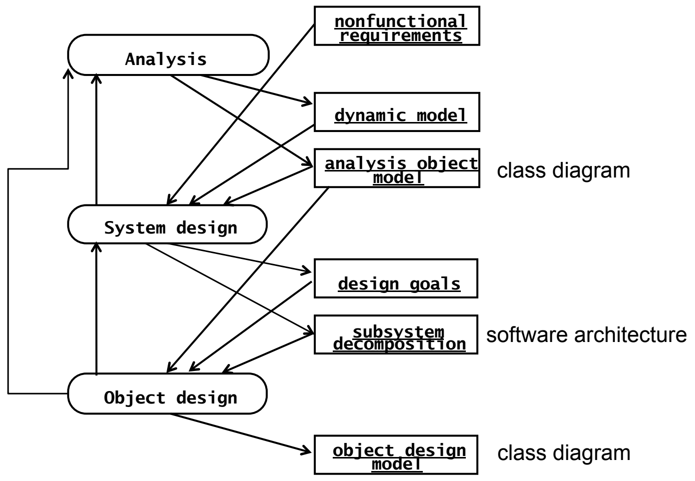
Chapter 7:
System design: Addressing design goals

CS 4354
Fall 2012

Jill Seaman

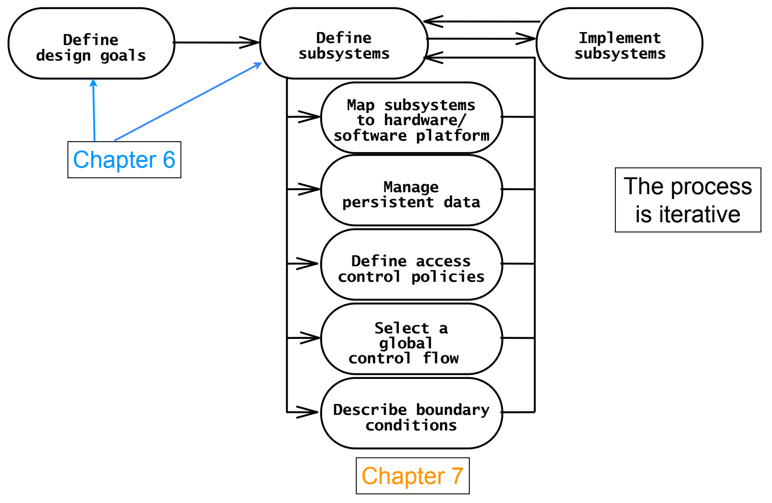
Activities and products of System design



Main activities of System Design

- **Identify design goals**: Developers identify and prioritize the qualities of the system that they should optimize
  - ✦Generally based on nonfunctional requirements
  - ✦Chapter 6
- **Design the initial subsystem decomposition**: Developers decompose the system into smaller parts.
  - ✦Based on use case and analysis models
  - ✦Chapter 6
- **Refine the subsystem decomposition to address design goals**: Initial decomposition usually does not satisfy all design goals, so developers refine it until as many as possible are satisfied.

Main activities of System Design


Define design goals
Define subsystems
Implement subsystems
Map subsystems to hardware/ software platform
Manage persistent data
Define access control policies
Select a global control flow
Describe boundary conditions
Chapter 6
Chapter 7
The process is iterative

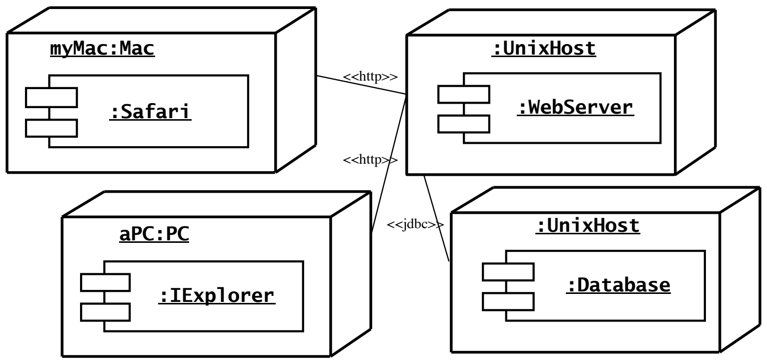
7.3 System Design Concepts UML Deployment Diagrams

- UML **deployment diagrams** are used to depict the relationship among run-time components and nodes.
- **Components** are self-contained entities that provide services to other components or actors.
  - ✦For example, a Web server is a component that provides services to Web browsers.
- A **node** is a physical device or an execution environment in which components are executed.
- A **system** is composed of interacting run-time components that can be distributed among several nodes.

5

Example UML deployment diagram

Shows the allocation of components to different nodes. Web browsers on PCs and Macs can access a Web server that provides information from a database



6

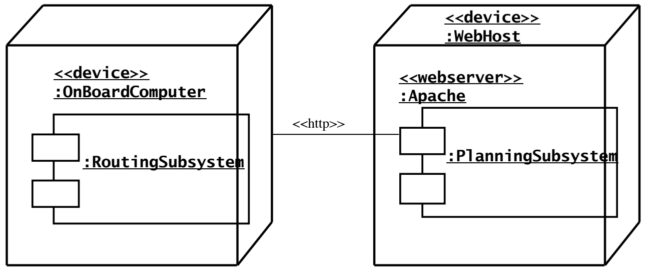
7.4.1 Mapping Subsystems to Processors and Components

- Use of multiple computers can address high-performance needs and interconnect multiple distributed users
- Often requires an infrastructure for supporting communication between subsystems
- This mapping has significant impact on performance and complexity of the system so it is done early.

7

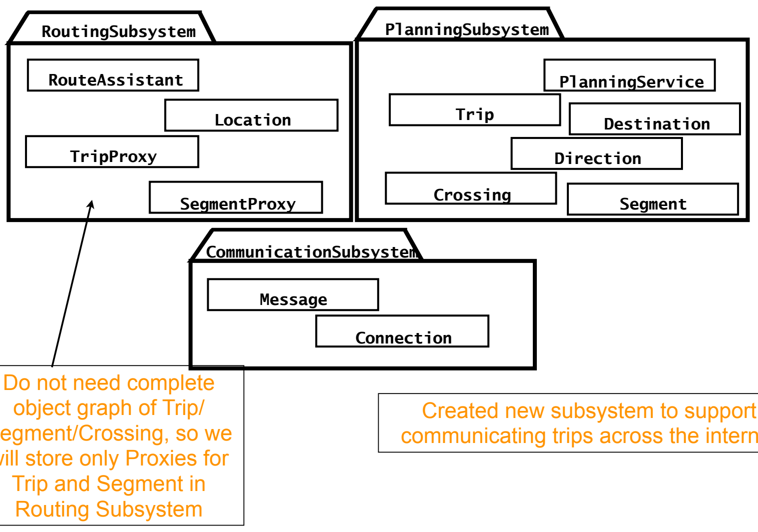
UML deployment diagram for MyTrip

- Deduce from the requirements that PlanningSubsystem is a Web-based service run on an internet host
- RoutingSubsystem runs on the onboard computer (in the car)



8

Revised design model for MyTrip

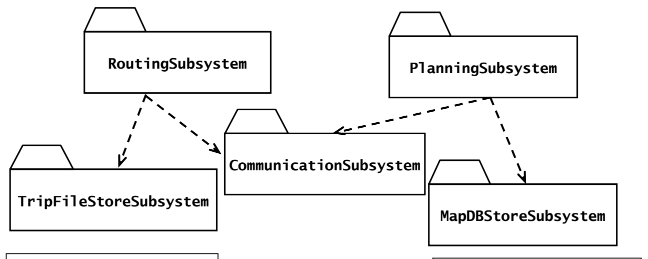


Do not need complete object graph of Trip/Segment/Crossing, so we will store only Proxies for Trip and Segment in Routing Subsystem

Created new subsystem to support communicating trips across the internet

7.4.2 Identifying and storing persistent data

- **Persistent data** outlive a single execution of the system.
- Data can be stored in a file or a database.
- How it is stored can affect system decomposition (i.e. repository)



Stores trips in a file on a small hard drive

Stores maps and trips in alarge database

Identifying persistent objects

- Usually the entity objects are persisted
- Determine: what must survive through shutdown?

- In MyTrip
  - ✦Trips, crossing, destination, PlanningService, Segment, drivers, window positions, user interface preferences, and state of long-running control objects are persistent and must be stored.
  - ✦Location and direction are not persistent. They are constantly recomputed as the car moves.

Selecting a storage management strategy

There are currently three options for storage management:

- **Flat files.** Cheap, simple, permanent storage, low level (Read, Write)
- **Relational database.** Powerful, easy to port, supports multiple writers and readers.
- **Object-oriented database.** Similar to relational database, but it stores data as objects and associations.

7.4.3 Providing Access Control

- In a multi-user system, different actors have access to different functionality and data.
- Access control: Need to determine which objects are shared among actors, and define how actors can access them.
- Access matrix:
  - ✦rows represent actors
  - ✦columns represent classes whose access is controlled
  - ✦cells contain access rights: list of operations that can be executed on instances of the class by the actor.

Access matrix for a banking system

Objects Actors	Corporation	LocalBranch	Account

Teller			postSmallDebit()
			postSmallCredit()
			examineBalance()

Manager		examineBranchStats()	postSmallDebit()
			postSmallCredit()
			postLargeDebit()
			postLargeCredit()
			examineBalance()
			examineHistory()

Analyst	examineGlobalStats()	examineBranchStats()	

Authentication and Encryption

- In order to enforce access control, the identity behind the actor must be verified
- The process of verifying the association between the identity of the user or subsystem and the system is called **authentication.**
  - ✦user names and passwords
  - ✦access cards, etc.
- **Encryption**: used to prevent unauthorized access to text data:
  - ✦uses a key to encode each character into some other character
  - ✦used to encode stored passwords, messages passed across a network.
- Authentication and encryption methods should be determined during System Design.

7.4.4 Designing Global Control Flow

- Control flow is the sequencing of actions in a system.
- Procedure driven control:
  - ✦Operations wait for input from users
  - ✦Procedures call other procedures and wait for results
- Event driven control:
  - ✦Main loop waits for external event, event is dispatched to appropriate object to handle it.
  - ✦Commonly used in GUI systems
- Threads:
  - ✦Concurrent operation of procedure driven control
  - ✦Hard to debug

7.4.6 Identifying Boundary Conditions

- We need to decide how the system is started, initialized, and shut down
- We need to define how we deal with major failures such as data corruption and network outages
- Use cases dealing with these conditions are called boundary use cases.

Identifying Boundary Conditions

- In general, we identify boundary use cases by examining each subsystem and each persistent object:
- **Configuration.** For each persistent object, we examine in which use case it is created or destroyed or archived. For objects that are not created or destroyed in any of the common use cases (e.g. Maps in the MyTrip system), we add a use case invoked by a system administrator.
- **Start-up and shutdown.** For each component (e.g., a WebServer), we add three use cases to start, shutdown, and configure the component.
- **Exception handling.** For each type of component failure (e.g., network outage), we decide how the system should react (e.g., inform users of the failure).

7.4.7 Reviewing System Design

- We need to ensure that the system design model is **correct**, **complete**, **consistent**, **realistic**, and **readable**.
- The system design model is **correct** if the analysis model can be mapped to the system design model.
  - ✦Can every subsystem be traced back to a use case or a nonfunctional requirement?
  - ✦Can every use case be mapped to a set of subsystems?
  - ✦Can every design goal be traced back to a nonfunctional requirement?
  - ✦Is every nonfunctional requirement addressed in the system design model?
  - ✦Does each actor have an access policy?
  - ✦Is every access policy consistent with the nonfunctional security requirement?

7.4.7 Reviewing System Design

- The model is **complete** if every requirement and every system design issue has been addressed.
  - ✦Have the boundary conditions been handled?
  - ✦Was there a walkthrough of the use cases to identify missing functionality in the system design?
  - ✦Have all use cases been examined and assigned a control object?
  - ✦Have all aspects of system design (i.e., hardware allocation, persistent storage, access control, legacy code, boundary conditions) been addressed?
  - ✦Do all subsystems have definitions?

7.4.7 Reviewing System Design

- The model is **consistent** if it does not contain any contradictions.
  - ✦Are conflicting design goals prioritized?
  - ✦Does any design goal violate a nonfunctional requirement?
- The model is **realistic** if the system can be implemented.
  - ✦Are any new technologies or components included in the system?
  - ✦Have performance and reliability requirements been reviewed in the context of subsystem decomposition?
- The model is **readable** if developers not involved in the system design can understand the model.
  - ✦Are subsystem names understandable?
  - ✦Do entities (e.g., subsystems, classes) with similar names denote similar concepts?