COMET BASED ELEVATOR CONTROLLER SYSTEM CASE STUDY

Brief System Description (Gomaa, 2000):

The system controls the motion of multiple elevators and responds to passenger requests at various floors:

- Each elevator has a set of destination buttons and a set of floor lamps.
- Each elevator has a hoist motor which responds to direction commands: up, down and stop, and returns a status.
- Each elevator has a door motor which may be controlled to open or close and returns a status.
- Each floor (apart from the top and bottom floors) has up and down floor buttons and corresponding lamps.
- Each floor (apart from the top and bottom floors) has two direction lamps to indicate which way the elevator is heading.
- The top and bottom floor have a single down and up floor button and corresponding lamp, respectively.
- Each floor has an elevator arrival sensor.

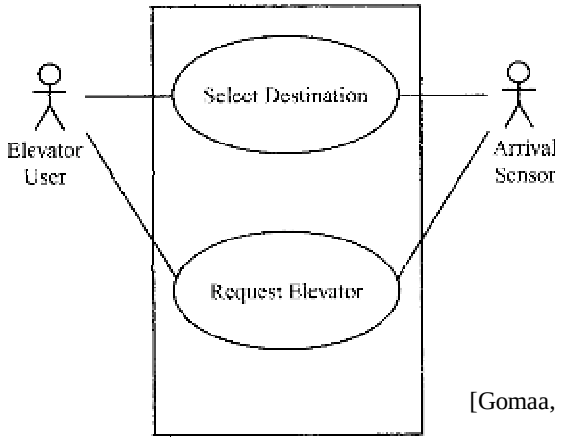
I/O characteristics:

- All buttons and sensors generate an interrupt, i.e. are asynchronous inputs.
- The elevator and floor lamps are switched on by hardware, but must be switched off by software.
- The direction lamps are fully software controlled.

To do:

1. ***Produce the Use Case model for the Elevator Control System.***
2. ***Produce the Elevator Control System Context Diagram.***
3. ***Produce the Statechart Diagram segment for the Stop Elevator at Floor included use case.***
4. ***Produce the Task Architecture Diagram based on the consolidated collaboration diagram and subsystem structuring.***

Use case model for the Elevator Control System:



[Gomaa, 2000]

Request Elevator Use Case

Actors:

Elevator User, Arrival Sensor

Precondition:

User is at a floor and wants 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 elevator should move in order to service the next request.
3. As the elevator moves between floors, the arrival sensor detects 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 stops the motor, and opens the door.
4. 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.

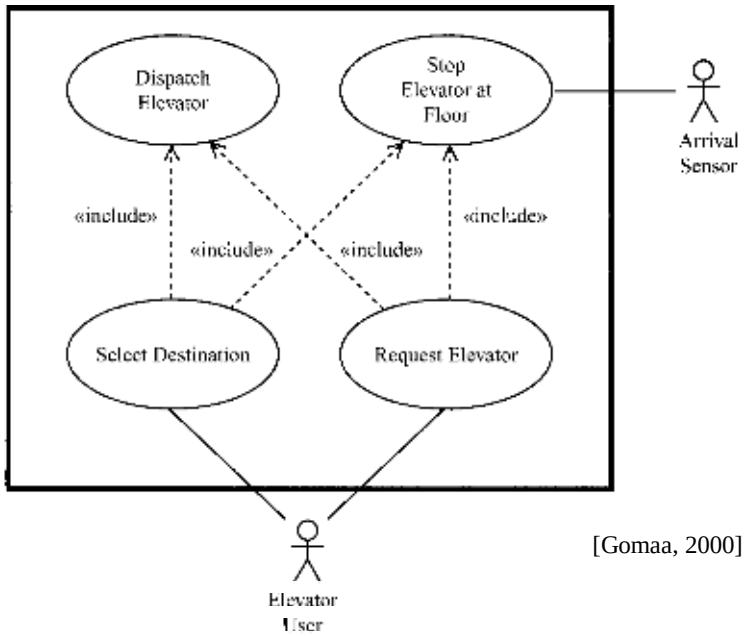
Alternatives:

1. User presses a down floor button, the system response is the same as 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 same floor, with the door open.

Postcondition:

Elevator has arrived at the floor in response to the user request.

If we also consider either use case, we can factor out included use cases in both primary use cases, i.e. in both cases the system must dispatch an elevator to a requested floor and it must stop the elevator at that floor:



[Gomaa, 2000]

Stop Elevator at Floor Included Use Case

Actors:

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 systems checks if the elevator should stop at this floor. If the system commands the motor to stop, and the door to open.

Alternatives:

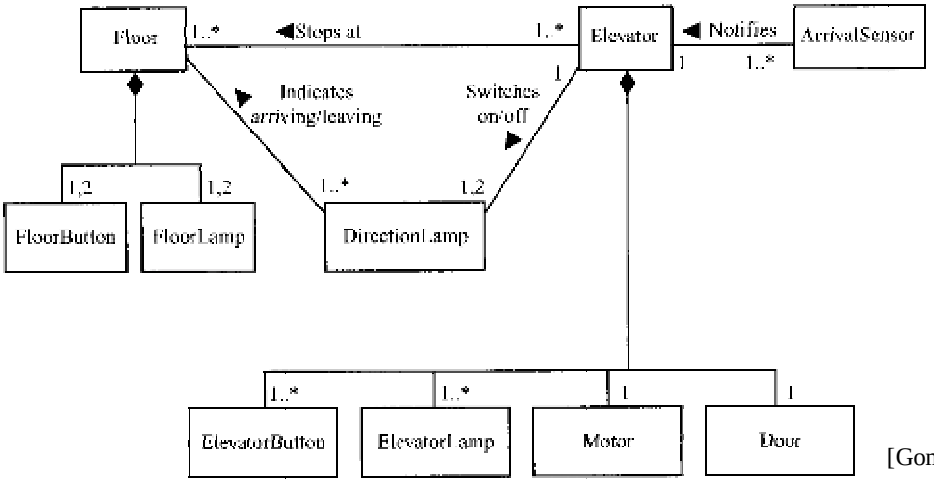
The elevator is not required to stop at this floor and so continues past the floor.

Postcondition:

Elevator has stopped at the floor with the door open.

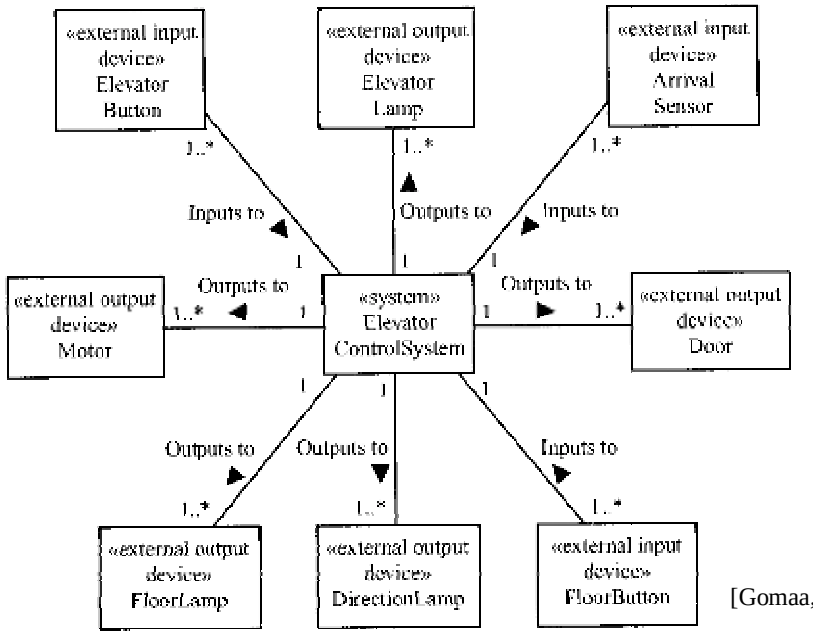
Conceptual Static Class Model and System Context Diagram

The conceptual static class model for the Elevator System shows the composite system classes and their associations:



[Gomaa, 2000]

All external devices are interfaced to the system via software device interface objects so the System Context Diagram shows this mapping:



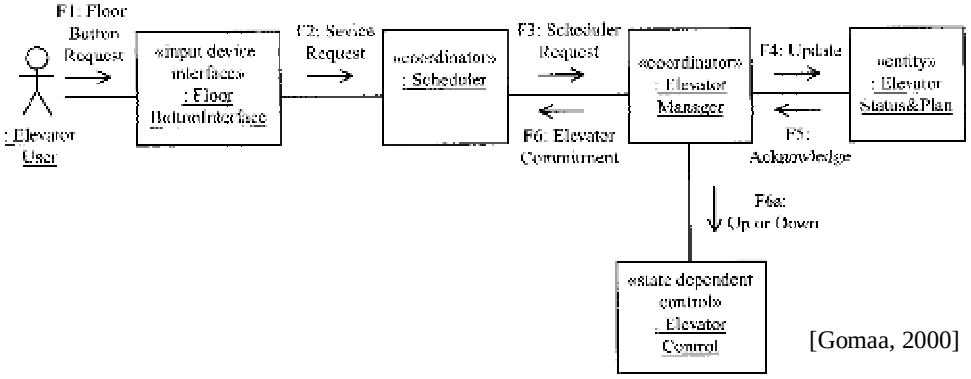
[Gomaa, 2000]

Dynamic Modelling

The dynamic modelling phase maps the use cases into object collaborations. For example, consider the *Request Elevator* use case:

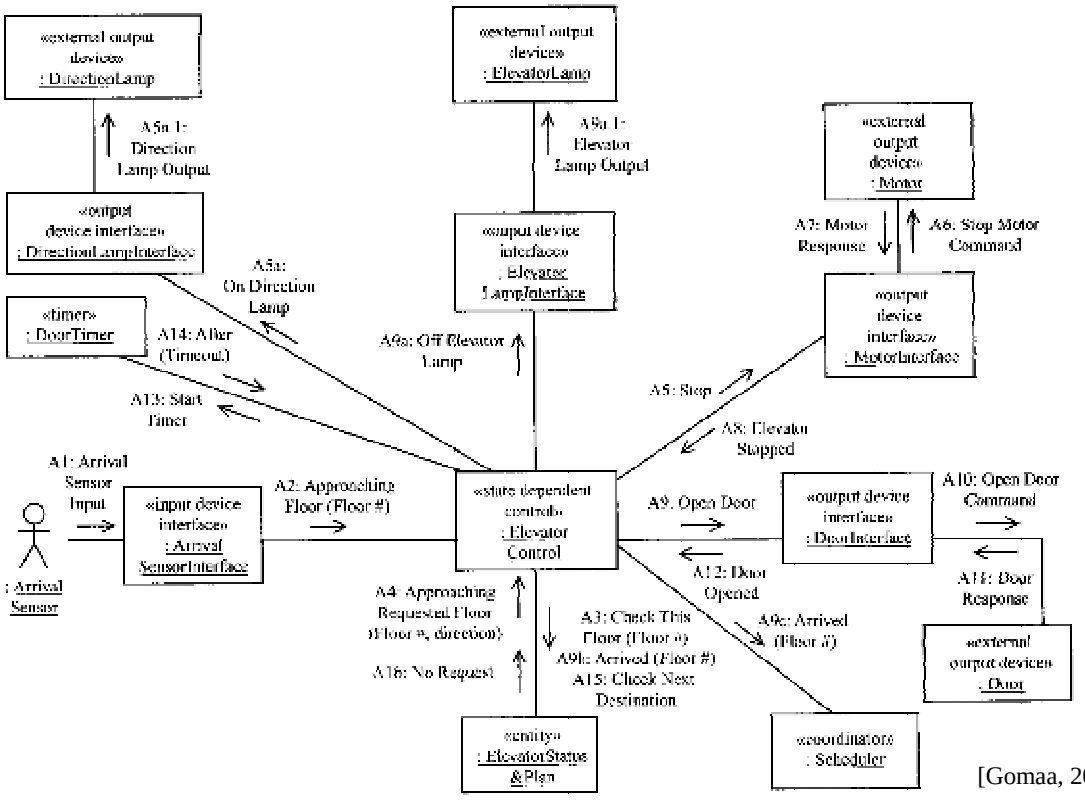
- On a floor button request – must decide which elevator should service the request.
- Scheduler decides – requires status of all elevators and their current plans

Collaboration Diagram – Request Elevator Use Case:



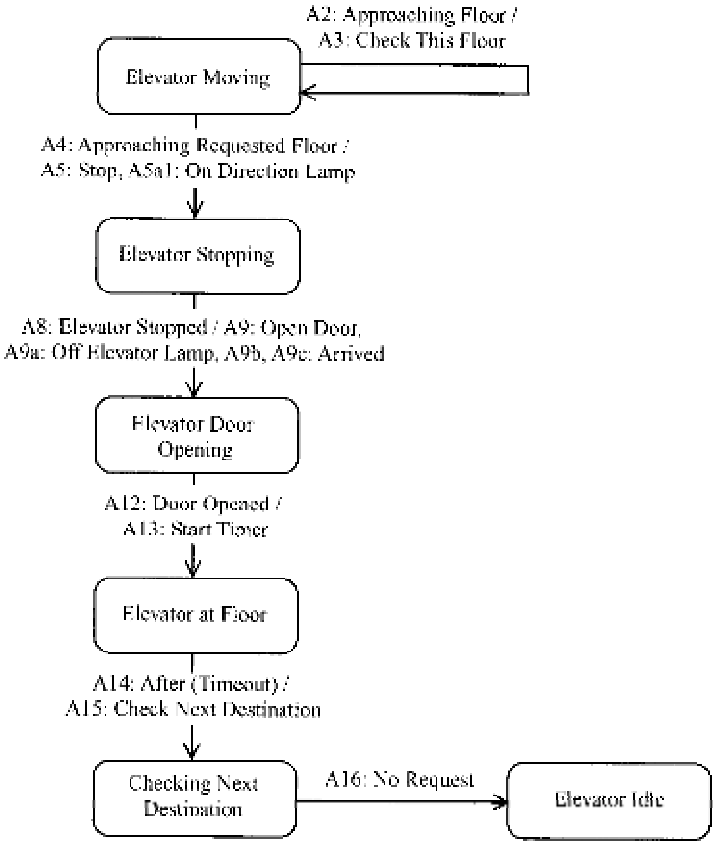
[Gomaa, 2000]

Collaboration Diagram - Stop Elevator at Floor Use Case:



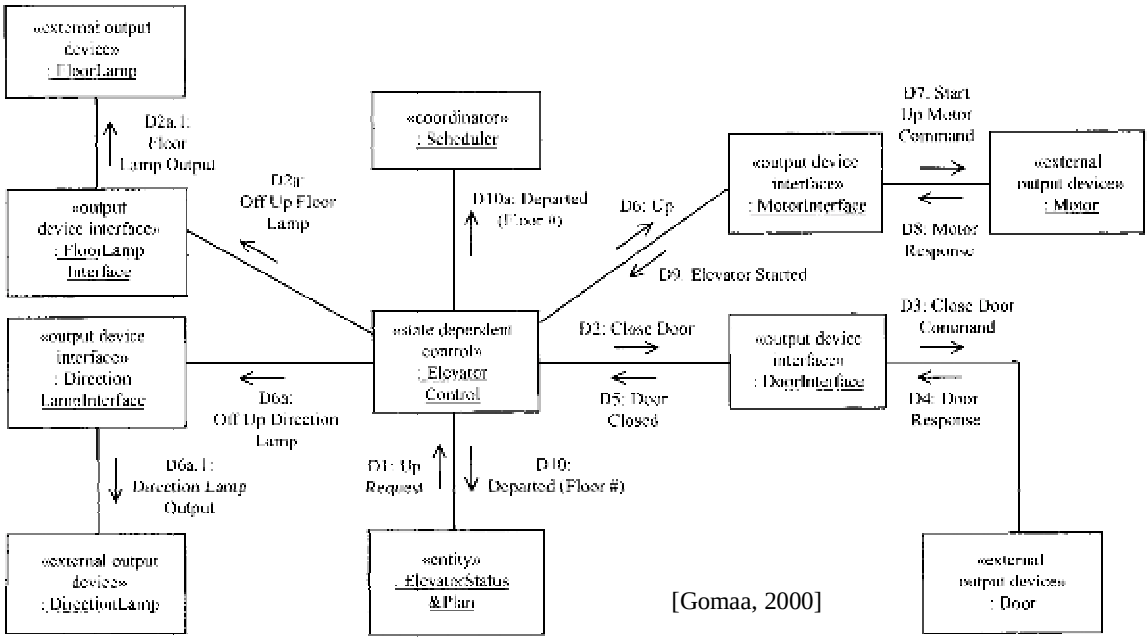
[Gomaa, 2000]

Statechart Diagram – Stop Elevator at Floor Use case:



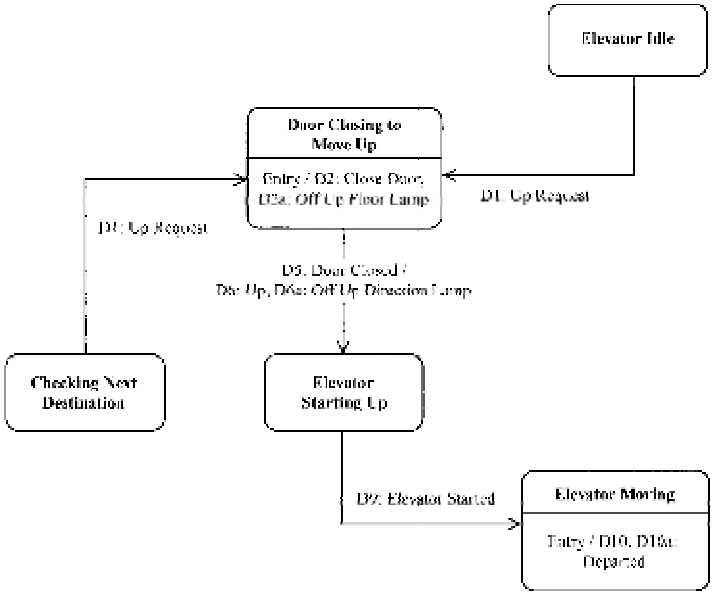
[Gomaa, 2000]

Collaboration Diagram - Dispatch Elevator to Floor Use Case:



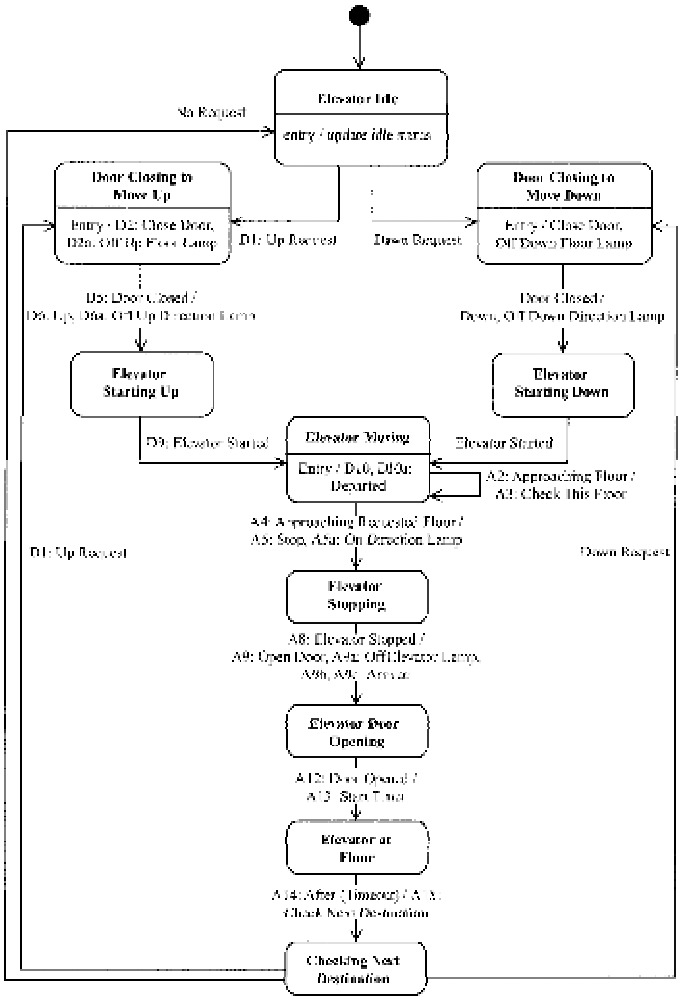
[Gomaa, 2000]

Statechart Diagram – Dispatch Elevator Use case:



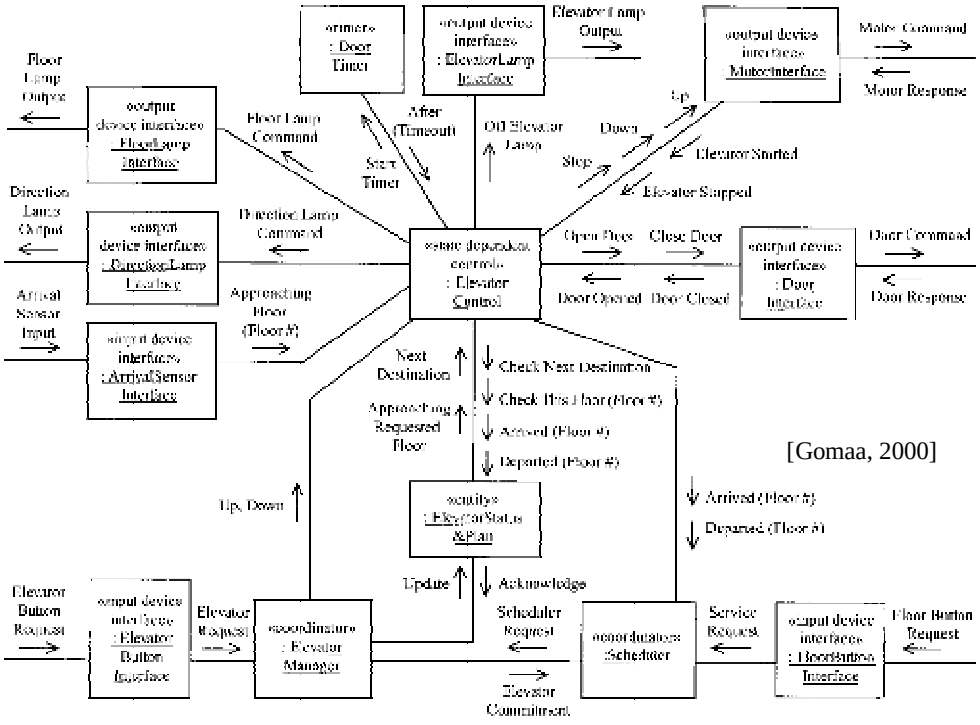
[Gomaa, 2000]

Complete Statechart Model:



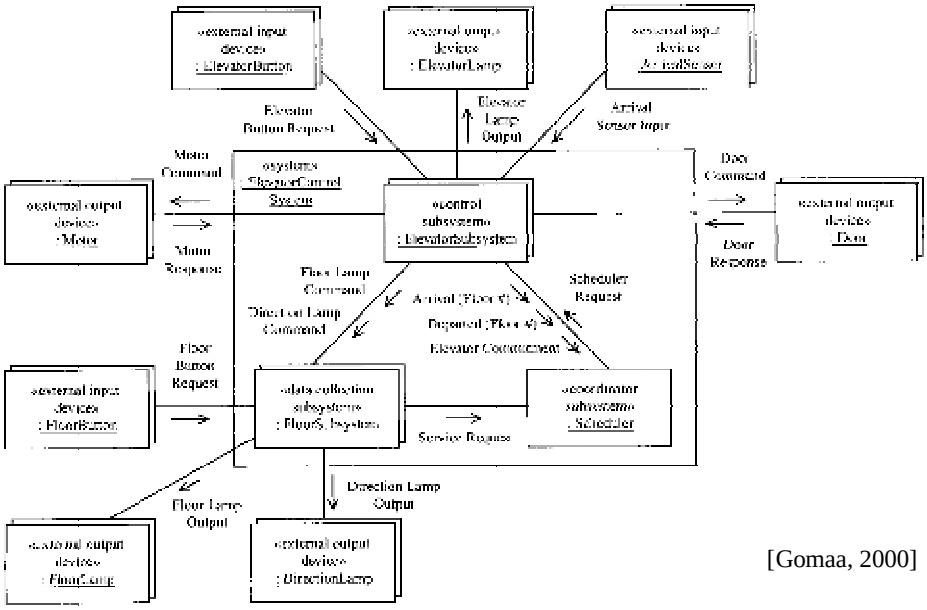
[Gomaa, 2000]

Consolidated Collaboration Diagram



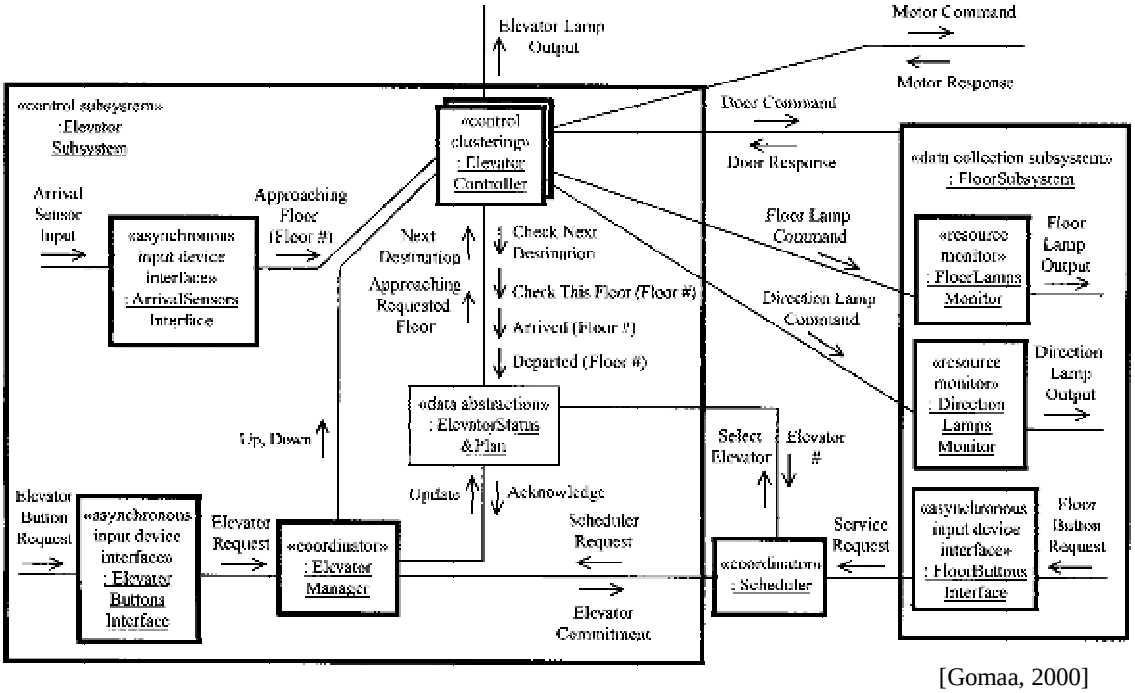
The consolidated collaboration diagram shows all main sequences through the use cases and all alternatives. Message names are usually aggregated for conciseness, e.g. *Direction Lamp Command*.

Subsystem Structuring



Task Structuring

All collaboration diagram objects are analyzed and the task structuring criteria are applied. In the non-distributed case, the *Elevator Status & Plan* entity object is accessible to all elevators. Each set of subsystem objects are mapped to tasks:



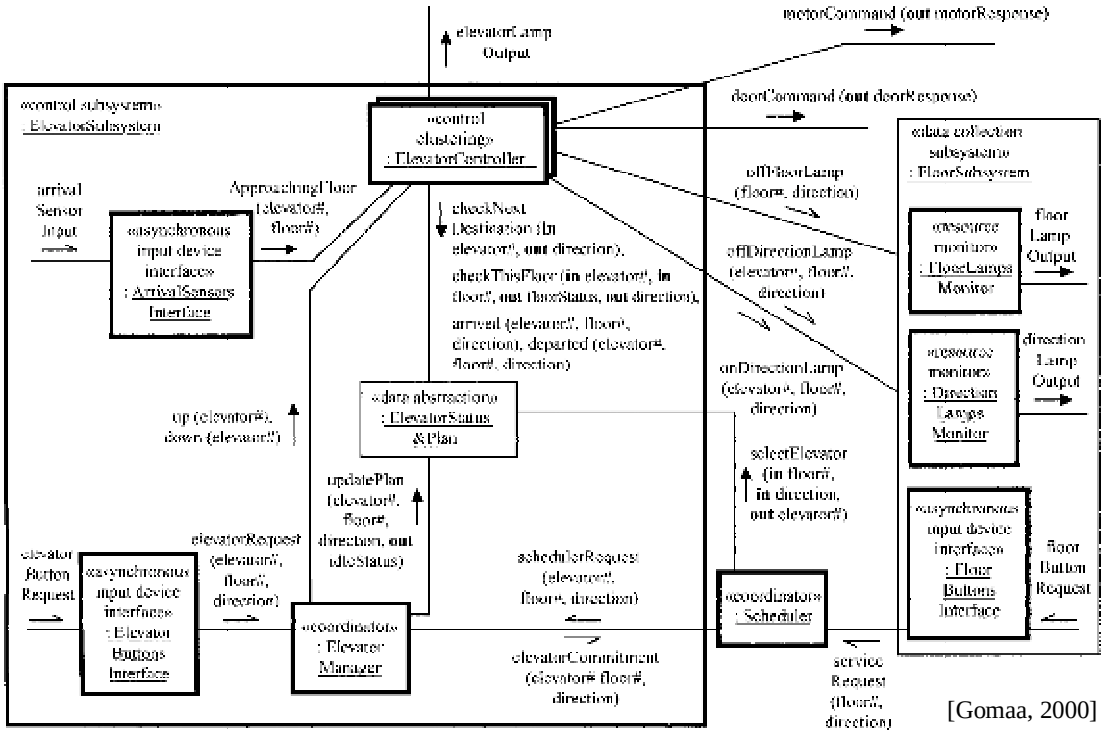
[Gomaa, 2000]

All asynchronous input device interfaces are mapped to asynchronous I/O tasks. *Elevator Manager* is designated as a coordinator task, and *Elevator Controller* is a multiple instance control task. The *Elevator Status & Plan* object is mapped to a passive data abstraction object. As the motor and door output interfaces are passive, and *Elevator Controller* must await a response, no additional I/O tasks are required to interface to these output objects.

Task Interfaces

Consider message interfaces between tasks, e.g:

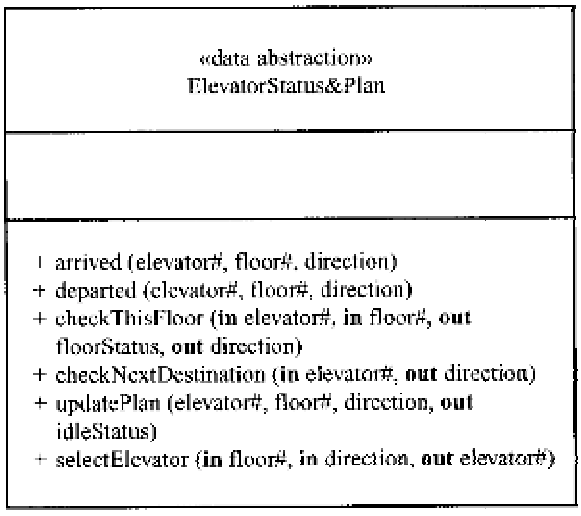
- Loose coupling between *Elevator Buttons Interface* and *Elevator Manager* tasks.
- Tight coupling between *Elevator Manager* and *Elevator Controller* tasks.



[Gomaa, 2000]

Data Abstraction Classes

Only one instance is required in the non-distributed case (*ElevatorStatus &Plan*) which holds the planned commitment for the elevator. The required operations on the class are developed:



[Gomaa, 2000]

Alternatives: Consider mappings for multiple CPUs for the elevator subsystems and floor subsystems – consider the distributed design in the second part of this case study.