

# Elevator Control System

**Koichiro Ochimizu**  
**School of Information Science**  
**Japan Advanced Institute of Science and Technology**

## Schedule(3/3)

- March 12
  - 13:00 Unified Process and COMET
  - 14:30 Case Study of Elevator Control System  
(problem definition, use case model)
- March 13
  - 13:00 Case Study of Elevator Control System  
(finding analysis classes by developing a consolidated communication diagram)
  - 14:30 **Case Study of Elevator Control System**  
**(sub-system structuring and task structuring)**
- March 14
  - 13:00 Case Study of Elevator Control System  
( performance analysis)
  - 14:30 UML2.0 and MDA

## 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

## Component and Node

- A distributed application is structured into distributed subsystems
- A subsystem is designed as a configurable component and corresponds to a logical node.
- Thus a subsystem component is defined as a collection of concurrent tasks executing on one logical node.
- More than one subsystem component (logical node) may execute on the same physical node.
- The term "component" will be used to refer to a logical node and the term "node" will be used to refer to a physical node.

## Configurable Architectures and Software Components

- Configurable Architecture
  - An important goal of software architecture for a distributed application is to provide a concurrent message-based design that is highly configurable.
  - In other words, the same software architecture should be capable of being mapped to many different system configuration.
  - Thus, a given application could be configured to have each subsystem allocated to its own separate physical node, or alternatively to have all or some of its subsystem allocated to the same physical node.
  - The decision about mapping subsystems to physical nodes is not made at design time, but is made later, at system configuration time.
- Distributed Component
  - A distributed component is an active object with a well-defined interface.
  - A component is usually a composite object composed of other objects
- Steps in Designing Distributed Applications
  - System Decomposition  
Subsystems are defined by applying subsystem structuring criteria. All communication between subsystems must be restricted to message communication.
  - Subsystem Decomposition  
Tasks are defined by applying task structuring criteria to each subsystem.
  - System Configuration  
The components instances of the target system are defined, interconnected, and mapped onto a hardware configuration consisting of distributed physical nodes.

## System Decomposition

- Consider the use-case-based collaboration diagrams, which show the objects that communicate frequently with each other. This is generally a good basis for subsystem structuring because objects that communicate frequently with each other are good candidates for being in the same subsystem. The object depicted on several collaboration diagrams can only be part of one subsystem.
- If one object is located at a geographically remote location from another object, the two objects should be in different subsystems ( for an example, clients and servers)
- Objects that are part of the same composite object should be in the same subsystem.
- On the other hand, objects in a aggregate subsystem grouped by functional similarity might span geographical boundaries

## Distributed Component Configuration Criteria

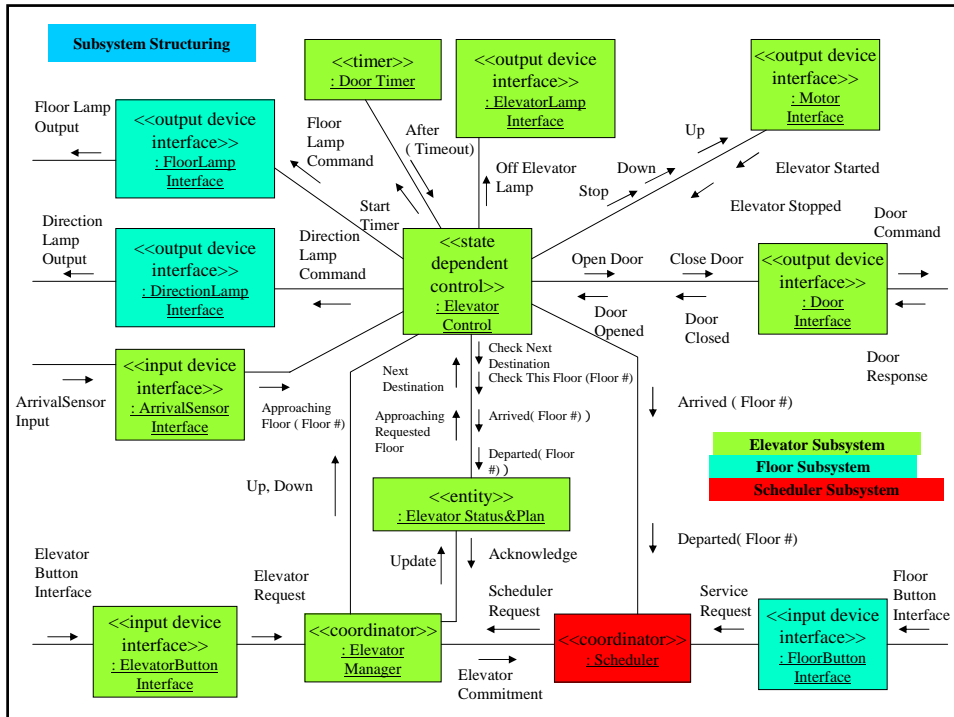
- **Proximity to the source of physical data**  
Provide proximity of the component to the source of physical data to ensure fast access to the data, particularly important if data access rates are high.
- **Localized autonomy**  
The component controls a given aspect of the system to do a specific site-related service.
- **Performance**  
A real-time component can perform a time-critical service at a given node, with non-real-time or less time-critical services performed elsewhere.
- **Specialized Hardware**  
A component might need to reside on a particular node because it supports special-purpose hardware like vector processors.
- **User Interface**  
A component providing a user interface might run on a separate node.
- **Server**  
A server component is often allocated its own node.

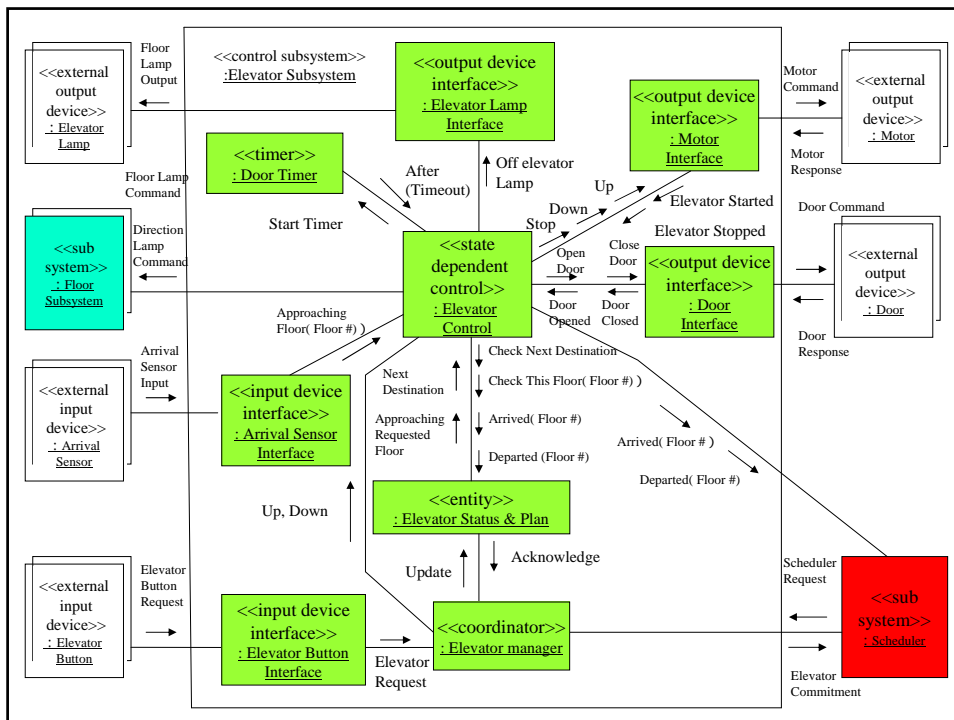
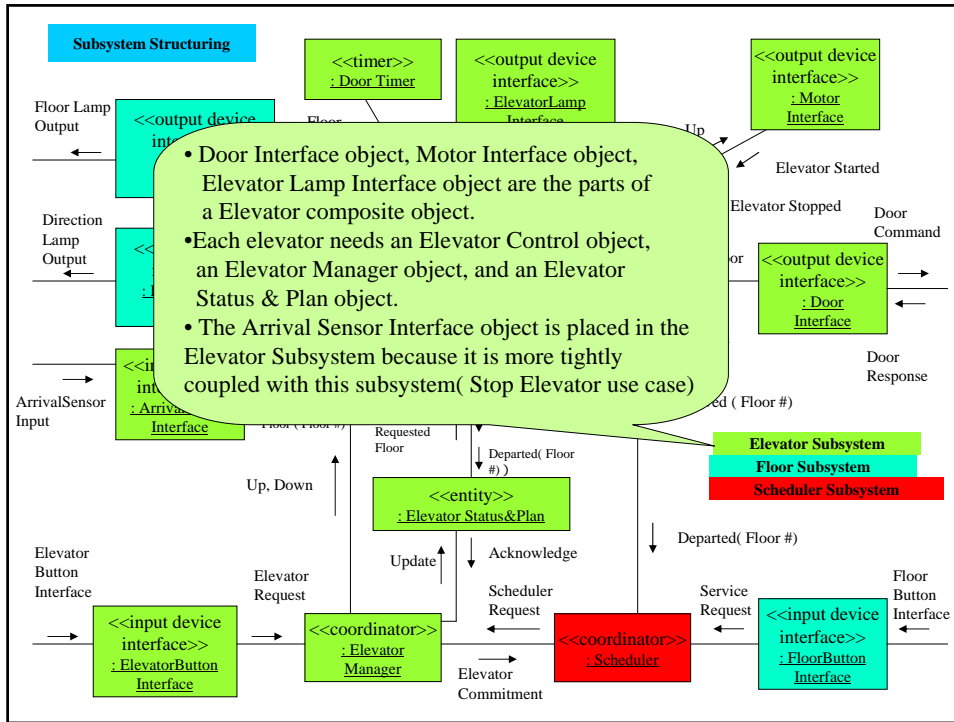
## Type of subsystems for concurrent, real-time, or distributed application domains

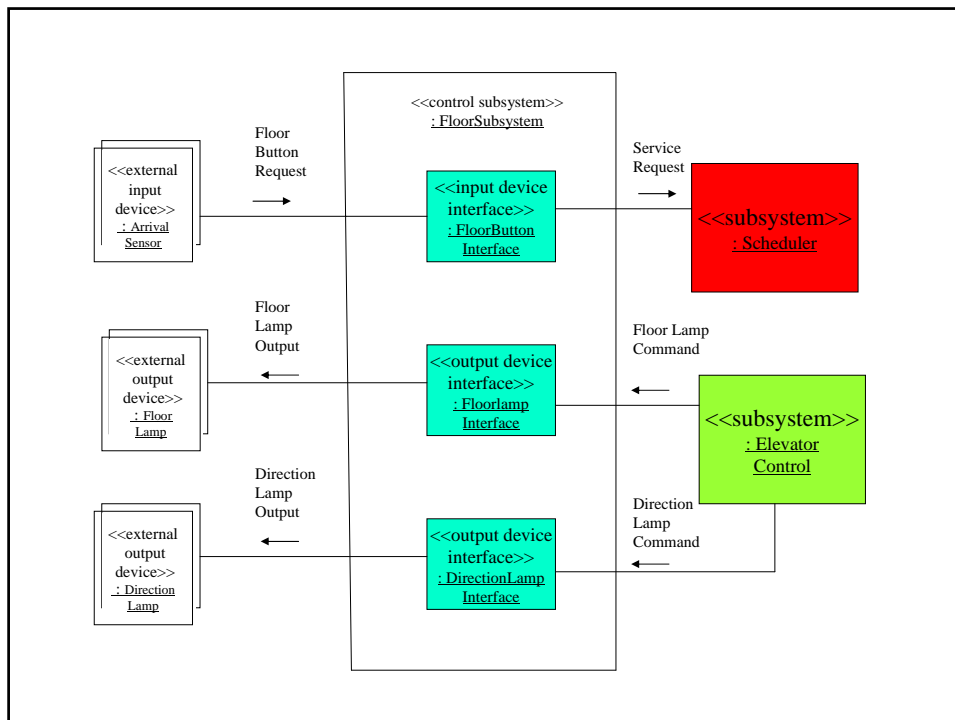
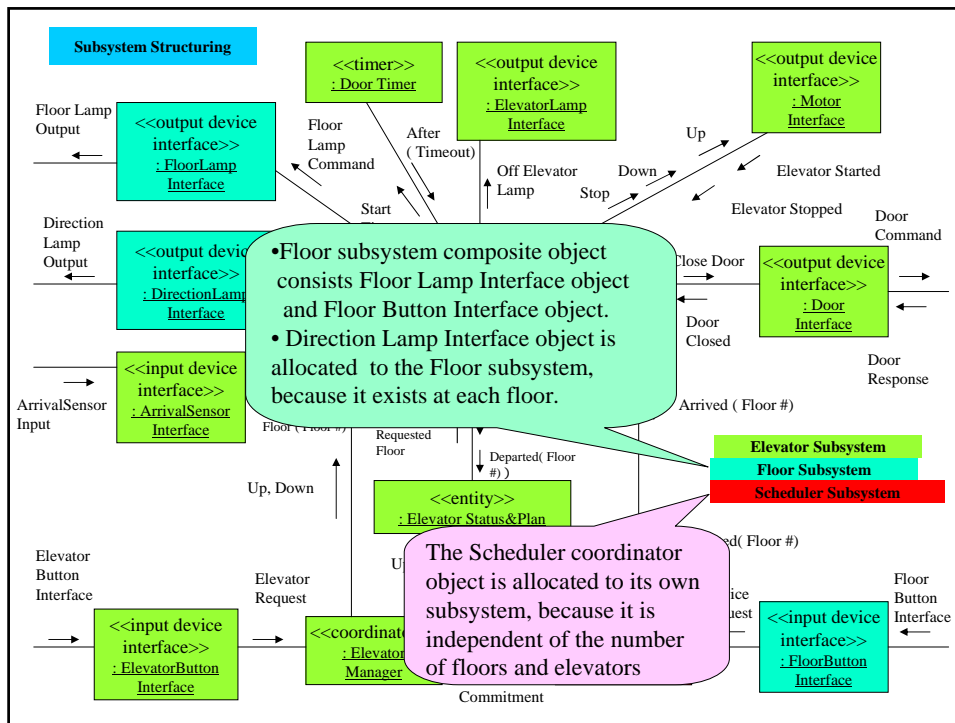
- **<<Control>>**  
The subsystem receives its inputs from the external environment and generates outputs to the external, usually without any human intervention. It includes at least one state-dependent control object. In some case, some Inputs data might be gathered by some other subsystem (s).
- **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

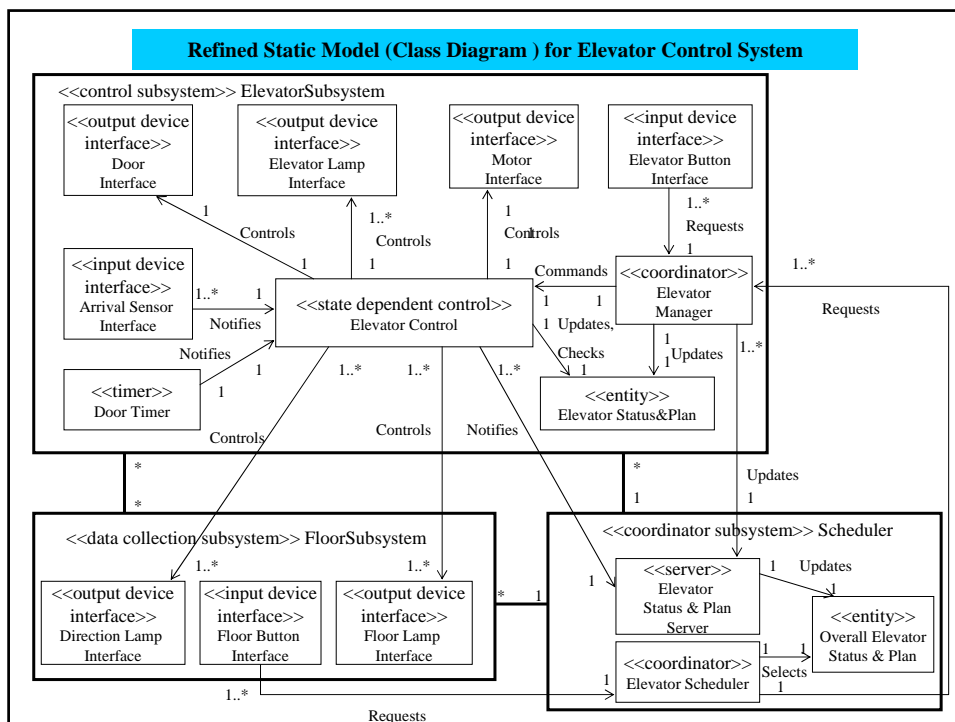
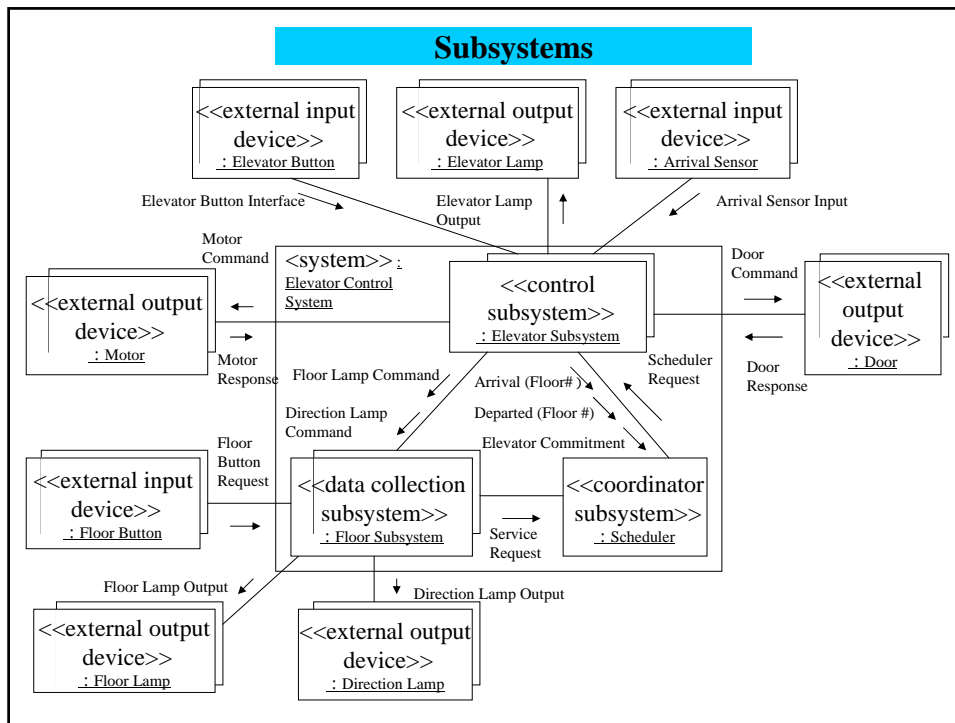
## Type of subsystems for concurrent, real-time, or distributed application domains

- **<<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 all 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.









## 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.

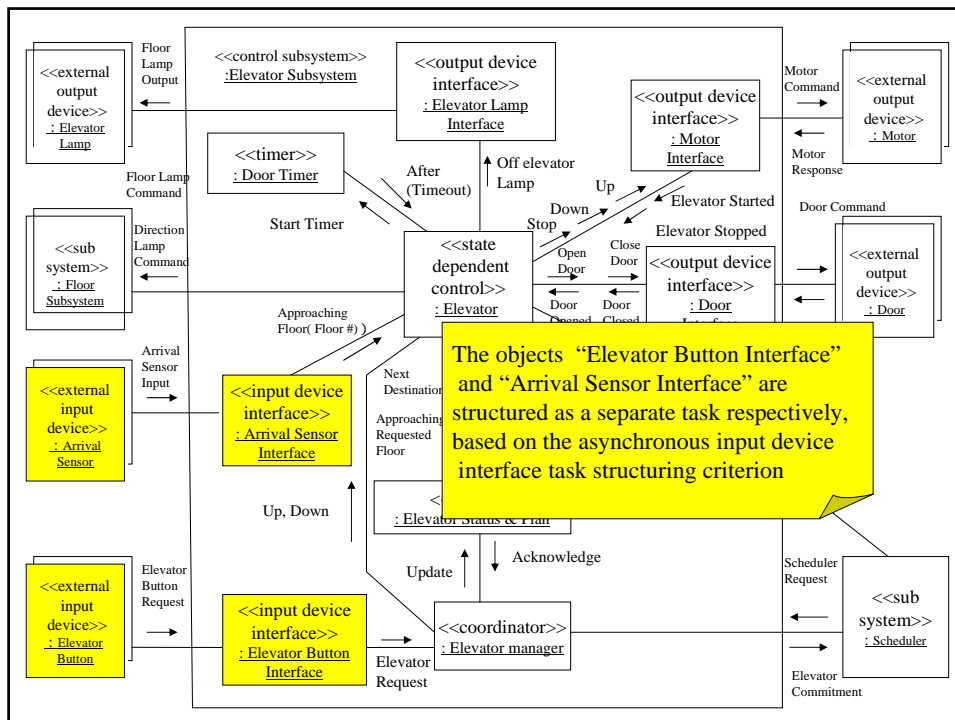
The activity should be structured as a separate task.

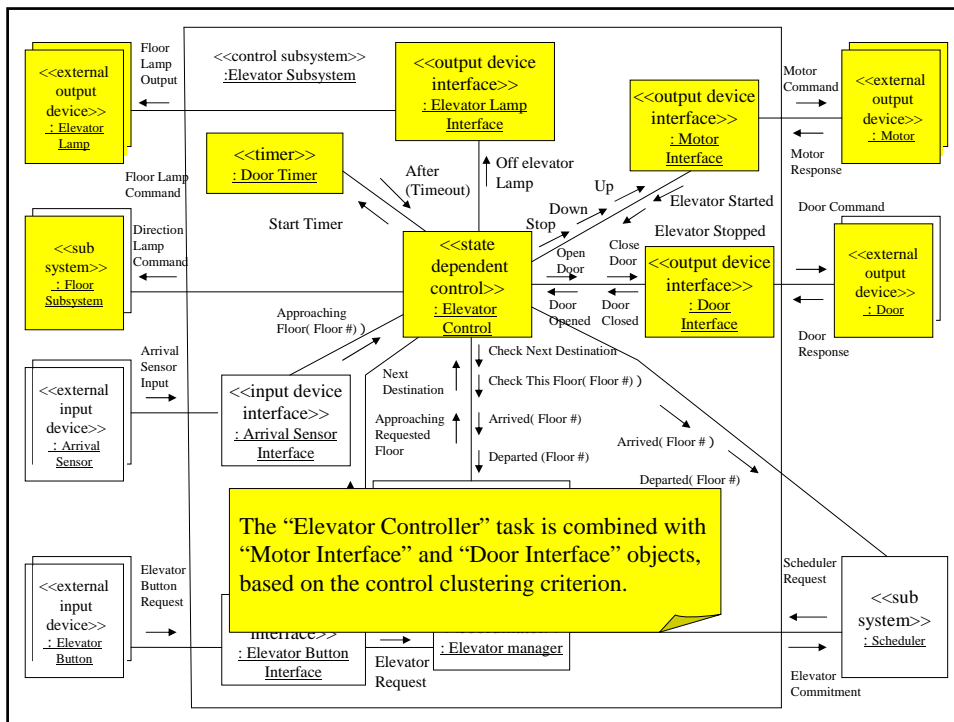
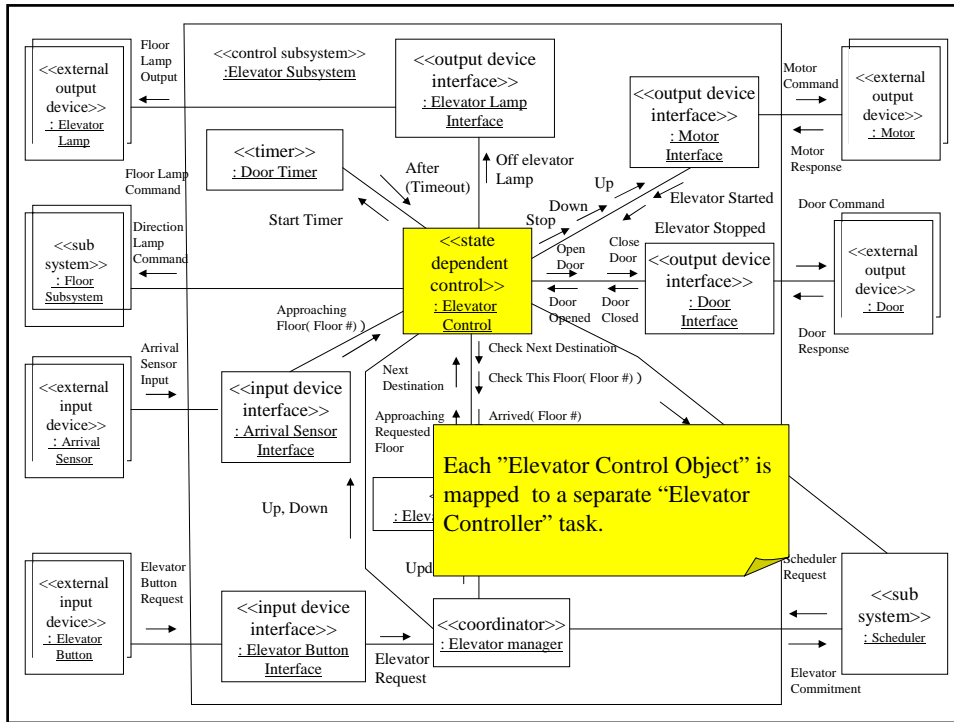
- **Mutually Exclusive Clustering**

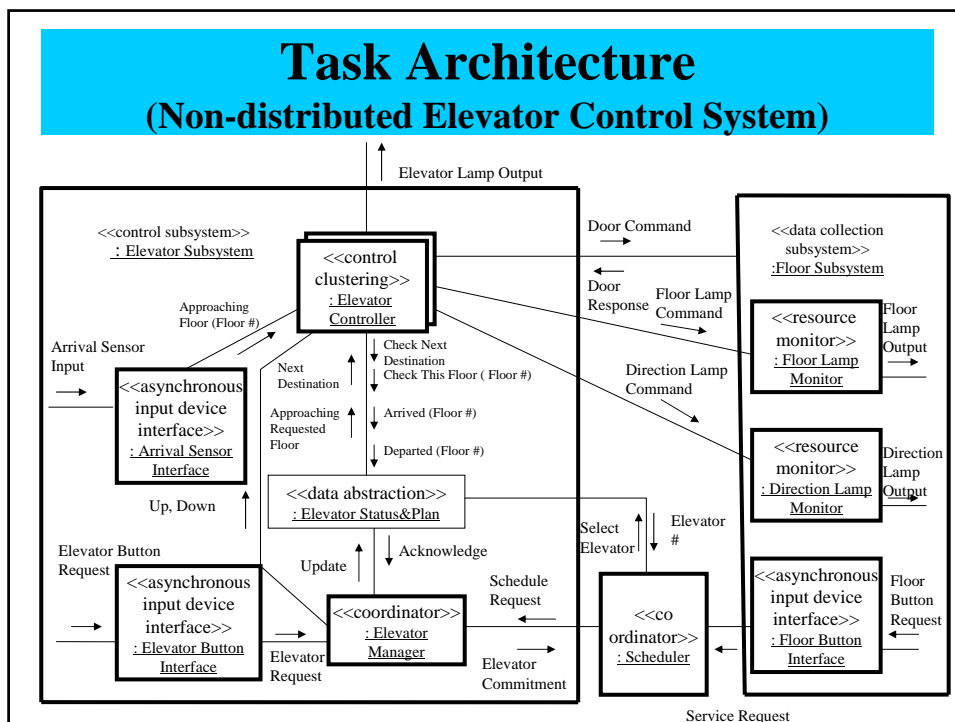
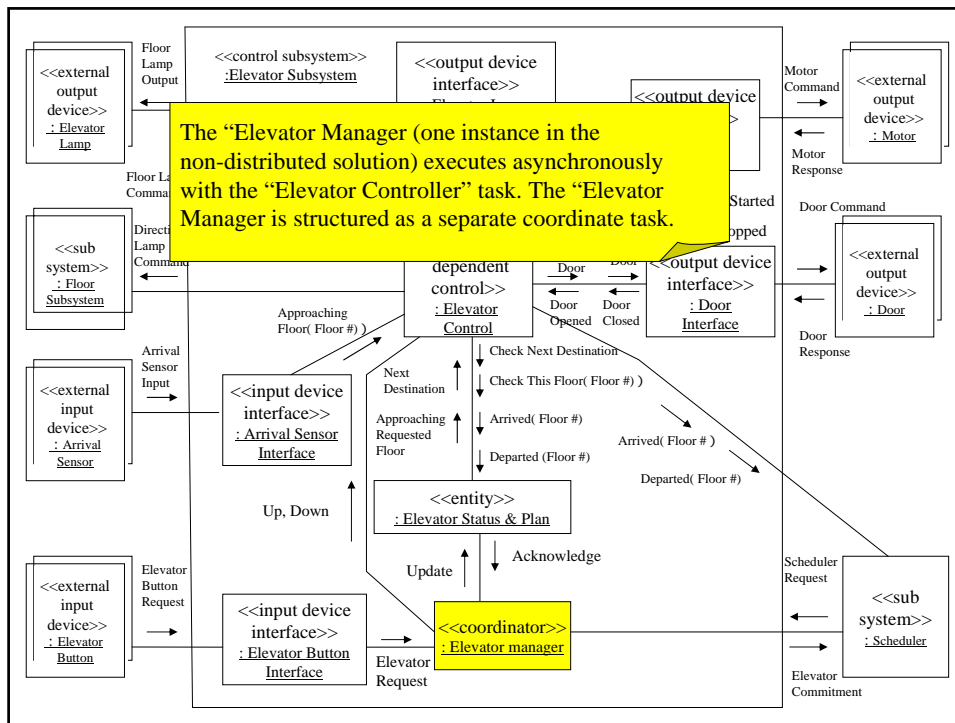
Mutually exclusive tasks may be clustered.

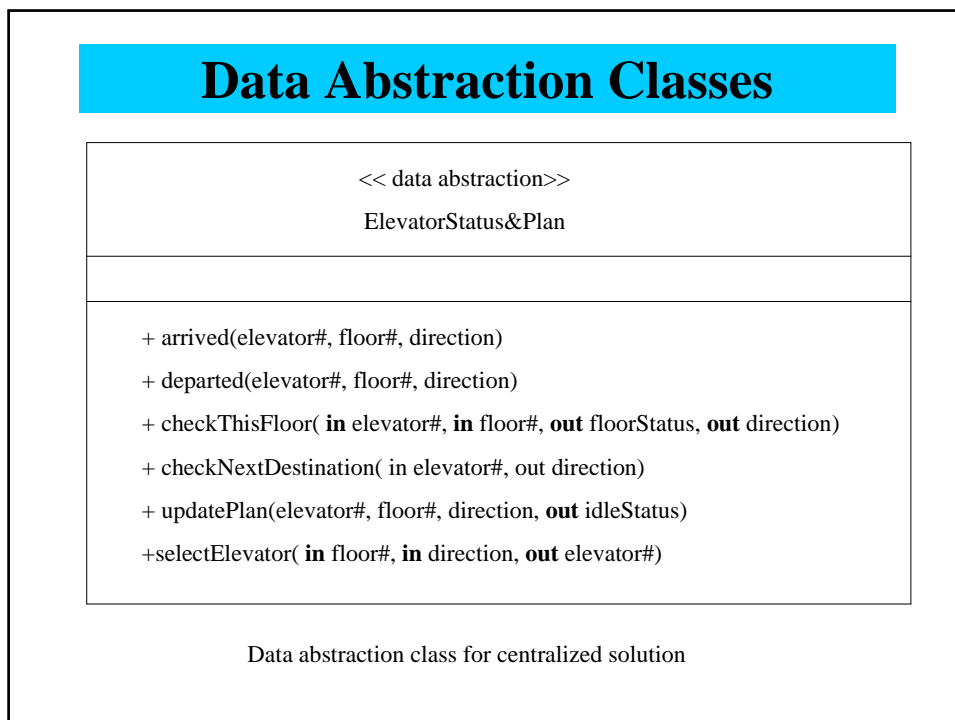
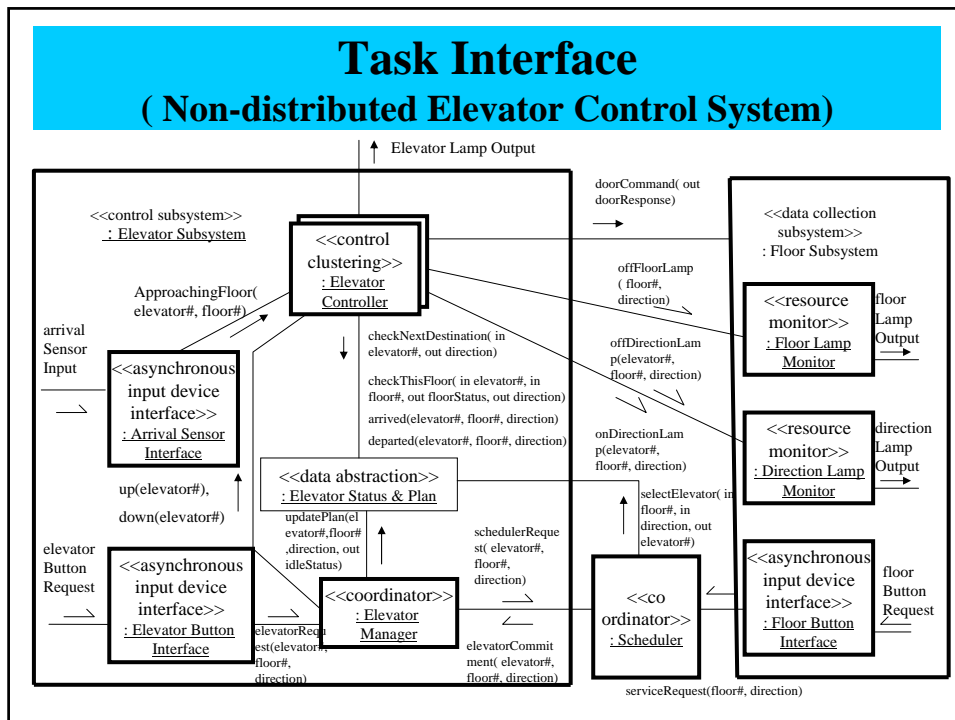
## Non-Distributed Solution

- 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









## 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 crating 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

