

# Route Planning System: MyTrip

- A driver can plan a trip from a home computer by contacting a trip planning service on the Web
- The trip is saved for later retrieval on the server and is reused on an on-board computer in the car
- The trip planning service must support more than one driver

# PlanTrip Use Case

Entry Condition: The `Driver` activates her home computer and logs into the trip planning Web service

## Flow of Events

1. The `Driver` enters constraints for a trip as a sequence of destinations
2. Based on a database of maps, the planning service computes the shortest way visiting the destinations in the specified order. The results is a sequence of segments binding a series of crossings and a list of directions
3. The `Driver` can revise the trip by adding or removing destinations
4. The `Driver` enters a name for the planned trip.

Entry Condition: The planned trip is saved by name in the planning service database for later retrieval

# ExecuteTrip Use Case

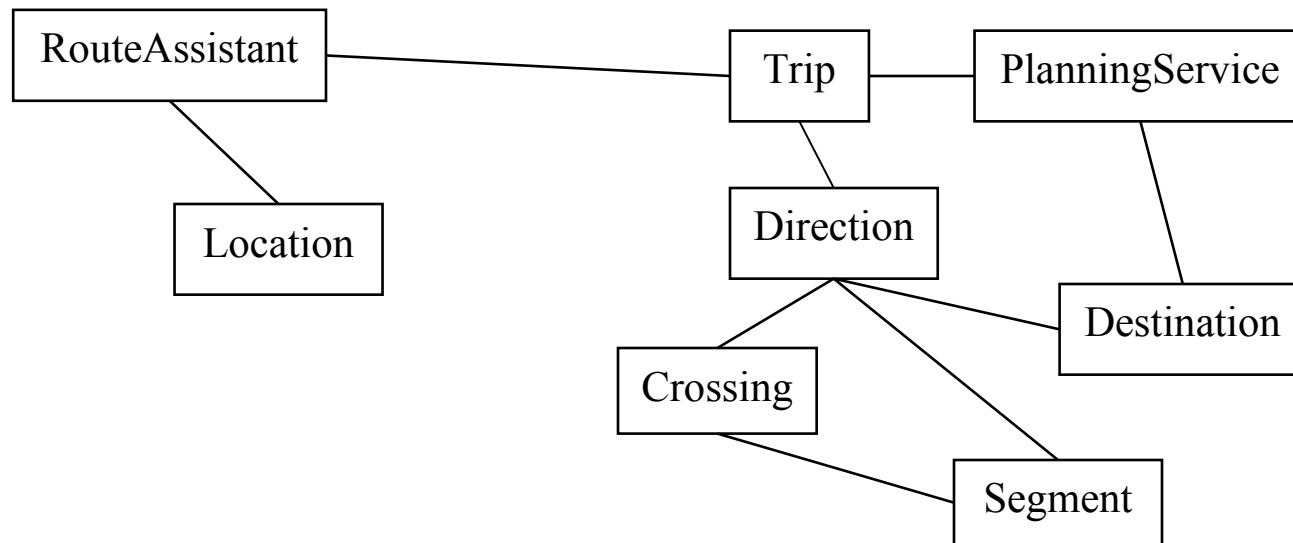
Entry Condition: The *Driver* starts her car and logs into the onboard computer route assistant

## Flow of Events

1. The Driver specifies the planning service and the name of the trip to be executed
2. The onboard route assistant obtains the list of destinations, directions, segments, and crossings from the planning service
3. Given the current position, the route assistant provides the *Driver* with the next set of directions

Exit Condition: The *Driver* arrives to destination and shuts down the route assistant

# Analysis Class Diagram



# Non-functional Requirements

- MyTrip is in contact with a web planningService via a wireless modem.
- MyTrip should give correct directions even if the modem fails to maintain a connection with the PlanningService.
- MyTrip should minimize connection time to reduce operation costs
- Replanning is possible only if the connection to the planningService is possible
- The PlanningService can support at least 50 different drivers and 1000 trips

# Identifying Design goals

- Many design goals can potentially be considered
  - Performance, dependability, end user criteria, cost, maintenance
- Can be inferred from non-functional requirements, application domain, clients, developers
- MyTrip Examples:
  - NF requirements: reliability, fault tolerance to connectivity loss with routine service
  - Application domain: security. Other drivers should not access trips from a driver.
  - Developers: modifiable to use different routing services
- Trade-offs are usually necessary

# SYSC-3020 — Introduction to Software Engineering

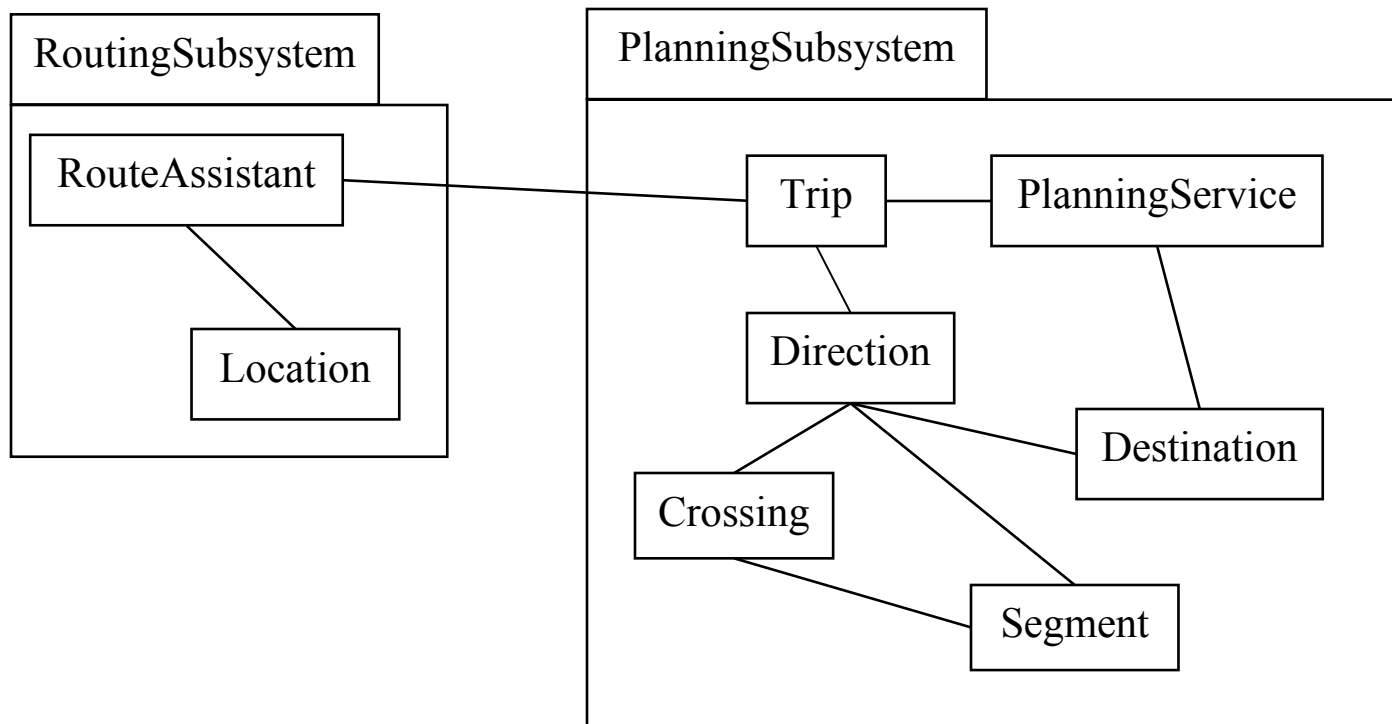
- System design concepts
  - Definitions: Architecture (subsystem decomposition), coupling, cohesion
  - Layers and partitions
  - Software architecture (MVC, Observer pattern)
  - Process architecture (UML notation and Distribution patterns)
- System design process
  - Identify design goals
  - Initial subsystem decomposition
  - Map subsystems to components and processors
  - Persistent storage
  - Define access control policies
  - Select control flow mechanism
  - Identify boundary conditions

# Identifying Subsystems

- Goal: Larger grain information-hiding solution, once subsystem interfaces defined, their design proceed independently
- Based on heuristics and iterative :
  - Subsystems are merged, split. New ones are added.
- Initial decomposition based on functional requirements, ie. objects involved in the use cases
- Heuristics:
  - Assign objects identified in one use case into the same subsystem
  - Create a dedicated subsystem for objects used for moving data among subsystems
  - Minimize interactions: number of associations crossing subsystem boundaries, (distinct) messages being exchanged between subsystems
  - All objects in the same subsystem should be functionally related



# MyTrip Subsystems



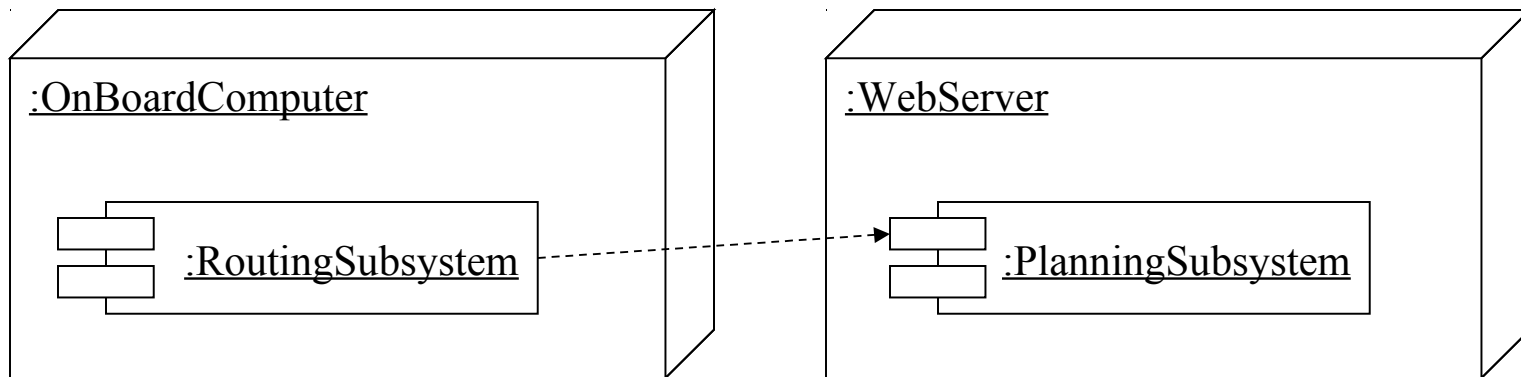
# SYSC-3020 — Introduction to Software Engineering

- System design concepts
  - Definitions: Architecture (subsystem decomposition), coupling, cohesion
  - Layers and partitions
  - Software architecture (MVC, Observer pattern)
  - Process architecture (UML notation and Distribution patterns)
- System design process
  - Identify design goals
  - Initial subsystem decomposition
  - Map subsystems to components and processors
  - Persistent storage
  - Define access control policies
  - Select control flow mechanism
  - Identify boundary conditions

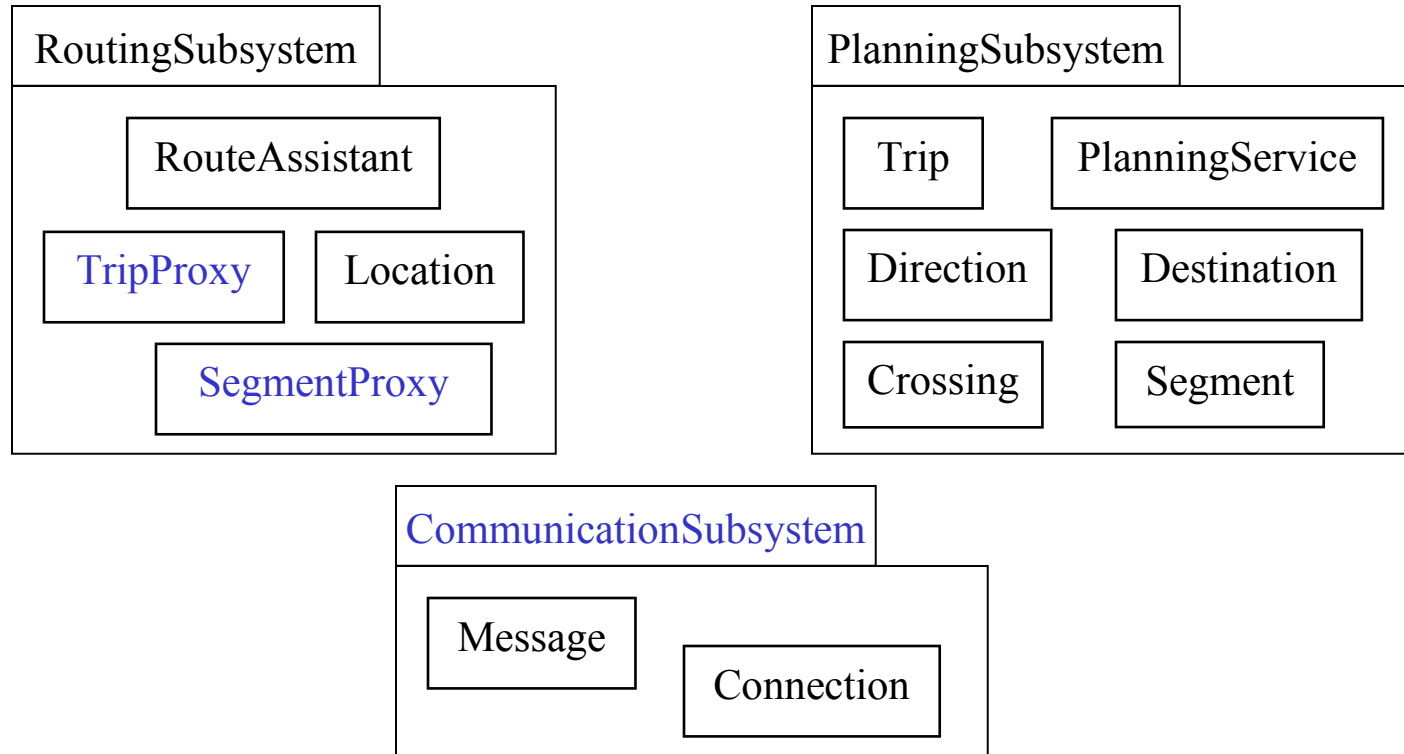
# HW Configuration and Platform

- Many systems run on several computers and depend on access to an intranet or the internet :
  - High-performance needs
  - Interconnect multiple distributed users
- Allocation of subsystems to computers is done early in system design, because
  - Adds complexity and impacts performances
  - Requires communication infrastructure between nodes (potentially new subsystems)
- Subsystem mapping given by :
  - Deployment diagram (HW Configuration)
  - Virtual machine or platform: OS, components (DBMS, communication)

# myTrip Deployment Diagram



# New Classes and Subsystems



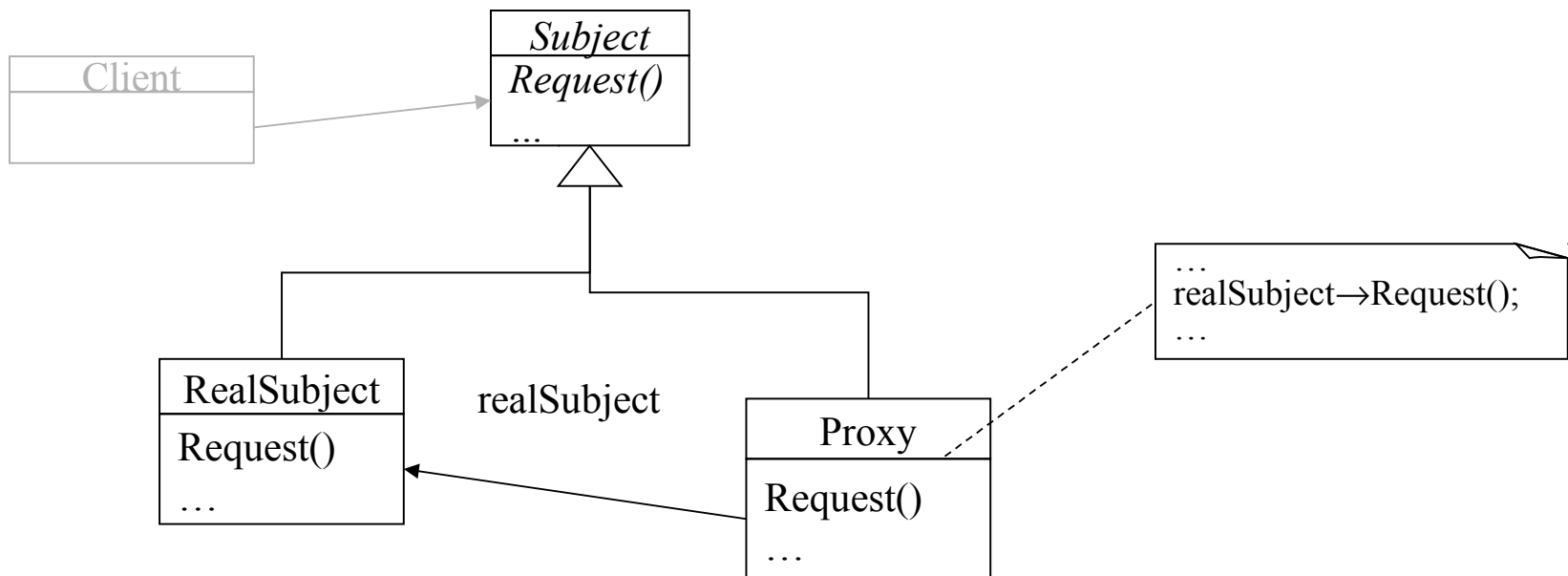
# Proxy Classes

- The RoutingSubsystem classes need to access Trip and Segment information, e.g., create a Trip providing Destinations (replan) and retrieving corresponding Segments
- That information is in the PlanningSubsystem but this must be transparent to RoutingSubsystem classes - they use *Proxy classes* for Trip and Segment
- Proxy classes use the *Proxy design pattern*

# Proxy Design Pattern

- **Intent:** Provide a surrogate or placeholder for another object to control access to it.
- **Applicability:**
  - *Remote proxies* are responsible for encoding a request and its arguments and for sending the encoded request to the real subject in a different address space.
  - *Virtual proxies* may cache additional information about the real subject so that they can postpone accessing it. For example, the ImageProxy from the Document editor (next) caches the real image's extent.
  - *Protection proxies* check that the caller has the access permissions required to perform a request.
  - *Smart reference*: additional services such as reference counting, loading from persistent storage

# Proxy Structure





## Consequences

- A remote proxy can hide the fact that an object resides in a different address space or a remote machine (myTrip example). Loading it over the network might be slow at peak load periods
- A virtual proxy (our editor example) can perform optimizations such as creating an object on demand, e.g., large images on a web browser or word processor
- Both protection proxies and smart references allow additional housekeeping tasks when an object is accessed

# SYSC-3020 — Introduction to Software Engineering

- System design concepts
  - Definitions: Architecture (subsystem decomposition), coupling, cohesion
  - Layers and partitions
  - Software architecture (MVC, Observer pattern)
  - Process architecture (UML notation and Distribution patterns)
- System design process
  - Identify design goals
  - Initial subsystem decomposition
  - Map subsystems to components and processors
  - **Persistent storage**
  - Define access control policies
  - Select control flow mechanism
  - Identify boundary conditions

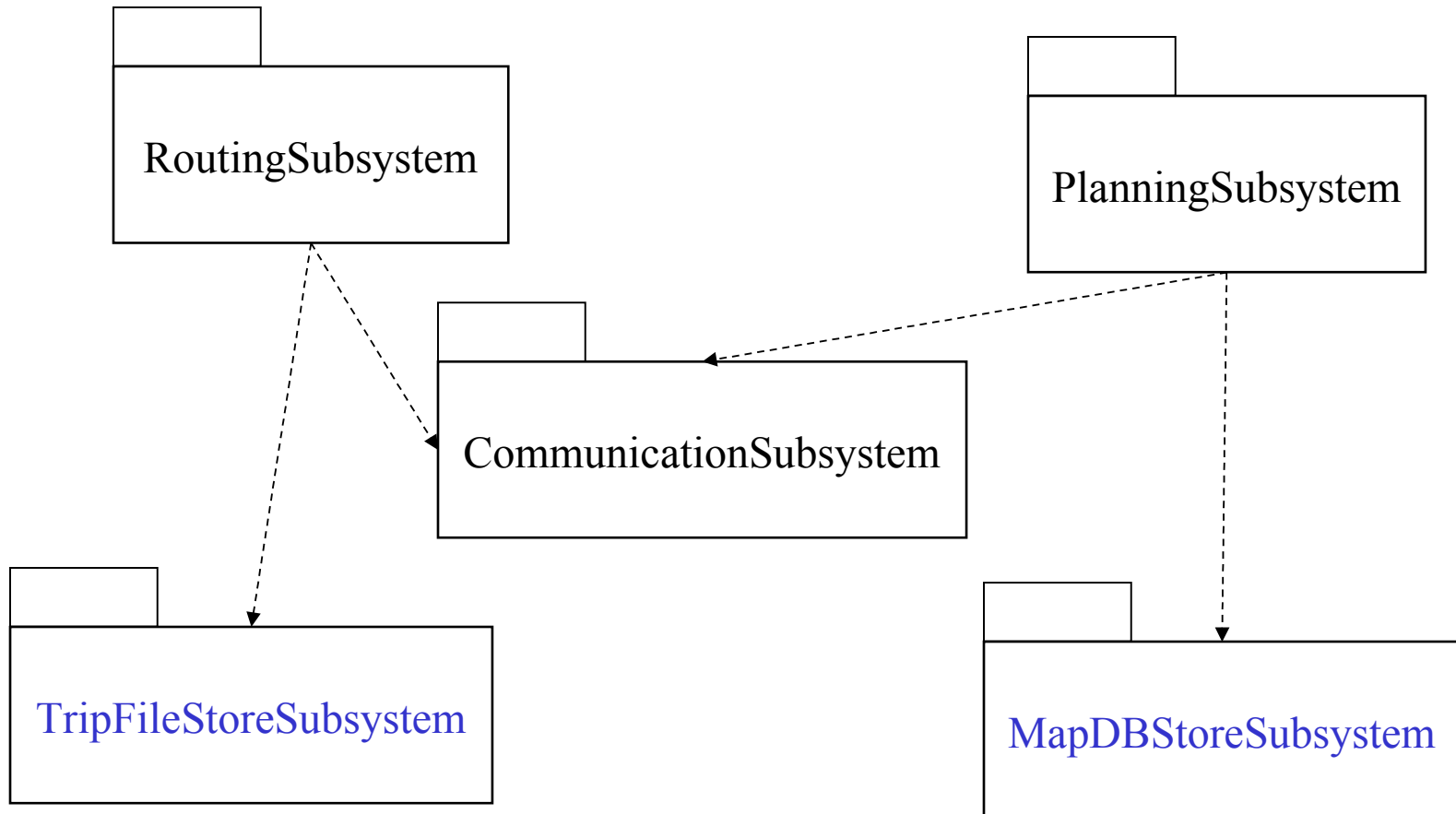
# Data Management

- Data management often an important component of many systems
  - Systems usually include one or several databases
  - How to handle data storage has significant impact on system structure, performance, etc.
  - Data storage and retrieval may represent a bottleneck
- Which data needs to be persistent?
- How should it be stored, accessed?
- Additional subsystem for data management ?

# Guidelines for Persistent Objects

- Need to identify which objects need to be persistent
- Entity objects are obvious candidates but not all need to be persistent
  - e.g., in MyTrip, `Trips` and their related classes need to be stored. But `Location` and `Direction` are constantly recomputed as the car moves.
- Other examples of persistent information:
  - information related to system users (e.g., Drivers)
  - attributes of some boundary objects (eg. user interface preferences).
- Question: which classes need to survive system shutdown or crash?

# MyTrip Data Storage



# SYSC-3020 — Introduction to Software Engineering

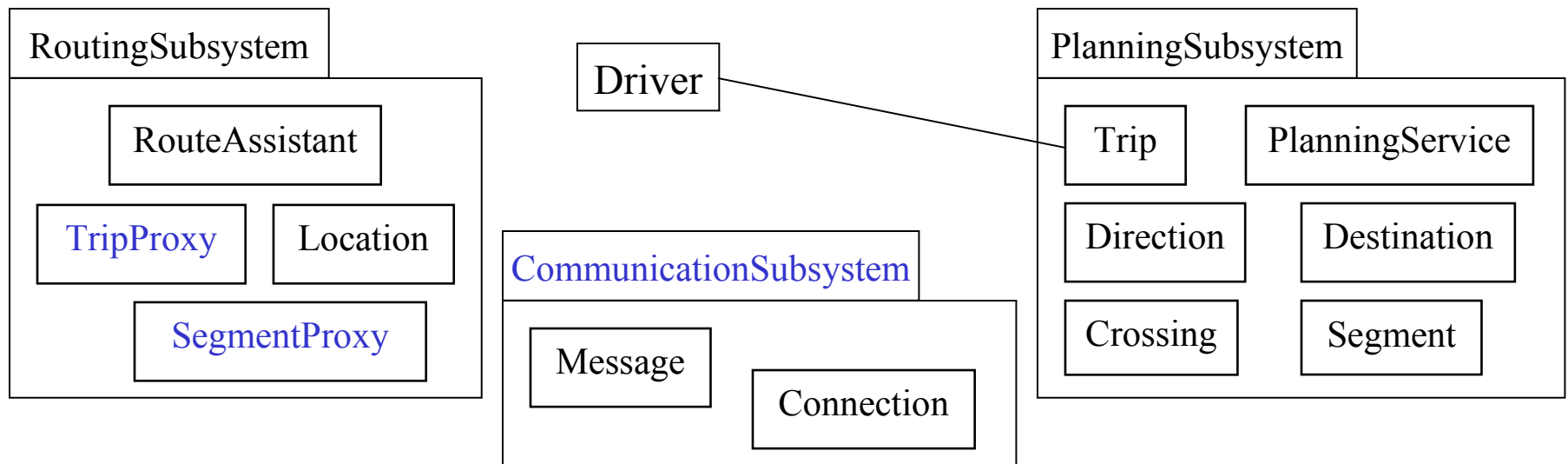
- System design concepts
  - Definitions: Architecture (subsystem decomposition), coupling, cohesion
  - Layers and partitions
  - Software architecture (MVC, Observer pattern)
  - Process architecture (UML notation and Distribution patterns)
- System design process
  - Identify design goals
  - Initial subsystem decomposition
  - Map subsystems to components and processors
  - Persistent storage
  - **Define access control policies**
  - Select control flow mechanism
  - Identify boundary conditions

# Defining Access Control

- Security requirements:
  - Which objects are shared among actors
  - How can actors control access
  - How actors are authenticated to the system
  - How selected data in the system should be encrypted

# MyTrip (I)

- Storing maps and trips in the same database for many drivers introduces security issues
- Recall security design goal: Other **drivers** should not access trips for a **driver**
  - Suggests additional `Driver` class, associate it with the `Trip` class
  - Leads to additional responsibilities for subsystems





## MyTrip (II)

- A `Driver` represents an authenticated user. It is used by the `CommunicationSubsystem` to remember keys associated with a user and by the `PlanningSubsystem` to associate `Trips` with users.
- The `CommunicationSubsystem` uses the `Driver` associated with the `Trip` being transported for selecting a key and encrypting the communication traffic
- Prior to processing any requests, the `PlanningSubsystem` authenticates the `Driver` from the `RoutingSubsystem`. The authenticated `Driver` is used to determine which `Trips` can be sent to the corresponding `RoutingSubsystem`.

# General Mechanisms

- For multiuser system, access control is usually more complex than in `MyTrip`
- For each actor, we need to define which operations they can access on which shared object
  - Static versus dynamic access control
  - Authentication: Verifying association of user/subsystem to system
  - Encryption: Preventing unauthorized access
- Access matrix is used to model access on classes:  $\text{Actors} \times \text{Classes}$ 
  - Each cell list operations that can be executed for the {actor,class} pair
- Alternative representations for access matrix :
  - Global access table: (actor, class, operation) tuple
  - Access control list: list of (actor, operation) for each class
  - Capability: list of (class operation) for each actor

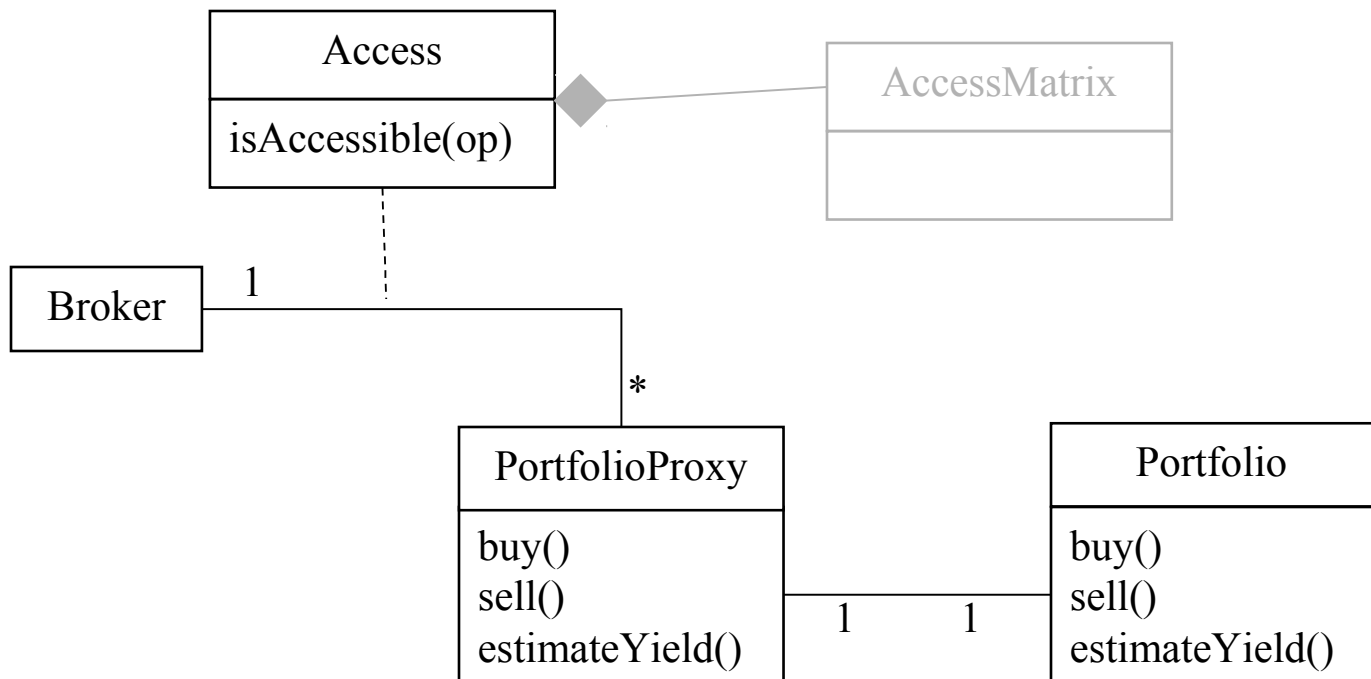
# Example

- Bank information system
- A `Teller` may debit or credit an account (operations on class `Account`) up to a predefined amount
- If this amount is exceeded, a `Manager` needs to approve the transaction
- `Managers` and `Tellers` can only access accounts in their own branch
- `Analysts` can access information across branches but cannot post transactions on individual accounts

# Example Access Matrix

		Objects		
		Corporation	LocalBranch	Account
Actors	Teller		lookupLocalAccount()	postSmallDebit() postSmallCredit() ...
	Manager		lookupLocalAccount()	... postLargeDebit() postLargeCredit() examineHistory()
	Analyst	examineGlobalDebits() examineGlobalCredits()	examineLocalDebit() examineLocalCredits()	

# Dynamic Access with Proxy Pattern Adaptation



# SYSC-3020 — Introduction to Software Engineering

- System design concepts
  - Definitions: Architecture (subsystem decomposition), coupling, cohesion
  - Layers and partitions
  - Software architecture (MVC, Observer pattern)
  - Process architecture (UML notation and Distribution patterns)
- System design process
  - Identify design goals
  - Initial subsystem decomposition
  - Map subsystems to components and processors
  - Persistent storage
  - Define access control policies
  - **Select control flow mechanism**
  - Identify boundary conditions

# Select control flow mechanism

- Control flow = sequencing of actions (operations in an OO system) in a system
- Design issue rather than an Analysis issue
  - During Analysis, we simply assume that all objects are running simultaneously, executing operations any time they need to execute them
  - During Design, we need to take into account that not every object has the luxury of running on its own processor
- Three possible control flow mechanisms:
  - Procedure-driven control
  - Event-driven control
  - Threads

# Select control flow mechanism

## Procedure-driven control

- Operations wait for input whenever they need data from an actor
- Mostly in legacy systems and systems written in procedural languages

## Event-driven control

- A main loop waits for an external event
- Whenever an event becomes available, it is dispatched to the appropriate object
- Simpler structures, inputs centralized in the loop

## Threads

- The system can create an arbitrary number of threads, each responding to a different event
- If a thread needs additional data, it waits for inputs



# Boundary Conditions

- Most system design effort is concerned with steady-state behavior.
- However, system design phase must also address boundaries of system
  - **Initialization**
    - Describes how the system is brought from a non initialized state to steady-state ("startup use cases").
  - **Termination**
    - Describes what resources are cleaned up and which systems are notified upon termination ("termination use cases").
  - **Failure (Exception Handling)**
    - An exception is an unexpected event or error that occurs during the execution of the system
    - Sources: User error, external problems (network or hardware failure – power supply), software bug
    - Good system design foresees fatal failures ("failure use cases").
    - Exception handling: catch exceptions, treat them to minimize damage

# Boundary Condition Questions

- **Initialization**

- How does the system start up?
  - What data need to be accessed at startup time?
  - What services have to registered?
- What does the user interface do at start up time?
  - How does it present itself to the user?

- **Termination**

- Are single subsystems allowed to terminate?
- Are other subsystems notified if a single subsystem terminates?
- How are local updates communicated to the database?

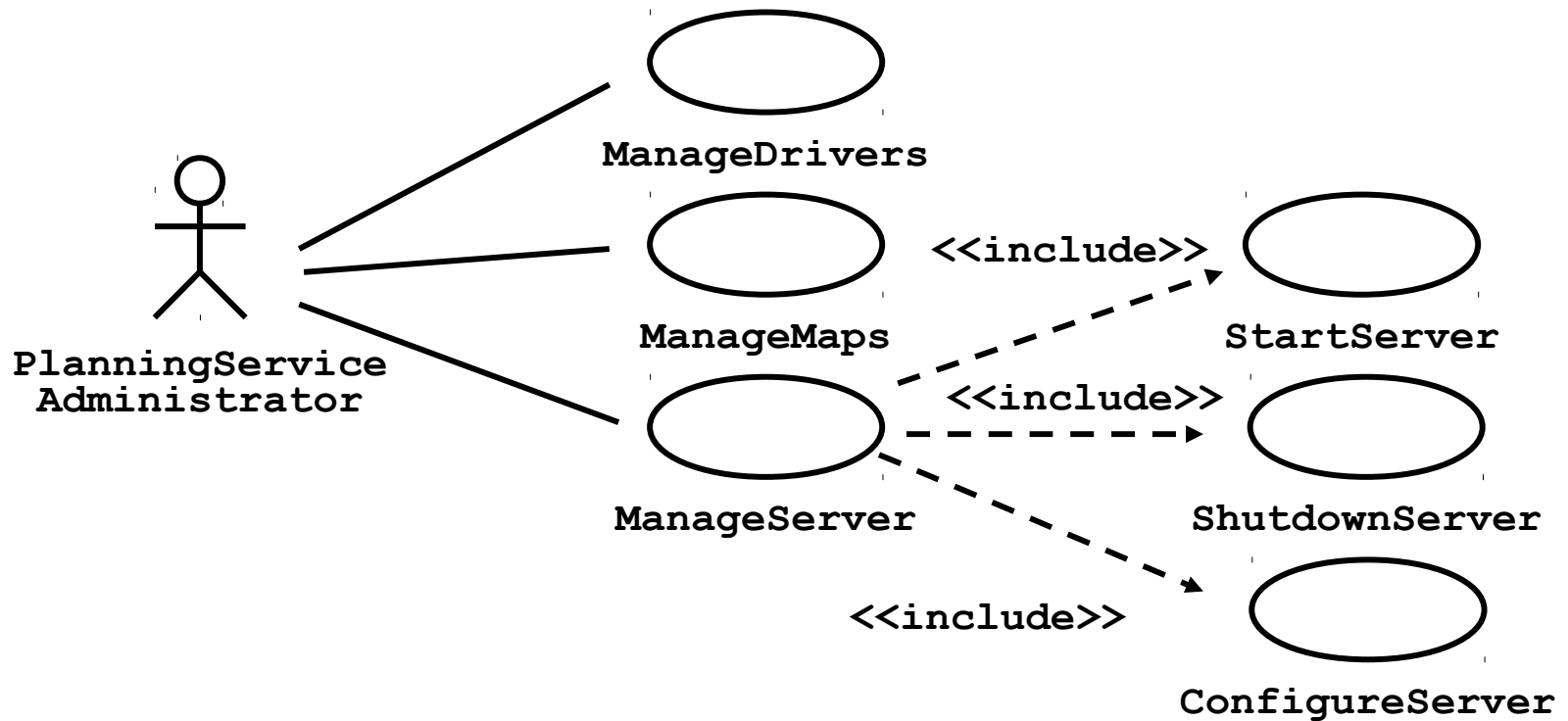
- **Failure**

- How does the system behave when a node or communication link fails? Are there backup communication links?
- How does the system recover from failure? Is this different from initialization?

# My Trip Example Questions

- Initialization and Configuration:
  - How are maps loaded into the PlanningService?
  - How is MyTrip installed in the car?
  - How does MyTrip knows which PlanningService to connect to? (configuration)
  - How are drivers added to the PlanningService?
- Termination
  - What if driver stops half-way through trip ?
- Exceptions:
  - Nonfunctional requirement: FRIEND tolerates connection failures
- These lead to new administration use cases
  - Usually treated separately

# MyTrip Administration



# Documenting System Design

1. Introduction
  - 1.1 Purpose of the system
  - 1.2 Design Goals
  - 1.3 Definitions, acronyms, abbreviations
  - 1.4 References
  - 1.5 Overview
2. Current software architecture
3. Proposed software architecture
  - 3.1 Overview
  - 3.2 Hardware/software mappings
  - 3.3 Persistent data management
  - 3.4 access control and security
  - 3.5 Global software control
  - 3.6 Boundary conditions
4. Subsystem services