(Floor #)
Next Destination
Approaching Requested Floor
Arrival Sensor Input
<<external input device>> : Arrival Sensor
<<input device interface>> : Arrival Sensor Interface
Up, Down
: Elev
Each ”Elevator Control Object” is mapped to a separate “Elevator Controller” task.
Upd
Elevator Button Request
<<external input device>> : Elevator Button
<<input device interface>> : Elevator Button Interface
<<coordinator>> : Elevator manager
Elevator Request
Scheduler Request
<<sub system>> : Scheduler
Elevator Commitment
JAIST Koichiro Ochimizu

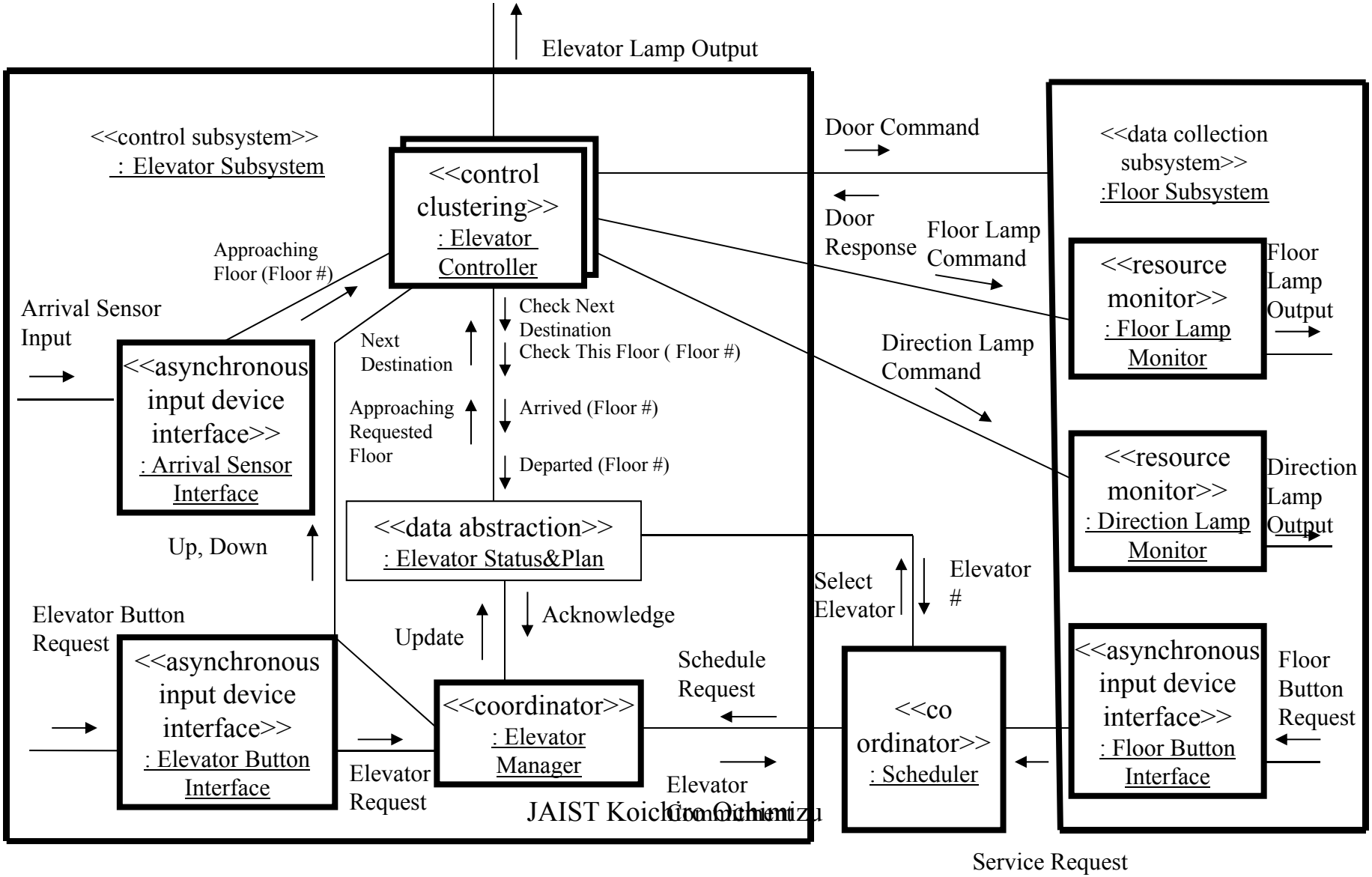

<<external output device>> : Elevator Lamp
Floor Lamp Output
<<control subsystem>> :Elevator Subsystem
<<output device interface>> : Elevator Lamp Interface
<<output device interface>> : Motor Interface
Motor Command
<<external output device>> : Motor
Motor Response
<<timer>> : Door Timer
After (Timeout)
Off elevator Lamp
Up
Down
Stop
Elevator Started
Elevator Stopped
Floor Lamp Command
Start Timer
<<sub system>> : Floor Subsystem
Direction Lamp Command
<<state dependent control>> : Elevator Control
Open Door
Close Door
Door Opened
Door Closed
<<output device interface>> : Door Interface
Door Command
<<external output device>> : Door
Door Response
Approaching Floor(Floor #))
Arrival Sensor Input
<<external input device>> : Arrival Sensor
<<input device interface>> : Arrival Sensor Interface
Next Destination
Approaching Requested Floor
Check Next Destination
Check This Floor(Floor #))
Arrived(Floor #)
Departed (Floor #)
Arrived(Floor #)
Departed(Floor #)
The "Elevator Controller" task is combined with "Motor Interface" and "Door Interface" objects, based on the control clustering criterion.
<<external input device>> : Elevator Button
Elevator Button Request
: Elevator Button Interface
: Elevator manager
Elevator Request
Scheduler Request
<<sub system>> : Scheduler
Elevator Commitment
JAIST Koichiro Ochimizu



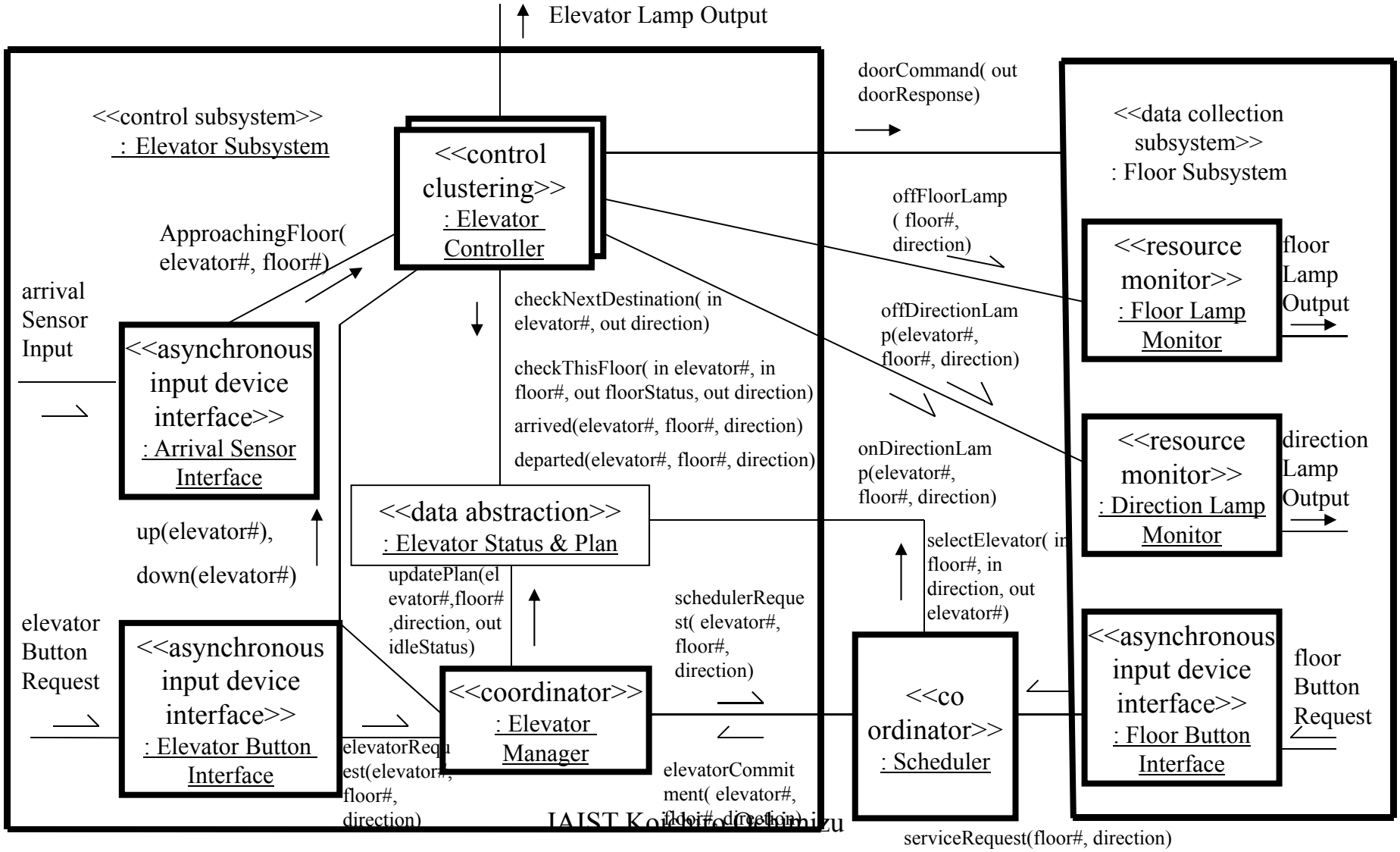
The "Elevator Manager (one instance in the non-distributed solution) executes asynchronously with the "Elevator Controller" task. The "Elevator Manager is structured as a separate coordinate task.

JAIST Koichiro Ochimizu

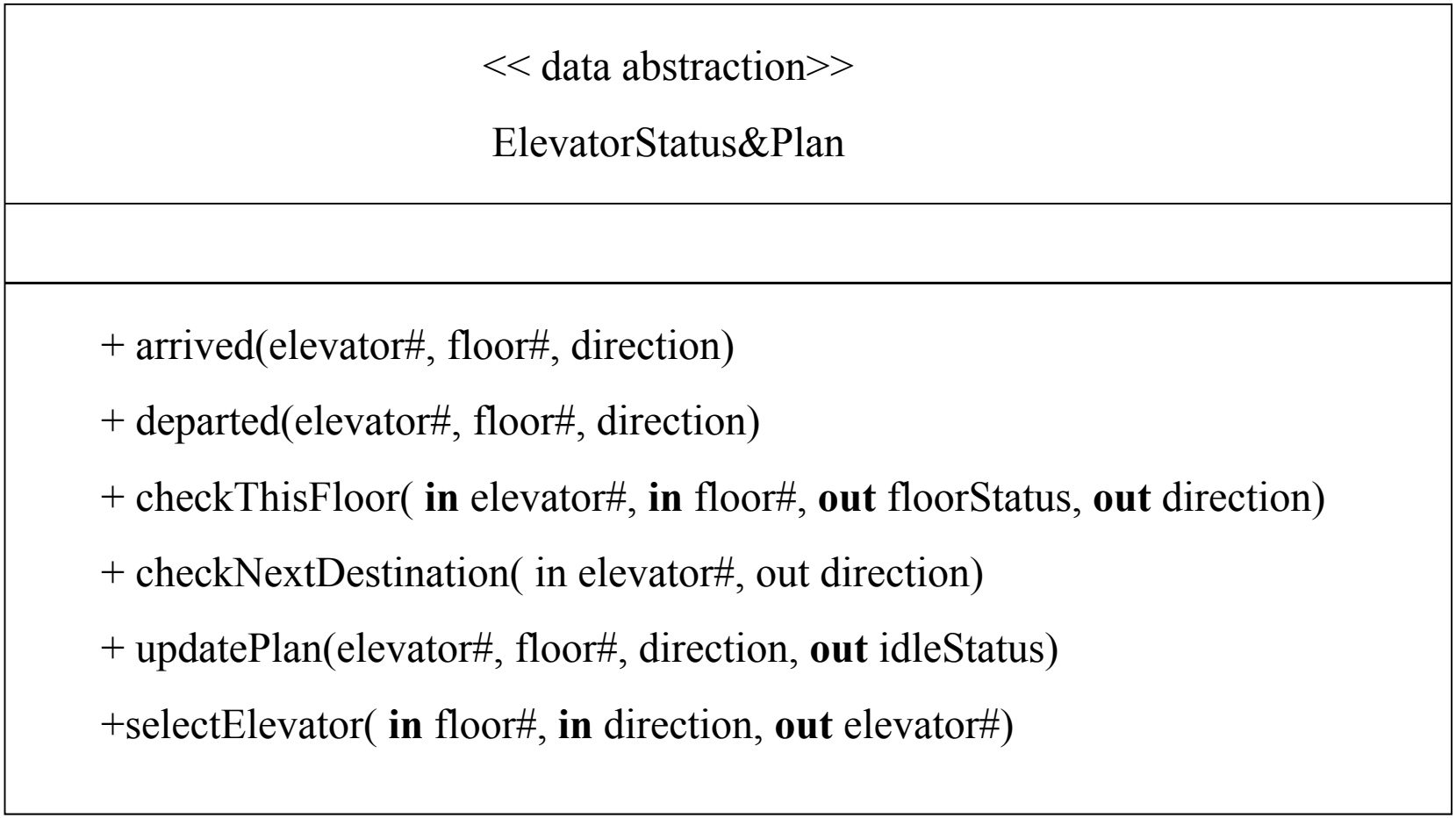
Task Architecture
(Non-distributed Elevator Control System)



Task Interface
(Non-distributed Elevator Control System)



Data Abstraction Classes



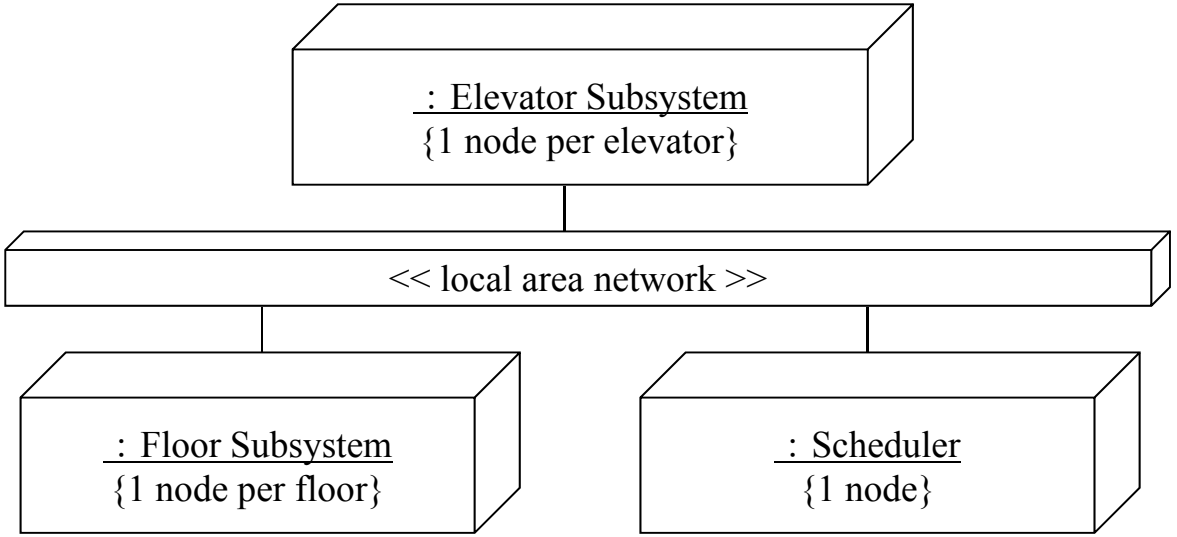
Data abstraction class for centralized solution

JAIST Koichiro Ochimizu

Distributed Elevator Control System

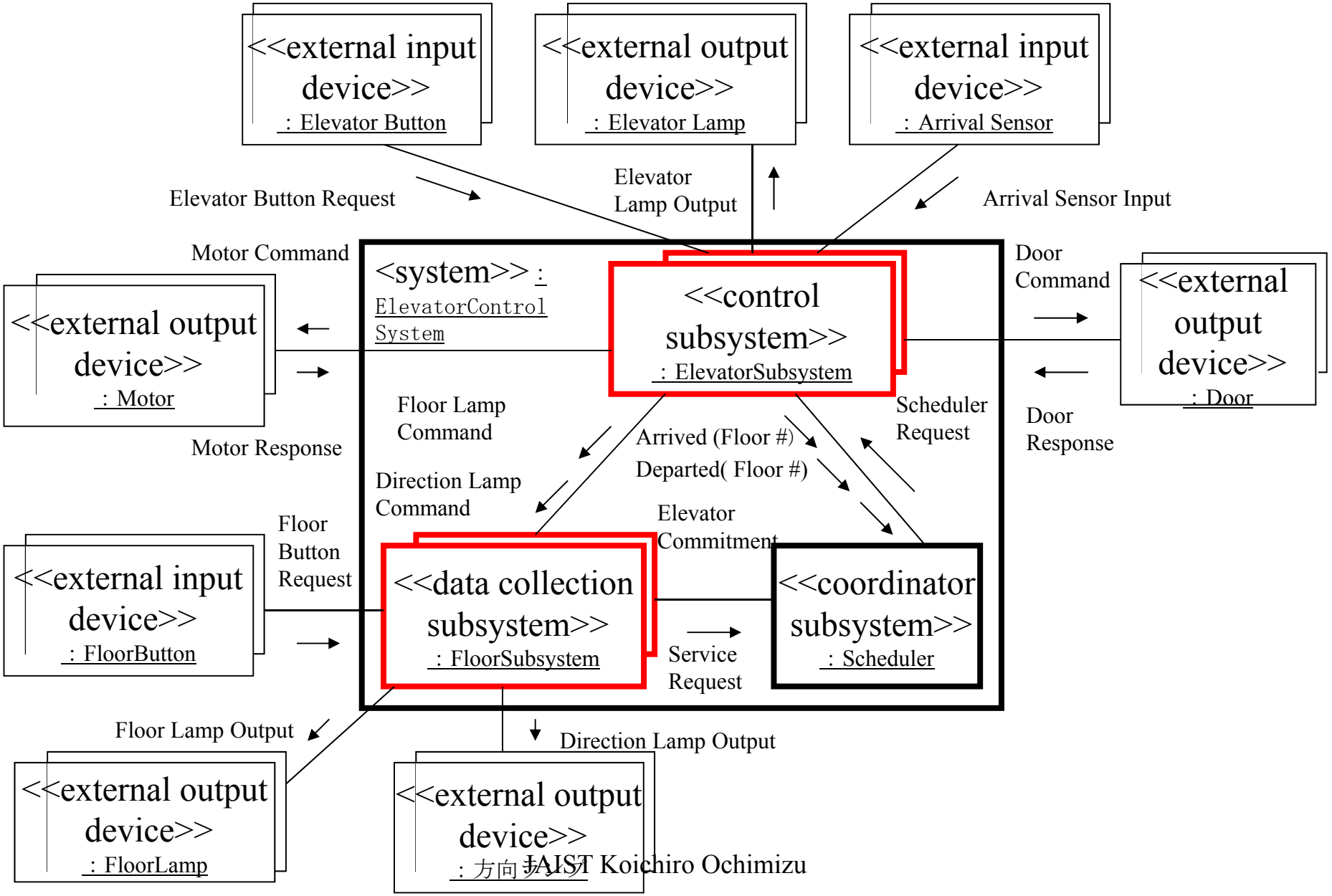
- The physical configuration consists of multiple nodes interconnected by a local area network.
- Multiple instances of the Elevator Subsystem (one instance per elevator)
- Multiple instances of the Floor Subsystem (one instance per floor)
- One instance of the Scheduler subsystem
- All communication between the subsystems is via loosely coupled message communication.
- There is no shared memory in a distributed configuration; thus the "Scheduler" and multiple instances of the "Elevator Subsystem" can not directly access the "Elevator Status & Plan" data abstraction object.
- Client-Server solution presents the potential danger of creating a bottleneck at this server. Instead, an alternative solution is to use replicated data. Each instance of the Elevator Subsystem maintains its own local instance of the Elevator Status & Plan, called Local Elevator Status & Plan. The scheduler also maintains a copy of the Elevator Status & Plan, called Overall Elevator Status & Plan.

Distributed Elevator Control System Deployment Diagram

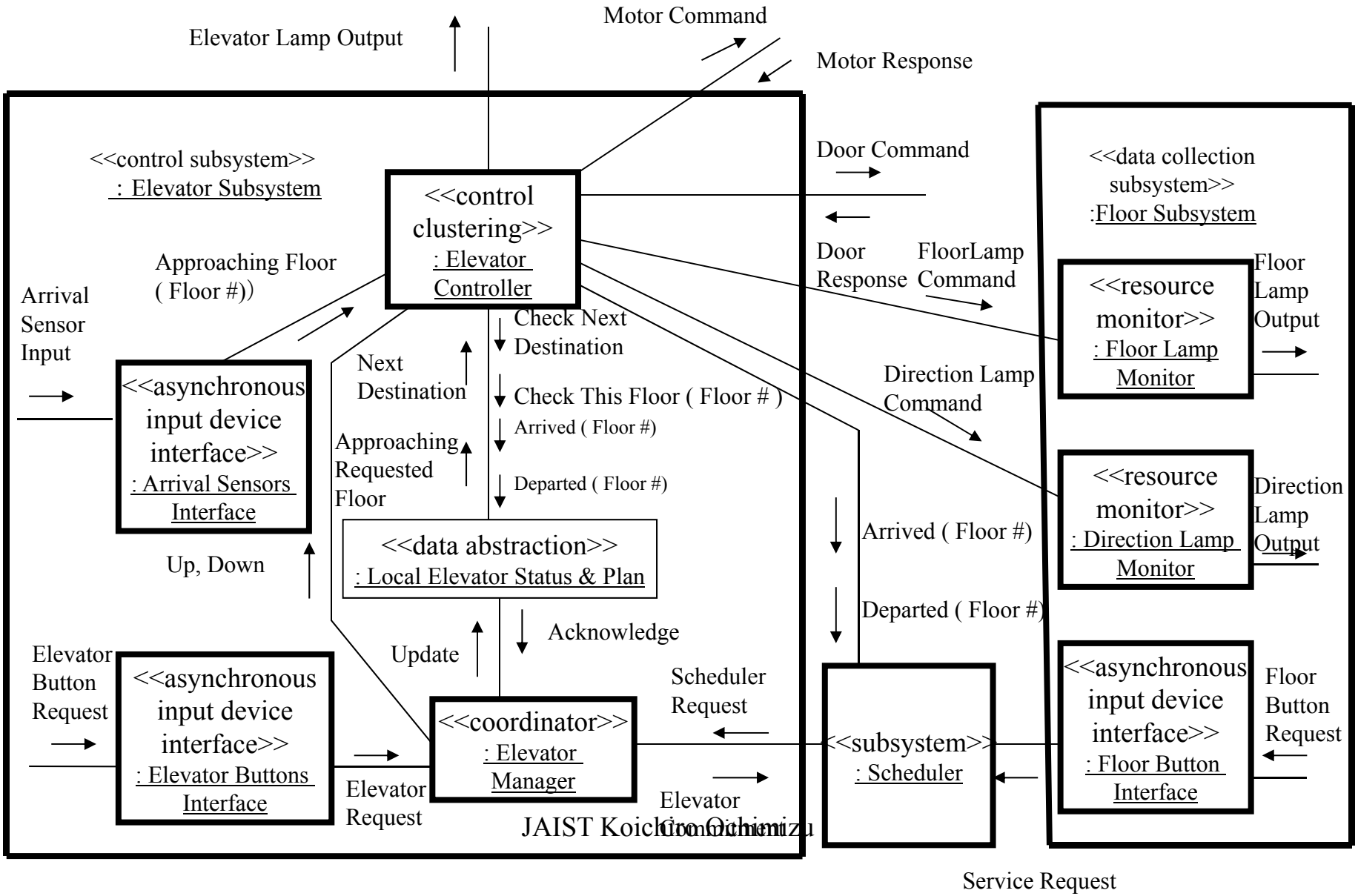


JAIST Koichiro Ochimizu

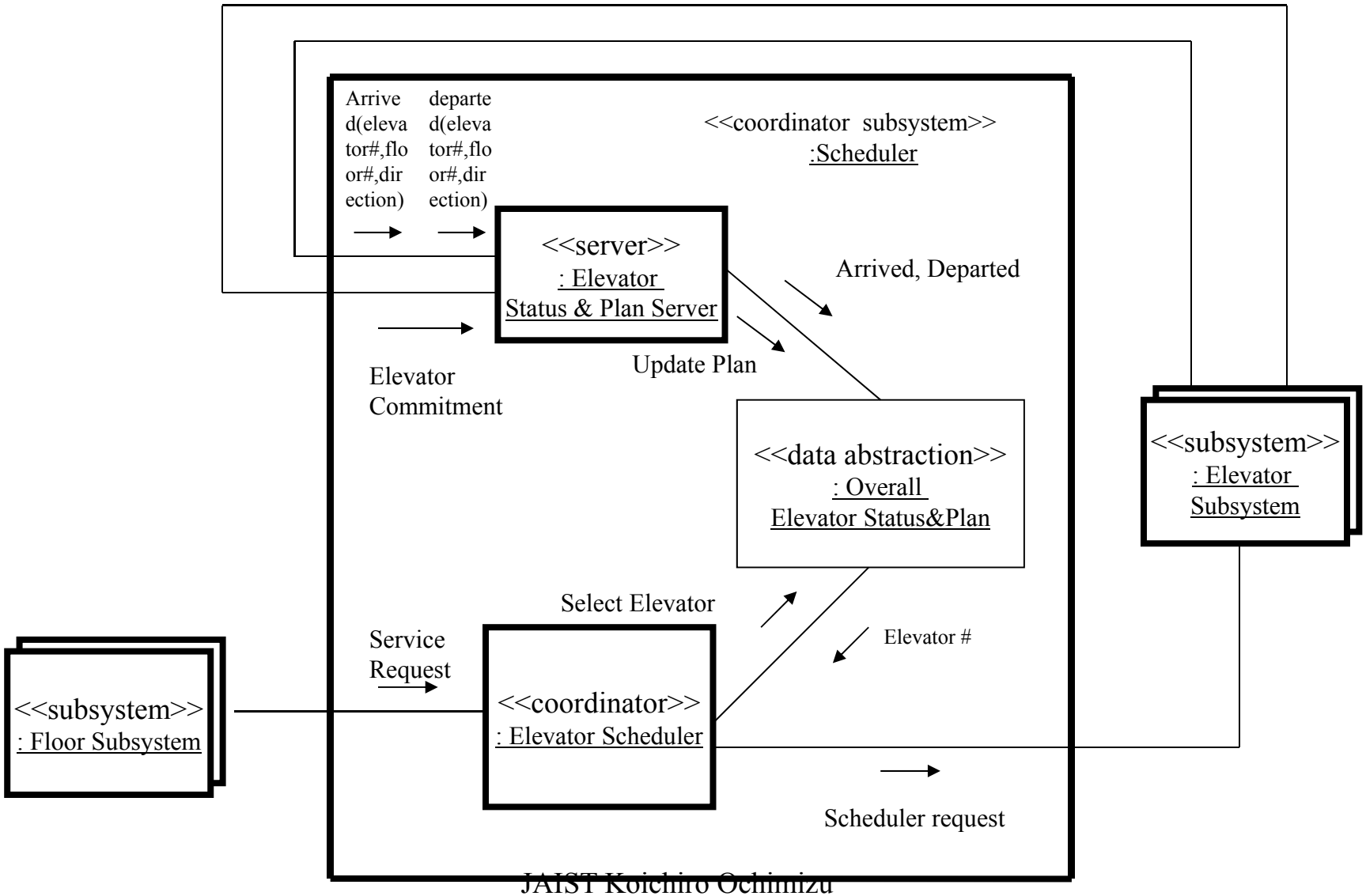
Distributed Software Architecture



Task Architecture (Distributed)



Task Architecture of Scheduler Subsystem



Content(2)

- **Object-oriented Software Development Methodology**
  - Outline of Unified Process and Use-case Driven Approach
  - Elevator Control System:
    Problem Description and Use-case Model
  - Elevator Control System:
    Finding of Problem Domain Objects
  - Elevator Control System:
    Sub-System Design and Task Design
  - Elevator Control System:
    Performance Evaluation
- **Product Line Technology**
  - Feature modeling
- **Aspect Oriented Software Design**
- **Contribution of OOT in Software Engineering**
  - History of SE Technologies and Contribution of OOT in SE field

Performance Analysis of Non- Distributed Elevator Control System

- A building with 10 floors and three elevators
- The Worst-Case Scenario
  - Elevator button interrupts arrive with a maximum frequency of 10 times a second, which represents a minimum inter-arrival time of 100 msec. This worst-case scenario assumes that all 30 buttons are pressed within 3 seconds.
  - Floor button interrupts arrive with a maximum frequency of 5 times a second, which represents a minimum inter-arrival time of 200 msec. There are 18 floor buttons ($2 \times 8 + 1 + 1$). This worst-case scenario assumes that all 30 buttons are pressed within 3 seconds.
  - All three elevators are in motion and arrive at floors simultaneously. Three floor-arrival interrupts arrive within 50 msec of each other. This is the most time-critical aspect of the probem, because when a floor arrival interrupt is received, the Elevator Controller has to determine whether the elevator should stop at this floor or not. If it need to stop, the controller must stop the elevator before the floor has been past.
  - Related usecases are “Select Destination”, “Request Elevator”, and “Stop Elevator at Floor”.

Event Sequences in Three UseCases

- Stop Elevator at Floor (Period = Ta)
  - A1: The Arrival Sensors Interface receives and process the interrupt
  - A2: The Arrival Sensors Interface sends “approaching Floor” message to the Elevator Controller.
  - A3: The Elevator Controller receives message and checks the Elevator Status & Plan object to determine whether the elevator should stop or not.
  - A4: The Elevator Controller invokes “stop Motor” operation if it should stop.

Event Sequences in Three UseCases

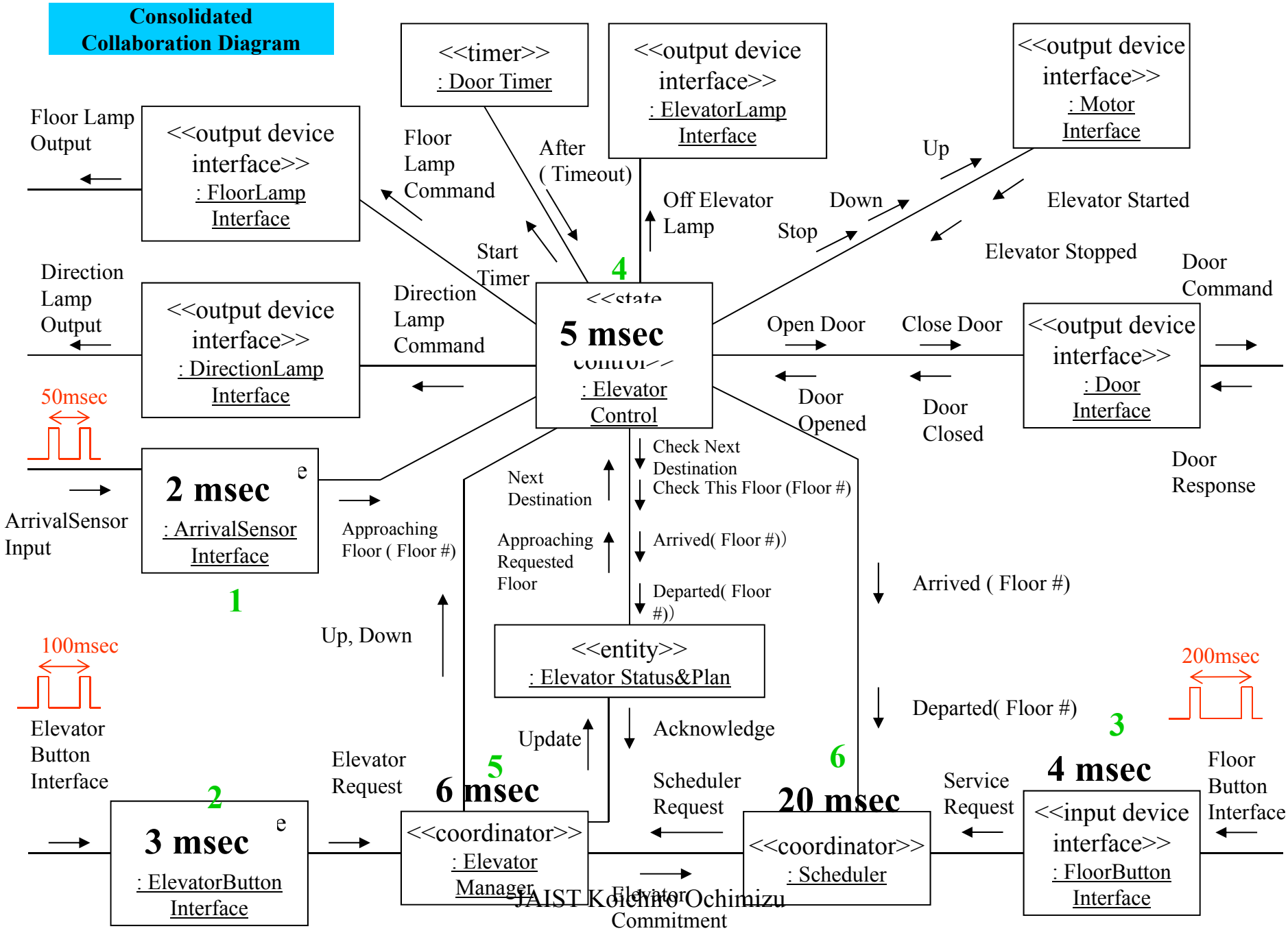
- Select Destination (Period = Tb)
  - E1: The Elevator Buttons Interface receives and processes the interrupt.
  - E2: The Elevator Buttons Interface sends “elevator Request” message to the Elevator Manager.
  - E3: The Elevator Manager receives message and records destination in Elevator Status & Plan object.

Event Sequences in Three UseCases

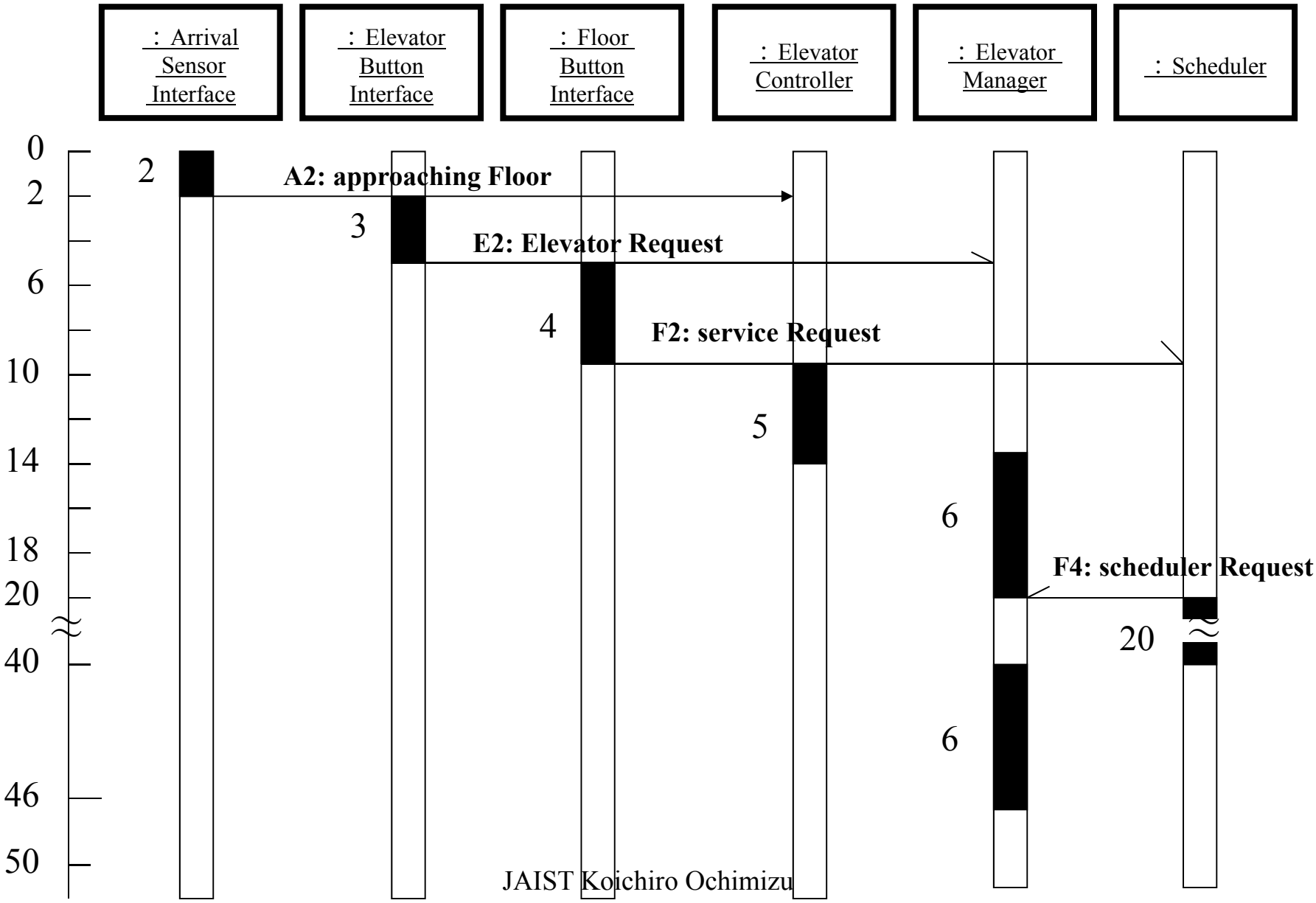
- Request Elevator (Period = Tc)
  - F1: The Floor Buttons Interface receives and processes the interrupt.
  - F2: The Floor Buttons Interface sends “service Request” message to the Scheduler.
  - F3: The Scheduler receives message and interrogates Elevator Status & Plan object to determine whether an elevator is on its way to this floor. Assume not, so that the Scheduler selects an elevator.
  - F4: The Scheduler sends a “scheduler Request” message identifying the selected elevator to the Elevator Manager.
  - F5: The Elevator Manager receives message and records destination in Elevator Status & Plan object.

Task Parameters

Task	CPU time Ci	Period Ti	Utilization Ui	Assigned
Stop Elevator at Floor				Priority
Arrival Sensors Interface	2	50	0.04	1
Elevator Controller	5	50	0.10	4
Total elapsed time = 34 msec			Total utilization = 0.68	
Select Destination				
Elevator Buttons Interface	3	100	0.03	2
Elevator Manager(Case b)	6	100	0.06	5
Total elapsed time = 47 msec			Total utilization = 0.47	
Request Elevator				
Floor Buttons Interface	4	200	0.02	3
Scheduler	20	200	0.10	6
Elevator Manager(Case c)	6	200	0.03	
Total elapsed time = 76msec			Total utilization = 0.38	
Other Tasks				
Floor Lamps Monitor	5	500	0.01	7
Direction Lamps Monitor	5	500	0.01	8


Consolidated Collaboration Diagram
<<timer>>
: Door Timer
<<output device interface>>
: ElevatorLamp Interface
<<output device interface>>
: Motor Interface
Floor Lamp Output
<<output device interface>>
: FloorLamp Interface
Floor Lamp Command
After (Timeout)
Off Elevator Lamp
Up
Down
Stop
Elevator Started
Elevator Stopped
Start Timer
4
Direction Lamp Output
<<output device interface>>
: DirectionLamp Interface
Direction Lamp Command
5 msec
: Elevator Control
Open Door
Close Door
<<output device interface>>
: Door Interface
Door Command
Door Opened
Door Closed
Door Response
50msec
2 msec
: ArrivalSensor Interface
ArrivalSensor Input
Approaching Floor (Floor #)
1
Check Next Destination
Check This Floor (Floor #)
Next Destination
Approaching Requested Floor
Arrived(Floor #))
Departed(Floor #))
Arrived (Floor #)
Up, Down
<<entity>>
: Elevator Status&Plan
100msec
200msec
Elevator Button Interface
Departed(Floor #)
3
Update
Acknowledge
Elevator Request
5
6 msec
6
20 msec
4 msec
Floor Button Interface
Scheduler Request
Service Request
2
3 msec
: ElevatorButton Interface
<<coordinator>>
: Elevator Manager
<<coordinator>>
: Scheduler
<<input device interface>>
: FloorButton Interface
Elevator Commitment
JAIST Koichiro Ochimizu

Time-annotated sequence diagram



Result of Analysis(1/3)

- Stop elevator at Floor (Ta=50msec)
  - Execution time: Total execution time Ca＝2msec (Arrival Sensors Interface)+5msec (Elevator Controller). Execution utilization Ue=Ca/Ta=7/50=0.14
  - Preemption by higher priority tasks: Total preemption time Pa=3 (Elevator Buttons Interface)+4(Floor Buttons Interface)=7msec。 Preemption utilization Up=Pa/Ta =7/50=0.14
  - Blocking time: Total worst-case blocking time Ba= 20msec (Scheduler). Worst-blocking utilization Ub = Ba/Ta=20/50=0.40
  - Total Utilization= Ue+Up+Ub=0.14+0.14+0.40=0.68

Result of Analysis(2/3)

- Select Destination (Tb=100msec)
  - Execution time: Total execution time Cb＝3msec (Elevator Buttons Interface)＋6msec(Elevator Manager). Execution Utilization Ue=Ca/Ta=9/100=0.09
  - Preemption by higher priority tasks with shorter periods: Arrival Sensors Interface and Elevator Controller (preempts Elevator Manager) can each execute twice during 100 msec period, giving a preemption time of 14 msec.
  - Preemption by higher priority tasks with longer periods: 4 msec from Floor Buttons Interface to handle floor Button interrupt.
  - Total preemption time Cp=14+4=18 。 Total preemption utilization Up= Cp/Tb = 18/100 = 0.18
  - Blocking time: Worst-case blocking time Ba= 20msec (Scheduler). Worst-case blocking utilization Ub = Bb/Tb=20/100=0.20
  - Total Utilization= Ue+Up+Ub=0.09+0.18+0.20=0.47

Results of Analysis(3/3)

- Request Elevator (Tc=200msec)
  - Execution time: Total execution time Cc＝4msec (Floor Buttons Interface)＋20msec(Scheduler)＋6msec(Elevator Manager). Execution utilization Ue=Cc/Ta=30/200=0.15
  - Preemption by higher priority tasks with shorter periods: Arrival Sensors Interface and Elevator Controller (preempts Elevator Manager and Scheduler) can each execute four times for a total of 28 msec.
  - Total preemption time Cp=28+18=46. Preemption utilization Up=Cp/Tp=0.23
  - Total elapsed time=30+46+0=76
  - Total Utilization= Ue+Up=0.15+0.23=0.38

Performance Analysis of Distributed Elevator Control System

- one node per elevator, one node per floor, one scheduler node.
- 40 floors and 12 elevators

Task Parameters

Task	CPUtime Ci	Period Ti	Utilization Ui	Assigned Priority
Elevator Subsystem				
Arrival Sensors Interface	2	50	0.04	1
Elevator Controller	5	50	0.10	3
Elevator Buttons Interface	3	100	0.03	2
Elevator Manager	6	100	0.06	4
Floor Subsystem				
Floor Buttons Interface	4	200	0.02	1
Floor Lamps Monitor	5	500	0.01	7
Direction Lamps Monitor	5	500	0.01	8
Scheduler Subsystem				
Elevator Status and Plan Server	2	10	0.20	1
Elevator Scheduler	20	50	0.40	2

Network transmission delay = 2 msec

Elapsed time for Stop Elevator at Floor = 41 msec < 50 msec

Elapsed time for Select Destination = 48 msec < 100 msec

Elapsed time for Request Elevator = 82 msec < 200 msec

Event Sequences for three Usecases

- Stop Elevator at Floor (Period = Ta)
  - A1: The Arrival Sensors Interface receives and process the interrupt
  - A2: The Arrival Sensors Interface sends “approaching Floor” message to the Elevator Controller.
  - A3: The Elevator Controller receives message and checks the ***Local*** Elevator Status & Plan object to determine whether the elevator should stop or not.
  - A4: The Elevator Controller invokes “stop Motor” operation if it should stop.
  - A5: The Elevator controller sends arrived message over the LAN to the Scheduler subsystem, where it is received by the Elevator Status & Plan Server.
  - A6: The Elevator Status & Plan Server calls the arrived operation of the *Overall* Elevator Status & Plan data abstraction object.

Event Sequences for three Usecases

- Select Destination (Period = Tb)
  - E1: The Elevator Buttons Interface receives and processes the interrupt.
  - E2: The Elevator Buttons Interface sends “elevator Request” message to the Elevator Manager.
  - E3: The Elevator Manager receives message and records destination in ***Local*** Elevator Status & Plan object.
  - E4: The Elevator Manager sends an “elevator Commitment” message over the LAN to the Scheduler subsystem, where it is received by the Elevator Status & Plan Server.
  - E5: The Elevator Status & Plan Server calls the update Plan operation of the ***Overall*** Elevator Status & Plan data abstraction object.

Event Sequences for three Usecases

- Request Elevator (Period = Tc)
  - F1: The Floor Buttons Interface receives and processes the interrupt.
  - F2: The Floor Buttons Interface sends "service Request" message over the LAN to the Elevator Scheduler task in the Scheduler subsystem.
  - F3: The Elevator Scheduler receives message and interrogates ***Overall*** Elevator Status & Plan object to determine whether an elevator is on its way to this floor. Assume not, so that the Scheduler selects an elevator.
  - F4: The Elevator Scheduler sends a "scheduler Request" message identifying the selected elevator over the LAN to the Elevator Manager task in the selected elevator's instance of the Elevator subsystem
  - F5: The Elevator Manager receives message and records destination in the ***Local*** Elevator Status & Plan object.
  - F6: The Elevator Manager sends an "elevator Commitment" message over the LAN to the Scheduler subsystem, where it is received by the Elevator Status & Plan Server
  - F7: The Elevator Status & Plan Server calls the update Plan operation of the Overall Elevator Status & Plan data abstraction object.

Result of Analysis(1/3)

- Stop elevator at Floor (Ta=50msec)
  - Execution time: Total execution time Ca＝2msec (Arrival Sensors Interface)＋5msec (Elevator Controller) Execution utilization Ue= Ca/Ta=7/50=0.14
  - Preemption by higher priority tasks with longer periods. Preemption time Pa=3 (Elevator Buttons Interface). Preemption utilization Up=3/50=0.06
  - Blocking time: Total worst-case blocking time Ba= 6msec (Elevator Manager). Worst-case blocking utilization Ub = Ba/Ta=6/50=0.12
  - Total elapsed time=7＋3＋6 =16msec<50. Total Utilization= Ue+Up+Ub=0.14+0.06+0.12=0.32
  - Total elapsed time for Stop elevator at Floor＝16(Total elapsed time for Elevator Subsystem)＋2 (Transmission Delay, 25byte, 100Mbau, Transmission Delay Dt = 200/100000 = 2msec＋23 Worst-case elapsed time of Scheduler system)＝16＋2＋23＝ 41msec

Result of Analysis(2/3)

- Select Destination (Tb=100msec)
  - Execution time: Total execution time Cb＝3msec (Elevator Buttons Inteface)＋6msec(Elevator Manager). Execution utilization Ue=Ca/Ta=9/100=0.09
  - Preemption time by higher priority tasks with shorter periods: Arrival Sensors Interface and Elevator Controller can each execute twice during the 100 msec period, giving a total preemption time of 14 msec. Up=0.14.
  - Total elapsed time =9+14=23.　Total utilization = Ue+Up =0.09+0.14=0.23
  - Total elapsed time for Select Destination Eb =23 (Elevator subsystem elapsed time)＋2(transmission delay)＋23 (Scheduler subsystem elapsed time) =48msec.

Result of Analysis(3/3)

- Request Elevator (Tc=200msec)
  - Total elapsed time for floor subsystem Ef=4msec(floor buttons Interface)＋1msec(transmission delay)=5msec.
  - Total elapsed time for scheduler subsystem Es=1msec(transmission delay)＋20msec(elevator Scheduler)+ 1 msec (transmission delay)＋2msec (Blocking time by Elevator Status & Plan subsystem) ＝1+20+1+2= 24msec.
  - Execution time of Elevator Manager＝1+6+1=8.
  - Total elapsed time of Elevator subsystem＝16+8=24.
  - Total elapsed time for Request Elevator ＝ 5+2+24+2+24+2+23=82