Chapter 7,
System Design (Part 2)

Attività del System design

- Gli obiettivi (Design Goals) guidano le decisioni che gli sviluppatori devono prendere specialmente quando sono necessari dei compromessi.
- Gli sviluppatori dividono il sistema in sottosistemi per gestire la complessità in modo che lo sviluppo di ogni sottosistema possa essere assegnato ad un team e realizzato in modo indipendente
- Perché questo sia possibile devono far fronte a determinate "scelte" quando il sistema viene decomposto

  Mapping Hardware/Software

  Gestione dei Dati Persistenti

  Controllo di Accesso

  Flusso di controllo globale

  Condizioni limite

Attività del System design (cont.)

- Mapping Hardware/Software
  - **Qual'è la configurazione hardware del sistema?**
  - **Quale nodo è responsabile di una certa funzionalità?**
  - **Come è gestita la comunicazione tra i nodi realizzati?**
  - **Quali servizi sono realizzati utilizzando componenti software esistenti? Come queste componenti sono incapsulate?**

- Molte volte tale mapping porta alla definizione di *componenti addizionali* che consentono di "muovere" le informazioni da un nodo ad un altro, di gestire problemi di concorrenza
  - **Le componenti legacy o Off-the-shelf consentono agli sviluppatori di realizzare servizi complessi in modo più economico**
  - **Esempi: user interface packages, data base management systems**
  - **Le componenti dovrebbero essere incapsulate per minimizzare le dipendenze da una particolare componente (se una azienda competitor in futuro fornirà un prodotto migliore o più economico potremmo voler cambiare senza influenzare il resto del sistema)**

Attività del System design (cont.)

- Gestione dei Dati Persistenti
  - **Quale dovrebbe essere l'informazione persistente?**
  - **Dove dovrebbe essere memorizzata?**
  - **Come accedere ai dati persistenti?**

- Potrebbe rappresentare un "collo di bottiglia"
  - **Molte funzionalità del sistema richiedono la creazione o la manipolazione di dati persistenti**
  - **Quindi l'accesso dovrebbe essere "veloce" e "affidabile"**
  - **Si devono effettuare scelte consistenti a livello di sistema**
  - **Spesso richiede la scelta di un database management system e di un *sottosistema addizionale* per la gestione dei dati persistenti**

Attività del System design (cont.)

- Controllo di Accesso
  - **Chi può accedere alle informazioni?**
  - **Il controllo di accesso può cambiare dinamicamente?**
  - **Come è specificato e realizzato, il controllo di accesso?**

- Deve essere consistente
  - **La politica utilizzata per specificare chi può e chi non può accedere a certe informazioni dovrebbe essere la stessa per tutti i sottosistemi**

Attività del System design (cont.)

- Controllo del Flusso
  - **Come è gestita la sequenza delle operazioni?**
  - **Il sistema è guidato da eventi?**
  - **Il sistema può gestire più di un'interazione utente alla volta?**

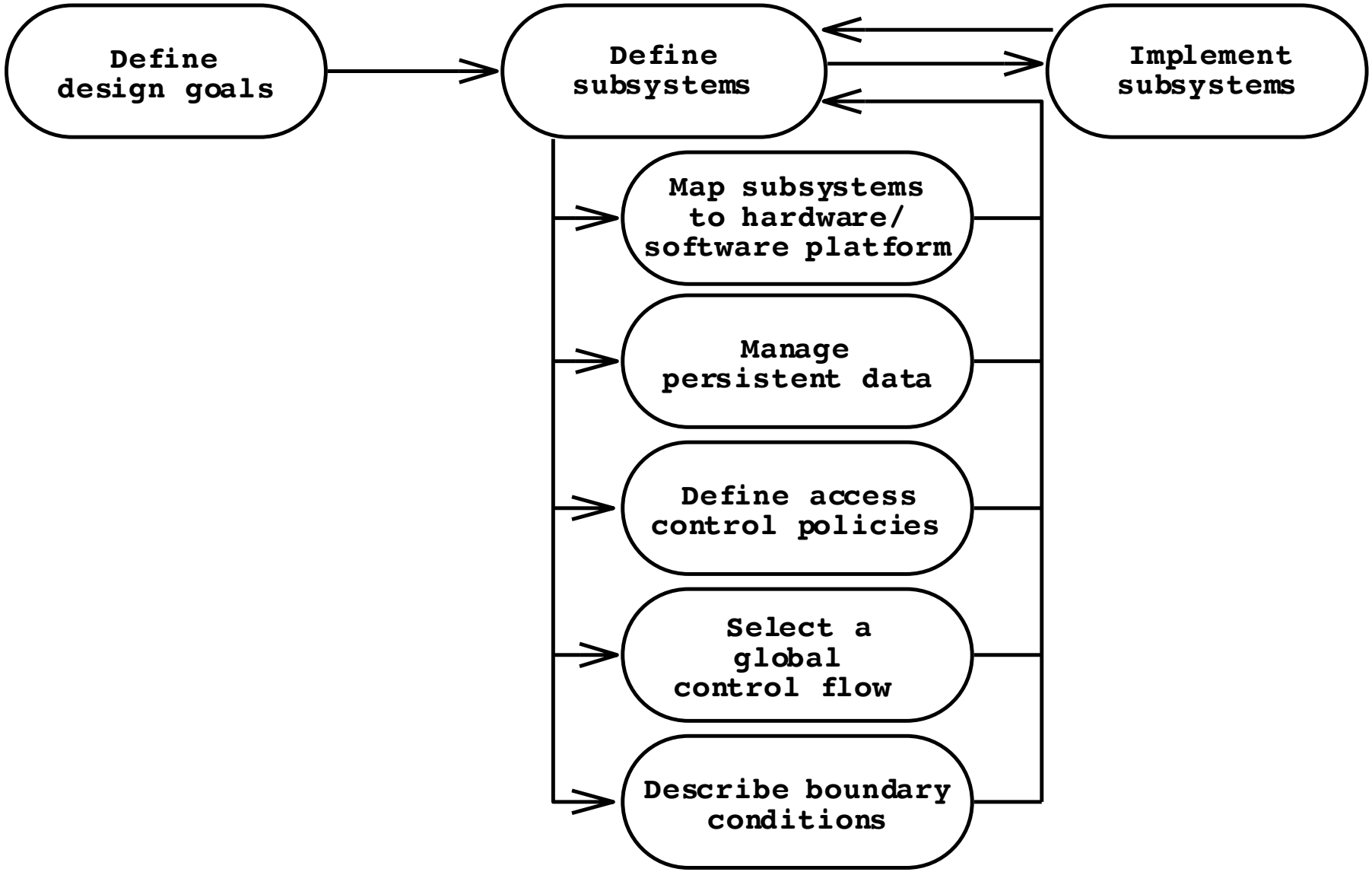
- La scelta del controllo del flusso ricade sulle interfacce delle componenti
  - **nel caso di un controllo guidato da eventi, i sottosistemi forniranno un gestore degli eventi.**
  - **nel caso di un controllo basato su threads, i sottosistemi devono garantire la mutua esclusione nelle sezioni critiche**

Attività del System design (cont.)

- Condizioni limite
  - **Come è avviato il sistema?**
  - **Come è interrotto?**
  - **Come sono individuati e gestiti i casi eccezionali?**

- L'avvio e l'interruzione del sistema spesso rappresentano molta della complessità di un sistema, specialmente nei sistemi distribuiti
  - **Incidono sulle interfacce di tutti i sottosistemi**

Attività del system design (cont.)



Documenting system design

- System design is documented in the System Design Document (SDD). It describes:
  - **Design goals set by the project**
  - **Subsystem decomposition (with UML class diagrams)**
  - **Hardware/ software mapping (with UML deployment diagrams)**
  - **Data management**
  - **Access control**
  - **Control flow mechanisms**
  - **Boundary conditions**
- The SDD is used to define interfaces between teams of developers and serve as a reference when architecture-level decisions need to be revisited.
- Audience for the SDD includes:
  - **the project management**
  - **the system architects (i.e., the developers who participate in the system design)**
  - **the developers who design and implement each subsystem**

Gestione del system design

- Come nel caso della fase di analisi, la sfida primaria nel gestire la fase di system design è quella di mantenere la consistenza mentre si stanno utilizzando più risorse possibili

- Documento del System Design (SDD)

- Assegnare le responsabilità

- Iterare le attività

Documento del System Design (SDD)

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

   Glossary

System Design Document (SDD) Structure (1)

1. Introduction → *brief overview of the software architecture and the design goals*
   - 1.1 Purpose of the system
   - 1.2 Design goa1s
   - 1.3 Definitions, acronyms, and abbreviations
   - 1.4 References → *References to other documents and traceability information (e.g., related requirements analysis document, references to existing systems, constraints impacting the software architecture).*
   - 1.5 Overview
2. Current software architecture
3. Proposed software architecture
   - 3.1 Overview
   - 3.2 Subsystem decomposition
   - 3.3 Hardware/software mapping
   - 3.4 Persistent data management
   - 3.5 Access control and security
   - 3.6 Global software control
   - 3.7 Boundary conditions
4. Subsystem services

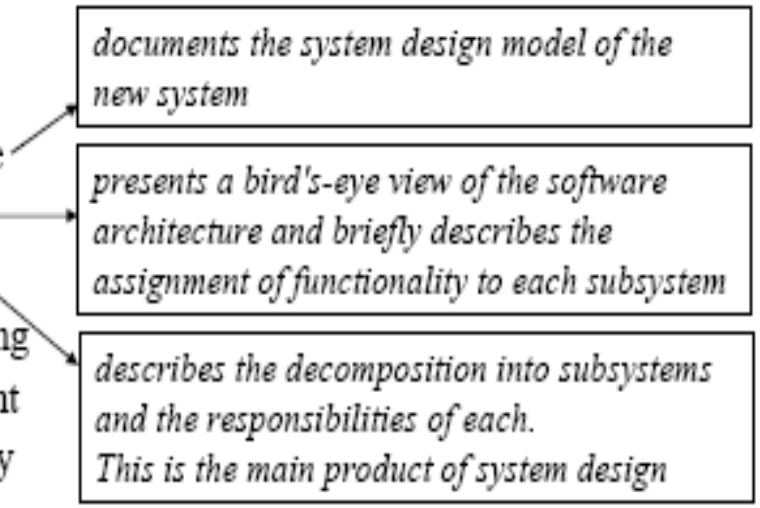
System Design Document (SDD) Structure (2)

1. Introduction
   - 1.1 Purpose of the system
   - 1.2 Design goa1s
   - 1.3 Definitions, acronyms, and abbreviations
   - 1.4 References
   - 1.5 Overview
2. Current software architecture
3. Proposed software architecture
   - 3.1 Overview
   - 3.2 Subsystem decomposition
   - 3.3 Hardware/software mapping
   - 3.4 Persistent data management
   - 3.5 Access control and security
   - 3.6 Global software control
   - 3.7 Boundary conditions
4. Subsystem services

the architecture of the system being replaced. If there is no previous system, this section can be replaced by a survey of current architectures for similar systems
The purpose of this section is to make explicit the background information that system architects used, their assumptions, and common issues the new system will address.

System Design Document (SDD) Structure (3)

1. Introduction
   - 1.1 Purpose of the system
   - 1.2 Design goa1s
   - 1.3 Definitions, acronyms, and abbreviations
   - 1.4 References
   - 1.5 Overview
2. Current software architecture
3. Proposed software architecture
   - 3.1 Overview
   - 3.2 Subsystem decomposition
   - 3.3 Hardware/software mapping
   - 3.4 Persistent data management
   - 3.5 Access control and security
   - 3.6 Global software control
   - 3.7 Boundary conditions
4. Subsystem services



System Design Document (SDD) Structure (4)

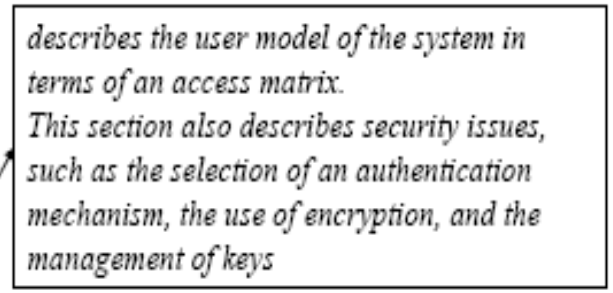
1. Introduction
   - 1.1 Purpose of the system
   - 1.2 Design goa1s
   - 1.3 Definitions, acronyms, and abbreviations
   - 1.4 References
   - 1.5 Overview
2. Current software architecture
3. Proposed software architecture
   - 3.1 Overview
   - 3.2 Subsystem decomposition
   - 3.3 Hardware/software mapping
   - 3.4 Persistent data management
   - 3.5 Access control and security
   - 3.6 Global software control
   - 3.7 Boundary conditions
4. Subsystem services

describes how subsystems are assigned to hardware and off-the-shelf components. It also lists the issues introduced by multiple nodes and software reuse

describes the persistent data stored by the system and the data management infrastructure required for it. This section typically includes the description of data schemes, the selection of a database, and the description of the encapsulation of the database

System Design Document (SDD) Structure (5)

1. Introduction
   - 1.1 Purpose of the system
   - 1.2 Design goa1s
   - 1.3 Definitions, acronyms, and abbreviations
   - 1.4 References
   - 1.5 Overview
2. Current software architecture
3. Proposed software architecture
   - 3.1 Overview
   - 3.2 Subsystem decomposition
   - 3.3 Hardware/software mapping
   - 3.4 Persistent data management
   - 3.5 Access control and security
   - 3.6 Global software control
   - 3.7 Boundary conditions
4. Subsystem services



System Design Document (SDD) Structure (6)

1. Introduction
   - 1.1 Purpose of the system
   - 1.2 Design goa1s
   - 1.3 Definitions, acronyms, and abbreviations
   - 1.4 References
   - 1.5 Overview
2. Current software architecture
3. Proposed software architecture
   - 3.1 Overview
   - 3.2 Subsystem decomposition
   - 3.3 Hardware/software mapping
   - 3.4 Persistent data management
   - 3.5 Access control and security
   - 3.6 Global software control
   - 3.7 Boundary conditions
4. Subsystem services

describes how the global software control is implemented.
In particular, this section should describe how requests are initiated and how subsystems synchronize.
This section should list and address synchronization and concurrency issues

System Design Document (SDD) Structure (7)

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

describes the start-up, shutdown, and error behavior of the system. (If new use cases are discovered for system administration, these should be included in the requirements analysis document, not in this section.)

System Design Document (SDD) Structure (8)

1. Introduction
   - 1.1 Purpose of the system
   - 1.2 Design goa1s
   - 1.3 Definitions, acronyms, and abbreviations
   - 1.4 References
   - 1.5 Overview
2. Current software architecture
3. Proposed software architecture
   - 3.1 Overview
   - 3.2 Subsystem decomposition
   - 3.3 Hardware/software mapping
   - 3.4 Persistent data management
   - 3.5 Access control and security
   - 3.6 Global software control
   - 3.7 Boundary conditions
4. Subsystem services

describes the services provided by each subsystem in terms of operations. Although this section is usually empty or incomplete in the first versions of the SDD, this section serves as a reference for teams for the boundaries between their subsystems. The interface of each subsystem is derived from this section and detailed in the Object Design Document

Esempio: Modellare un sistema di route planning per autisti denominato MyTrip.

- Usando MyTrip, un autista può pianificare un viaggio da un computer di casa contattando un trip planning service sul web (**vedi use case PlanTrip**).
- Il viaggio è memorizzato sul server per recuperarlo successivamente.
- Il trip planning service deve supportare più di un autista.
- L'autista poi entra in auto e inizia il viaggio, mentre il computer a bordo fornisce le direzioni basandosi sull'informazione di viaggio fornite dal planning service e l'attuale posizione indicata dal sistema GPS a bordo (**vedi use case ExecuteTrip**).

Use case: PlanTrip

- 1. Entry Condition: The driver activates her home computer and logs into the trip planning web service
- 2. Upon successful login, the driver enters constraints for a trip as a sequence of destinations.
- 3. Based on a database of maps, the planning service computes the shortest way visiting the destinations in the specified order. The result is a sequence of segments binding a series of crossings and a list of directions.
- 4. The driver can revise the trip by adding or removing destinations.
- 5. Exit Condition: The driver saves the planned trip by name in the planning service database for later retrieval.

Use case: ExecuteTrip

- 1. Entry Condition: The driver starts her car and logs into the onboard route assistant.
- 2. Upon successful login, the driver specifies the planning service and the name of the trip to be executed.
- 3. The onboard route assistant obtains the list of destinations, directions, segments, and crossings from the planning service.
- 4. Given the current position, the route assistant provides the driver with the next set of directions.
- 5. Exit Condition: The driver arrives to destination and shuts down the route assistant.

Non Functional requirements

- 1. MyTrip is in contact with the trip planning service via a wireless modem. It can be assumed that the wireless modem functions properly at the initial destination.
- 2. Once the trip has been started, MyTrip should give correct directions even if modem fails to mantain a connection with the trip planning service.
- 3. MyTrip should minimize connection time to reduce operation costs.
- 4. Replanning is possible only if the connection to the trip planning service is possible.
- 5. The trip planning service can support at least 50 different drivers and 1000 trips.

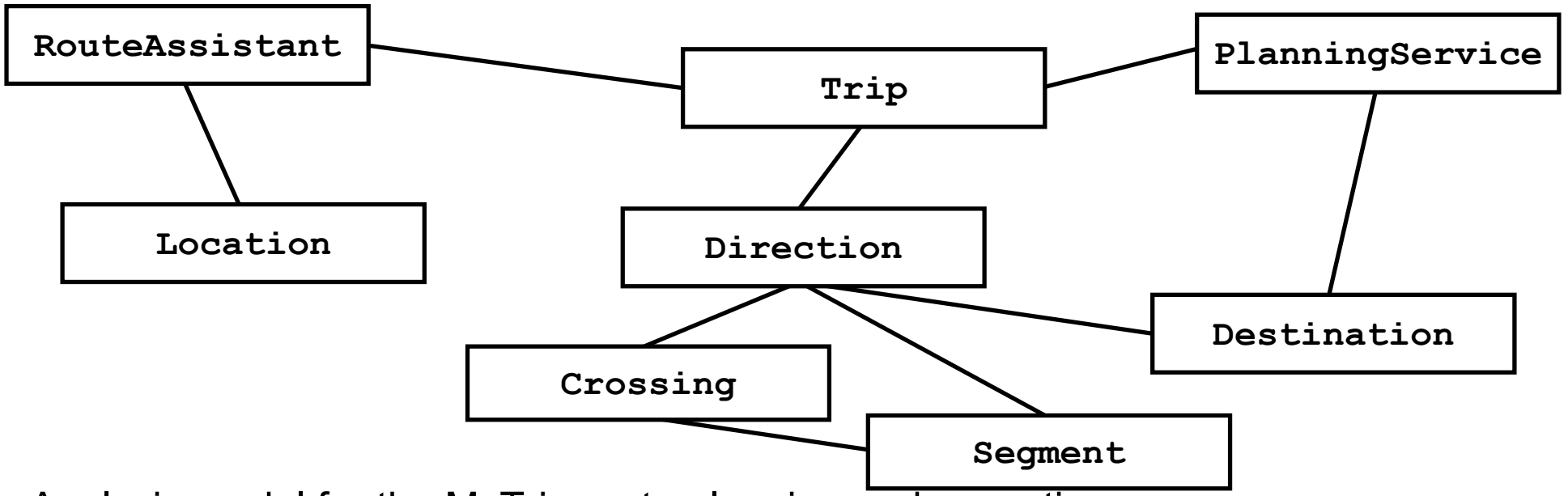
Design Goals

1. **Reliability**: MyTrip dovrebbe essere affidabile
   [generalizzazione del requisito non funzionale 2]

2. **Fault Tolerance**: MyTrip dovrebbe essere tollerante ai fallimenti dovuti alla perdita di connettività con il sistema di routing [requisito non funzionale 2 rifrasato]

3. **Security**: MyTrip dovrebbe essere sicuro, ad esempio non dovrebbe permettere ad utenti non autorizzati di accedere ai percorsi memorizzati dal driver [dedotto dal dominio di applicazione]

4. **Modificability**: MyTrip dovrebbe essere modificabile per utilizzare diversi servizi di routing [anticipazione di possibili cambiamenti da parte degli sviluppatori].



- Analysis model for the MyTrip route planning and execution.

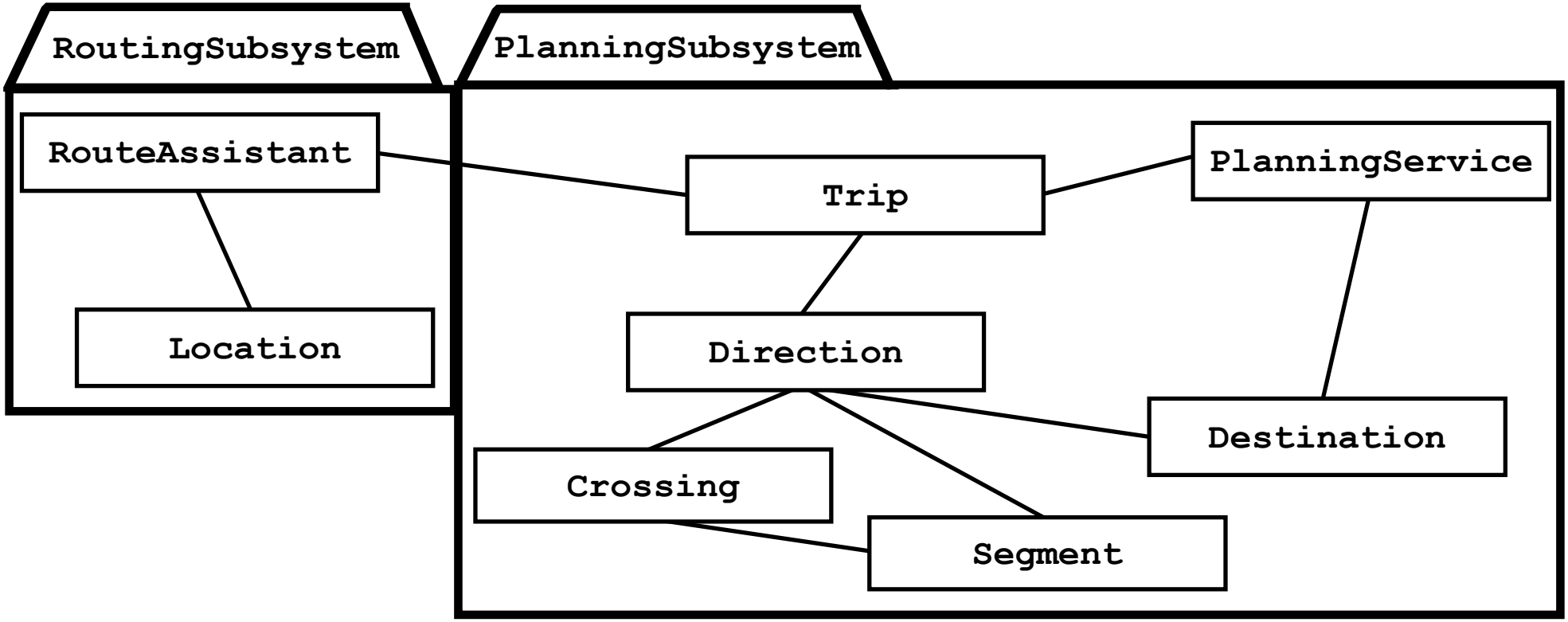
Crossing is a geographical point where a driver can choose between several Segments
Destination represents a location where the driver whishes to go
Direction, given a Crossing and adjacent Segment, a Direction describes in natural language terms how to steer the car onto the given Segment
Location is the position of the car as known by the onboard GPS system or the number of turns of the wheels.
PlanningService is a Web server that can supply a trip, linking a number of destinations in the form of a sequence of crossings and segments.
RouteAssistant gives Directions to the driver, given the current Locations and upcoming Crossing
Segment represents the road between two Crossings
Trip is a sequence of Directions between two Destinations

Identifying Subsystems

- Goal: Avere una soluzione con grosso livello di information-hiding, fa in modo che una volta che le interfacce dei sottosistemi sono definite, il loro progetto e sviluppo può procedere in modo indipendente
- Basato su euristiche e iterativo
  - **I sottosistemi sono fusi, divisi…ne sono aggiunti di nuovi**
- La decomposizione iniziale è basata sui requisiti funzionali, e.g., oggetti coinvolti nello use case (1) PlanTrip use case e (2) ExecuteTrip
  - **Classes Trip, Direction, Crossing, Segment, Destination sono un insieme strettamente accoppiato**
  - **Assegnato a PlanningSubsystem con PlanningService, per minimizzare le associazioni che attraversano i confini dei sottosistemi (can be refined based on sequence diagram)**

Euristiche

- 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



Initial subsystem decomposition for MyTrip (UML class diagram).

PlanningSubSystem ***is responsible for constructing a Trip connecting a sequence of destinations. The PlanningSubsystem is also responsible for responding to replan requests from RoutingSubsystem***

RoutingSubsystem ***is responsible for downloading a Trip from the PlanningService and executing it by giving Directions to the driver based on its Location***

Disegnare i sottosistemi con UML

- Durante il system design dobbiamo modellare la struttura statica e quella dinamica:
  - **Diagramma delle componenti per la struttura statica**
    - **Mostra la struttura al "design time", o al "compile time"**
  - **Diagramma di deployment per la struttura dinamica**
    - **Mostra la struttura al "run-time"**

UML Components

- In the context of UML, we say we distribute *components* to *nodes*.
- The definition of a component depends on the environment of the system
- May comprise files of source code but also executable, database systems, libraries, web pages, etc.
- A node is a processor available that can execute some of the system processes, e.g., on the network.
- In the simplest case, each subsystem executed as a component (or several) which are distributed.

UML Components

- If source code, labeled with source file name
- Dependencies between components indicate that a component refer to services offered by other components (<<use>> dependency)

UML Component Diagram

- Un grafo delle componenti connesse attraverso relazioni di dipendenza
- Mostra le dipendenze tra le componenti software
- Le dipendenze sono mostrate con archi dalla componente cliente alla componente fornitore
  - **I tipi di dipendenza sono specifici dei linguaggi di implementazione**

Dependencies

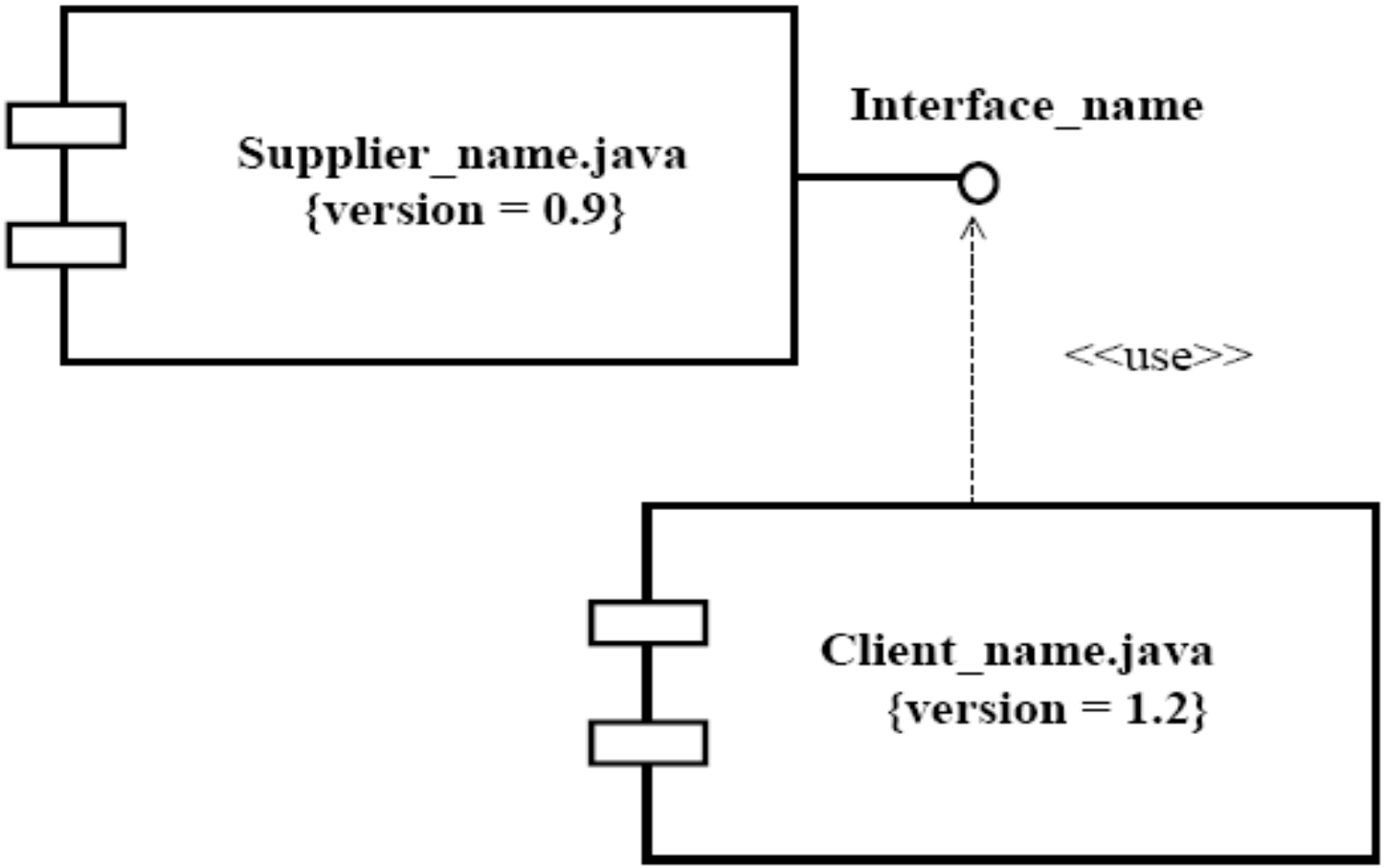
- Compilation, execution
- When a component is modified, other components that depend upon it may also have to be modified
- Dependencies are transitive
- Dependencies come from the logic design properties: generalization, aggregation, interface (parameter type),
- Cycles or loops should be avoided, e.g., affects testability

Interfaces

- A *named* set of operations
- Characterize the behavior of a component

- Essentially generalize the notion of interface in Java
- Components are said to *conform* to interfaces: support all operations in interface
- Interfaces can help specify what part of a class is actually used by client classes

UML Representations



UML Deployment Diagram

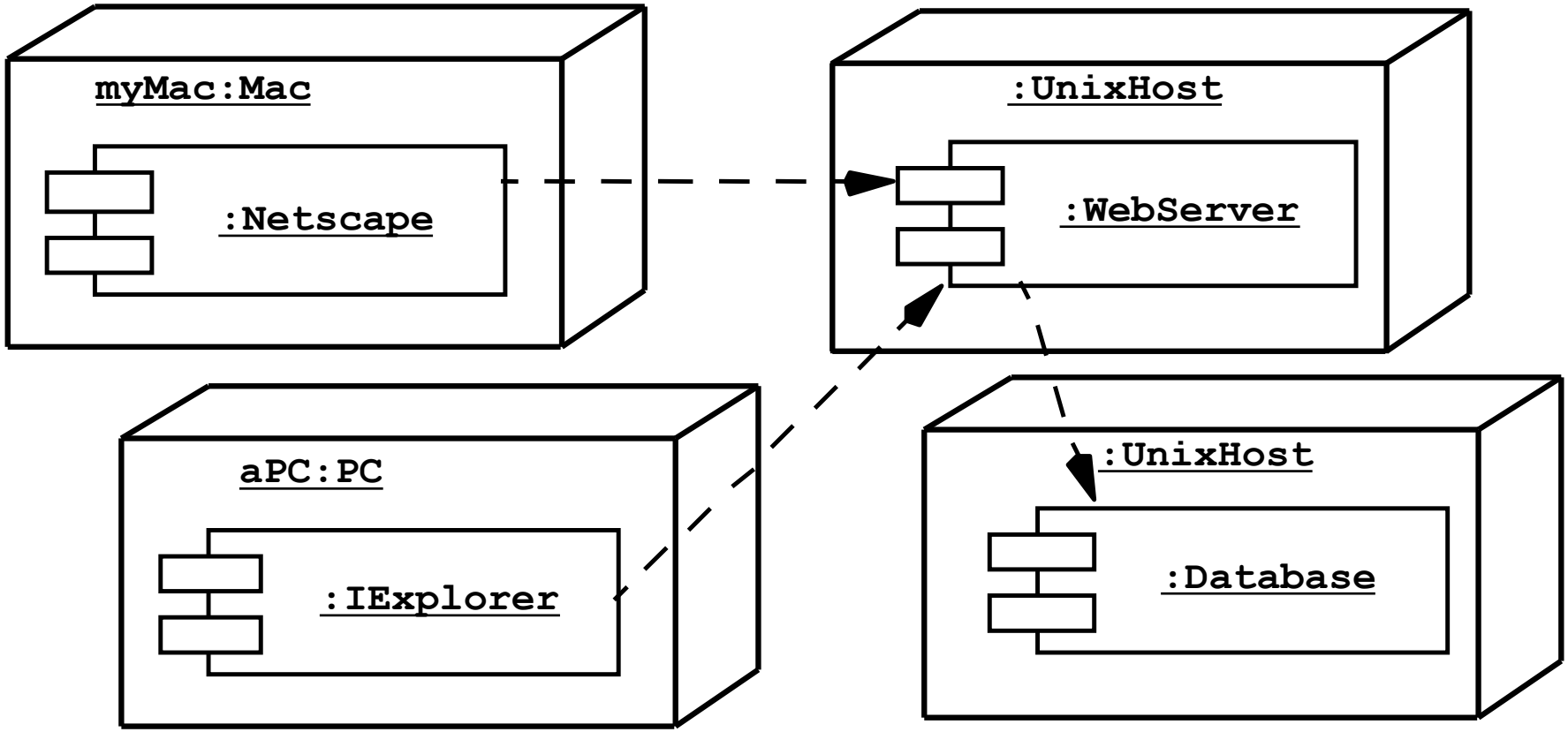
- Sono utili per mostrare il progetto del sistema dopo che le seguenti decisioni sono state prese:
  - **Decomposizione in sottosistemi**
  - **Concorrenza**
  - **Mapping Hardware/Software**

- Il deployment diagram è un grafo di nodi connessi attraverso associazioni di comunicazione.
  - **I nodi sono mostrati come box 3-D**
  - **I nodi possono contenere istanze di componenti.**
  - **Le componenti possono contenere oggetti (indica che un oggetto fa parte di una componente)**

UML Deployment Diagram (cont.)

- Sono utilizzati per descrivere le relazioni tra le componenti run-time e i nodi hardware
- Le componenti sono entità autonome (indipendenti) che forniscono servizi ad altre componenti o attori
  - **Per esempio, un Web server è una componente che fornisce servizi ai browser.**
  - **Un browser come Internet Explorer è una componente che fornisce un servizio ad un utente**
  - **Un sistema distribuito può essere composto da molte componenti a run-time che interagiscono**
  -

UML Deployment Diagram (cont.)



Un UML deployment diagram che rappresenta l'allocazione di componenti a diversi nodi e le dipendenze tra le componenti.
I browser su `PC` e `Mac` possono accedere un `WebServer` che fornisce informazioni da un `Database`.

Attività del System design

- Mappare le componenti su piattaforme e processori
- Identificare e memorizzare informazioni persistenti
- Stabilire controllo di accesso
- Progettare il flusso di controllo globale
- Identificare le condizioni limite
- Rivedere il modello del system design

Attività del System design (cont.)

- Mapping Hardware/Software
  - **Qual'è la configurazione hardware del sistema?**
  - **Quale nodo è responsabile di una certa funzionalità?**
  - **Come è gestita la comunicazione tra i nodi realizzati?**
  - **Quali servizi sono realizzati utilizzando componenti software esistenti? Come queste componenti sono incapsulate?**

- Molte volte tale mapping porta alla definizione di *componenti addizionali* che consentono di "muovere" le informazioni da un nodo ad un altro, di gestire problemi di concorrenza
  - **Le componenti legacy o Off-the-shelf consentono agli sviluppatori di realizzare servizi complessi in modo più economico**
  - **Esempi: user interface packages, data base management systems**

Selezionare una configurazione hardware e una piattaforma

- Questioni relative ai processori:
  - **La computazione richiede più processori?**
  - **Possiamo velocizzare l'elaborazione distribuendo le mansioni tra più processori?**
- Come realizzare i sottosistemi: Hardware o Software?
- Molte delle difficoltà della progettazione del sistema software sono legate alle scelte relative all'hardware e al software imposte.
- Significa selezionare una macchina virtuale:
  - **Sistema operativo e ogni altro software necessario (DBMS o package di comunicazione)**
  - **La macchina virtuale riduce il gap tra hardware e software da realizzare**
  - **Può essere vincolata dal cliente, da ciò che già possiede**
  - **Può essere vincolata da questioni di costi: trade-off tra costruire o comprare**
- Come mappare gli oggetti sull'hardware scelto e sul software?
  - **Mappare gli oggetti sui processori**
  - **Mappare le associazioni: connettività**

Connettività nei Sistemi distribuiti

- Se l'architettura è distribuita abbiamo bisogno di descrivere la rete (e quindi il sottosistema di comunicazione).

- Domande a cui rispondere
  - **Quali media sono utilizzati? (Ethernet, Wireless)**
  - **Qual'è il tasso medio di trasmissione?**
  - **Quale tipo di protocollo di comunicazione può essere usato?**
  - **Interazioni sincrone o asincrone?**
  - **Qual'è la richiesta di larghezza di banda tra i sottosistemi?**

Questioni relative al mapping Hardware/Software

- Qual'è la connessione tra le unità fisiche?
  - **Albero? matrice?**
  - **Qual'è il protocollo di comunicazione appropriato tra i sottosistemi?**
  - **Larghezza della banda richiesta? affidabilità richiesta?**
- Certe funzionalità sono già disponibili nell'hardware scelto?
- Certe mansioni richiedono specifiche locazioni per controllare l'hardware o per permettere operazioni concorrenti?
- Questioni generali relative alle performance :
  - **Qual'è il tempo di risposta desiderato?**

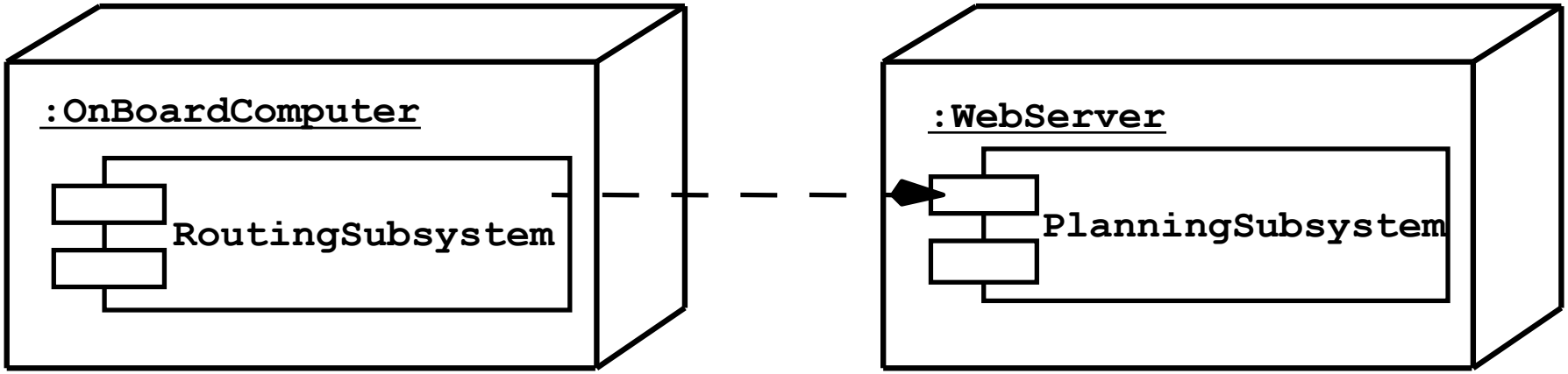
Allocare oggetti e sistemi a nodi

- Consente di distribuire le funzionalità e l'elaborazione dove è necessario
- Consente di identificare nuovi oggetti e sottosistemi per spostare informazioni tra i nodi
- Di contro, questo implica che bisogna fare scelte di memorizzazione, di trasferimento, e di sincronizzazione dell'informazione tra i vari sottosistemi.

Esempio MyTrip

- Dalle richieste si può dedurre che i due sottosistemi andranno in esecuzione su due nodi diversi
  - **Il PlanningSubsystem è un servizio web-based e quindi andrà su un host Internet**
  - **Il RoutingSubsystem andrà in esecuzione sul computer di bordo**

DEPLOYMENT DIAGRAM



Allocation of MyTrip subsystems to hardware (UML deployment diagram).

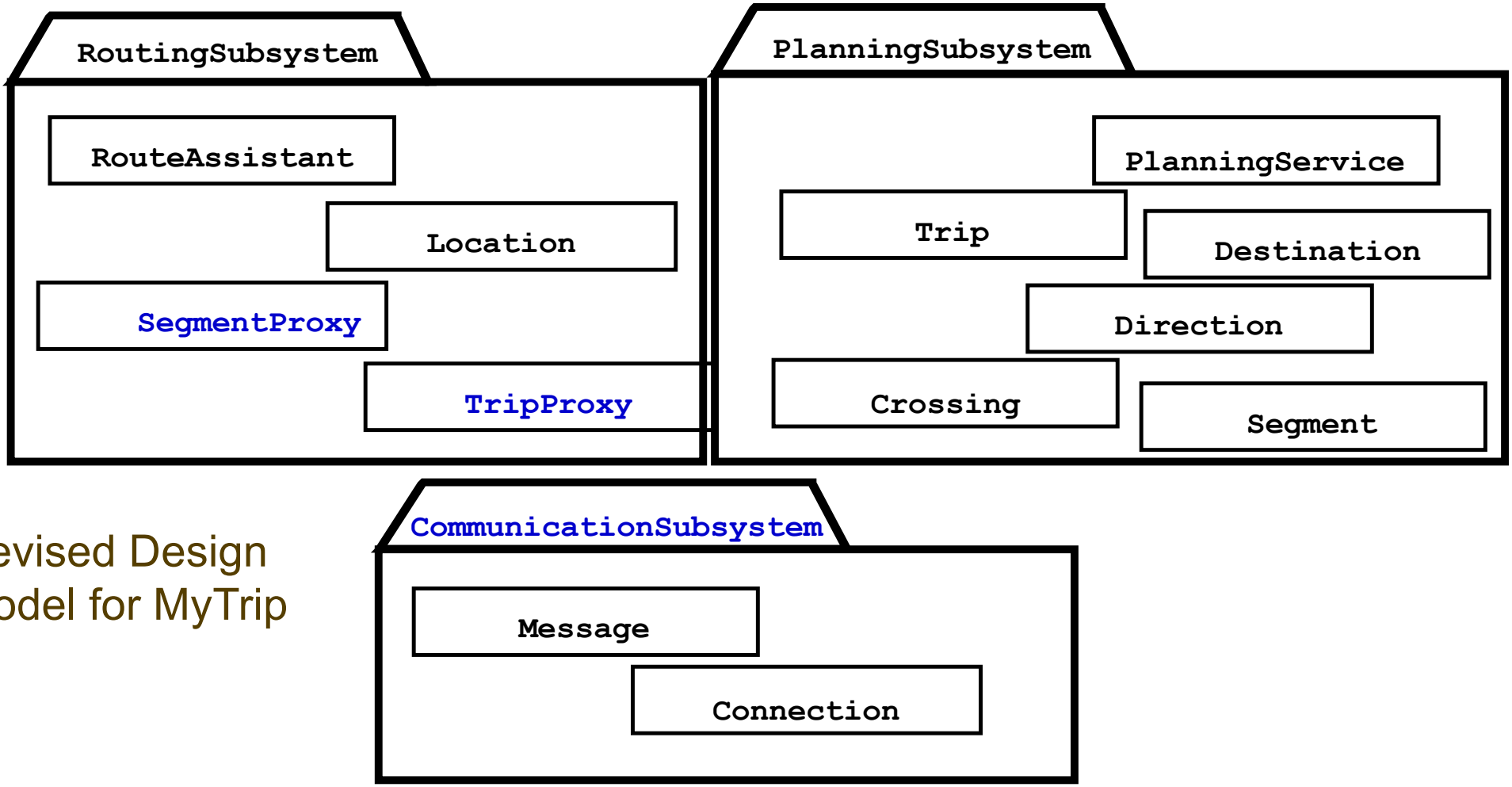
RoutingSubsystem runs on the OnBoardComputer
PlanningSubsystem runs on a WebServer.

MyTrip: Selezione macchina virtuale

- La macchina virtuale per :WebServer sarà una macchina Unix
- La macchina virtuale per :OnBoardComputer il web browser Internet Explorer

MyTrip: Identificazione di nuovi oggetti e sottosistemi

- Una volta definita la configurazione hardware, selezionate le macchine virtuali, e assegnati oggetti e sottosistemi ai nodi → nuovi oggetti e sottosistemi possono essere identificati per il trasporto dei dati tra i nodi
- RoutingSubsystem e PlanningSubsystem condividono gli oggetti Trip, Destination, Crossing, Segment e Direction. Istanze di tale classi necessitano di comunicare tramite una comunicazione wireless usando un qualche protocollo. Per supportare tale comunicazione viene creato un nuovo sottosistema **CommunicationSubsystem** localizzato su entrambi i nodi per gestire la comunicazione tra di essi
- Nel RoutingSubsystem sono memorizzati solo le informazioni sui segmenti adiacenti che costituiscono il Trip. Si creano degli oggetti surrogati di Segment e Trip nel RoutingSubsystem (classi SegmentProxy e tripProxy): nel caso di replanning del viaggio in modo trasparente tali classi richiederanno al CommunicationSubsystem di ritrovare le informazioni associate nel PlanningSubsystem



Revised Design Model for MyTrip

Communication SubSystem *is responsible for transporting objects from the PlanningSubsystem to the RoutingSubsystem*
Connection *represents an active link between the PlanningSubsystem and the RoutingSubsystem. A connection object handles exceptional cases associated with loss of network services*
Message *represents a Trip and its related Destinations, Segments, Crossings, and Directions, encoded for transport*

System design: attività

- Mappare le componenti su piattaforme e processori
- Identificare e memorizzare informazioni persistenti
- Stabilire controllo di accesso
- Progettare il flusso di controllo globale
- Identificare le condizioni limite
- Rivedere il modello del system design

Attività del System design (cont.)

- Gestione dei Dati Persistenti
  - **Quale dovrebbe essere l'informazione persistente?**
  - **Dove dovrebbe essere memorizzata?**
  - **Come accedere ai dati persistenti?**

- Potrebbe rappresentare un "collo di bottiglia"
  - **Molte funzionalità del sistema richiedono la creazione o la manipolazione di dati persistenti**
  - **Quindi l'accesso dovrebbe essere "veloce" e "affidabile"**
  - **Si devono effettuare scelte consistenti a livello di sistema**
  - **Spesso richiede la scelta di un database management system e di un *sottosistema addizionale* per la gestione dei dati persistenti**

Identificare e memorizzare le informazioni persistenti

- Le informazioni che sopravvivono ad una singola esecuzione del sistema
  - **Ad esempio a fine giornata un autore di romanzi salva il suo lavoro in un file di un elaboratore di testo. Il file può essere riaperto più tardi. Non è necessario che sia in esecuzione il word processor affinchè esista il file.**
  - **In modo simile le informazione sui dipendenti, il loro stato, il loro stipendio, sono memorizzate in un DB. Questo consente a diversi programmi che operano sulle informazioni degli impiegati in modo consistente. Inoltre, memorizzare le informazioni in un DB consente di effettuare query su grandi insiemi.**

- Gli oggetti entity identificati durante la fase di analisi sono candidati ad essere persistenti
  - **Non tutti necessariamente (es. Location e Direction sono computati dinamicamente)**
  - **Non solo (es. informazione relativa agli utenti in un sistema multi-utente, o relativa agli oggetti boundary, come preferenze delle interfacce utente, posizione delle finestre, ecc)**
  - **Analizzare tutte le classi che devono sopravvivere dopo lo shutdown (sia quello controllato che in caso di crash inattesi)**

MyTrip: Identificazione degli oggetti persistenti

- Trip e classi correlate (Crossing, Destination, PlanningService, Segment) devono essere memorizzati, ma non tutti devono essere persistenti
  - **Location e Direction sono ricomputati costantemente quando l'auto si muove**
- Informazioni sugli Autisti

Selezionare una strategia per la gestione della memorizzaione

- Quando gli oggetti persistenti sono identificati, bisogna decidere come devono essere memorizzati
  - **File**
    - **economici, semplici, memorizzazione permanente**
    - **Operazioni di basso livello (Read, Write)**
    - **Le applicazioni devono aggiungere codice per fornire un opportuno livello di astrazione**
  - **Database**
    - **potenti, facili da portare**
    - **Supportano letture e scritture multiple**

File or Database?

- Quando scegliamo un file?
  - **I dati sono voluminosi (immagini)?**
  - **Necessità di tenere traccia delle informazioni solo per un tempo breve?**
  - **La densità dell'informazione è bassa?**
- Quando scegliamo un database?
  - **Le informazioni richiedono un accesso ad un livello raffinato di dettaglio attraverso utenti multipli?**
  - **Le informazioni devono essere portate attraverso piattaforme multiple (heterogeneous systems)?**
  - **Più programmi hanno accesso alle informazioni?**

Database Relazionali

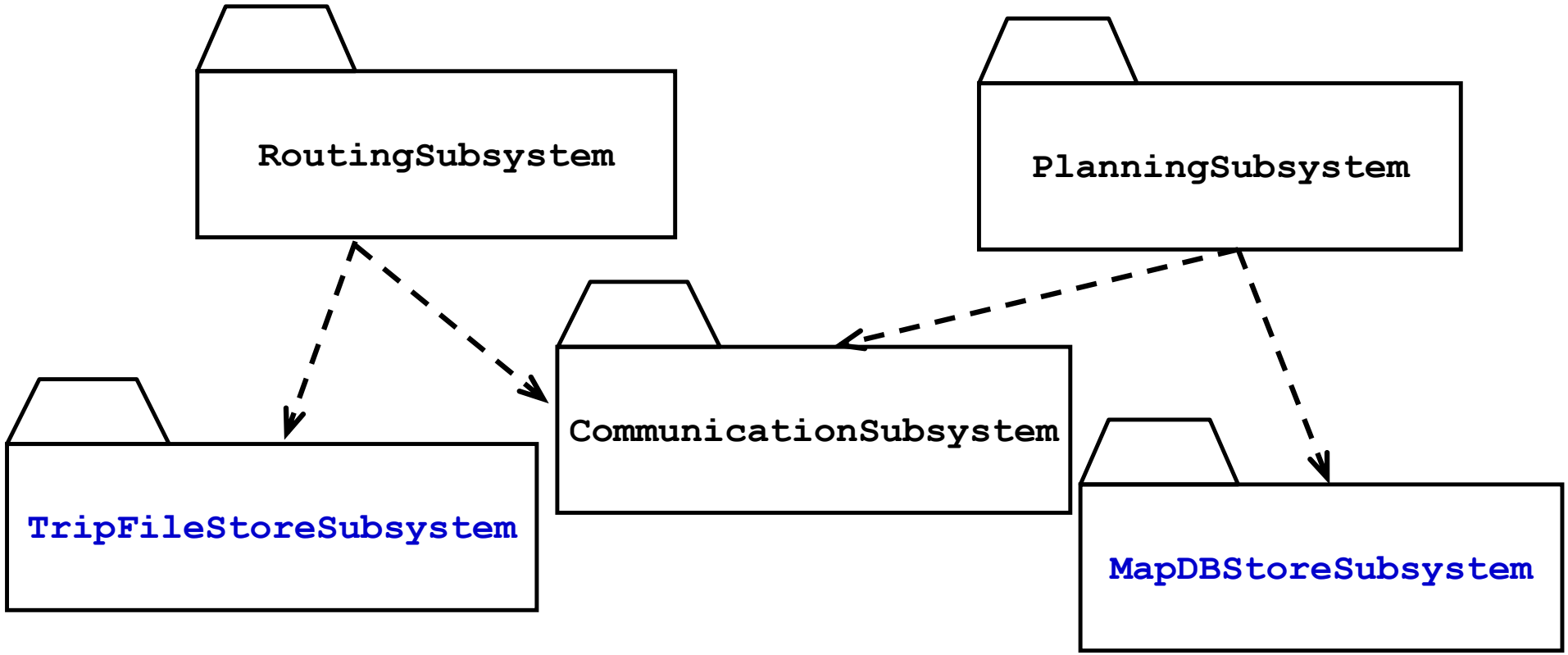
- Basati sull'algebra relazionale
- Le informazioni sono presentate come tabelle 2-dimensional. Le tabelle hanno uno dato numero di colonne e un arbitrario numero di righe
  - **Ogni colonna rappresenta un attributo**
  - **Chiave primaria: combinazione di attributi che identifica univocamente una riga nella tabella. Ogni tabella dovrebbe avere un'unica chiave primaria**
  - **Chiave esterna: riferimento alla chiave primaria di un'altra tabella**
- SQL è il linguaggio standard per definire e manipolare le tabelle
- I database commerciali supportano i vincoli
  - **Integrità referenziale, per esempio, significa che i riferimenti a entrate di altre tabelle devono esistere realmente**

Database Object-Oriented

- Forniscono servizi simili ad un database relazionale
- Supportano tutti i concetti fondamentali della modellazione objetct-oriented
  - **Classi, Attributi, Metodi, Associazioni, ereditarietà**
- Problema: Efficienza

MyTrip: Identificazione degli oggetti persistenti e memorizzazione

- Il Trip corrente sarà memorizzato in un file su un disco rimovibile per consentire il recovery del Trip nel caso l'autista spenga l'auto prima di raggiungere la destinazione finale → TripFileStoreSubsystem
  - **Il file rappresenta la soluzione migliore (più semplice e più efficiente) poiché il RoutingSubsystem memorizzerà solo Trip completi prima dello shutdown e caricherà il file allo start-up**
- Nel PlanningSubsystem i Trip saranno memorizzati in un database → MapDBStoreSubsystem
  - **Tale sottosistema consentirà di gestire più Trips per molti autisti e le mappe per generare i Trip**
  - **L'uso del database consente di eseguire query complesse su questi dati**



Subsystem decomposition of MyTrip after deciding on the issue of persistent data stores (UML class diagram, packages collapsed for clarity).

TripFileStoreSubsystem ***is responsible for storing trips files on the onboard computer. Because this functionality is only used for storing trips when the car shuts down, this subsystem only supports the fast storage and loading of whole trips***

MapDBStoreSubsystem ***is responsible for storing maps and trips in a database for the PlanningSubsystem. This subsystem supports multiple concurrent drivers and planning agents.***

System design: attività

- Mappare le componenti su piattaforme e processori
- Identificare e memorizzare informazioni persistenti
- Stabilire controllo di accesso
- Progettare il flusso di controllo globale
- Identificare le condizioni limite
- Rivedere il modello del system design

Attività del System design (cont.)

- Controllo di Accesso in sistemi multi-utenti
  - **Chi può accedere alle informazioni?**
  - **Il controllo di accesso può cambiare dinamicamente?**
  - **Come è specificato e realizzato, il controllo di accesso?**

- Deve essere consistente
  - **La politica utilizzata per specificare chi può e chi non può accedere a certe informazioni dovrebbe essere la stessa per tutti i sottosistemi**

Controllo di accesso

- In sistemi multi-utente, differenti attori hanno accesso a diverse funzionalità e informazioni.
  - **Un attore qualsiasi può accedere solo ai dati che crea**
  - **Un amministratore ha accesso a tutti i dati del sistema e a quelli degli utenti**
- Durante l'analisi si è modellato questa distinzione associando differenti use case a differenti attori
- Durante il system design si modellano gli accessi determinando quali oggetti sono condivisi tra gli attori
- Questo vuol dire:
  - **Descrivere i diritti di accesso per classi differenti di attori**
  - **Descrivere come gli oggetti "vigilano" contro gli accessi non autorizzati**
  - **Meccanismi di autenticazione, meccanismi di crittografia dei dati**

Questioni per il controllo di accessi

- Il sistema richiede un meccanismo di autenticazione?
- Se sì, qual'è lo schema di autenticazione?
  - **Nome utente e password? Lista dei controlli di accesso?**

- Quale è l'interfaccia utente per l'autenticazione?

MyTrip:

- La memorizzazione di mappe e Trip per diversi utenti nello stesso database introduce problemi di sicurezza
  - **Dobbiamo assicurare che i Trip siano inviati solo agli autisti che li hanno creati (consistente con il design goal di security)**

  → Occorre modellare la classe Driver e associare una classe Trip

  →Il PlanningSubsystem diventa responsabile dell'autenticazione del driver prima dell'invio del Trip

  → Decidiamo di crittografare il traffico di comunicazione tra il RoutingSubsystem e PlanningSubsystem e sarà realizzato dal CommunicationSubsystem

(revision to the design model stemming from the decision to authenticate Dirvers and encrypt communication traffic. The text added to the model is in italics

- **CommunicationSubsystem** is responsible for transporting Trips from the PlanningSubsystem to the RoutingSubsystem. *The CommunicationSusbsystem uses the Driver associated with the trip being transported for selecting a key and encrypting the communication traffic*.

- **PlanningSubsystem** is responsible for constructing a Trip connecting a sequence of destinations. The PlanningSubsystem is also responsible for responding to replan requests from RoutingSubsystem. *Prior to processing any requests, the PlanningSusbsystem authenticates the Driver from the RoutingSubsystem. The authenticated Driver is used to determine which Trips can be sent to the corresponding RoutingSubsystem*.

- ***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*.

Matrice di Accesso

- Matrice di accesso modella i diritti di accesso su una classe. Le righe rappresentano attori, le colonne le classi. Ogni entry (attore, classe) contiene le operazioni consentite da quell'attore sulle istanze di quella classe.
- Diversi modi di rappresentarli (in base a design goal)
- **Global access table,** rappresenta esplicitamente ogni cella nella matrice come una tupla (actor,class,operation)
  - **Se una tale tupla non c'è allora l'accesso è negato**
  - **Richiede molto spazio per la memorizzazione**
- **Access control list,** associa una lista di coppie (actor,operation) per ogni classe a cui si può fare accesso
  - **Ogni volta che si accede ad un oggetto, la sua lista degli accessi è controllata per il corrispondente attore e operazione**
  - **Consente di rispondere facilmente a domande del tipo "chi accede a questo oggetto?"**
- **Capability,** associa una coppia (class,operation) con un attore
  - **una capality consente ad un attore di accedere ad un oggetto della classe descritta nella capability**
  - **Consente di rispondere facilmente a domande del tipo "a quali oggetti accede questo attore?"**

System design: attività

- Mappare le componenti su piattaforme e processori
- Identificare e memorizzare informazioni persistenti
- Stabilire controllo di accesso
- Progettare il flusso di controllo globale
- Identificare le condizioni limite
- Rivedere il modello del system design

Attività del System design (cont.)

- Controllo del Flusso
  - **Come è gestita la sequenza delle operazioni?**
  - **Il sistema è guidato da eventi?**
  - **Il sistema può gestire più di un'interazione utente alla volta?**

- La scelta del controllo del flusso ricade sulle interfacce delle componenti
  - **nel caso di un controllo guidato da eventi, i sottosistemi forniranno un gestore degli eventi.**
  - **nel caso di un controllo basato su threads, i sottosistemi devono garantire la mutua esclusione nelle sezioni critiche**

Flusso di controllo globale

- È la sequenza di azioni nel sistema.
- In un sistema Object-Oriented, la sequenza delle azioni include le decisioni su quali operazioni dovrebbero essere eseguite e in quale ordine
- Queste decisioni si basano su eventi esterni generati da attori o dal trascorrere del tempo
- È un problema che riguarda la fase di system design
  - **Durante l'analisi, assumiamo che tutti gli oggetti siano eseguiti simultaneamente, eseguendo le operazioni ogni volta che necessitano di eseguirle**
  - **Durante il system design bisogna tener conto che non tutti gli oggetti hanno un proprio processore a disposizione**

Meccanismi per il flusso di controllo globale

- Procedure-driven control
  - **I controlli risiedono nel codice del programma.**
    - **Example: il programma principale chiama le procedure dei sottosistemi**
    - **Operazioni aspettano che un attore fornisca un input**
  - **Semplice, facile da costruire per sistemi procedurali**
  - **Usato nei sistemi legacy, e quelli scritti con linguaggi procedurali**
- Event-driven control
  - **Il ciclo principale attende un evento esterno. Quando un evento si verifica, è spedito all'oggetto appropriato (sulla base dell'info associata all'evento)**
  - **Il controllo risiede in un dispatcher che chiama le funzioni di sottosistema.**
  - **Flessibile, buono per le interfacce utenti**
- Threads
  - **Rappresentano la versione concorrente del Procedure-driven control**
  - **Il sistema può creare un numero arbitrario di threads, ognuno risponde ad un differente evento.**
  - **Se un threads necessita di informazioni addizionali aspetta un input da uno specifico attore**

Concorrenza

- Identificare i processi concorrenti e trattare i problemi della concorrenza
- Design goal: tempo di risposta, performance.
- Due oggetti sono concorrenti se possono ricevere eventi nello stesso istante senza interagire

- Oggetti concorrenti dovrebbero essere assegnati a differenti thread di controllo

Questioni di concorrenza (cont.)

- Quali oggetti del modello sono indipendenti?
- Che tipi di thread sono identificabili?
- Il sistema fornisce accesso a utenti multipli?
- Può una singola richiesta del sistema essere decomposta in richieste multiple? Queste richieste possono essere gestite in parallelo?

Gestire il flusso di controllo globale

- Quando un meccanismo per il flusso di controllo è stato scelto, esso viene realizzato con un insieme di uno o più oggetti control.
- Il ruolo degli oggetti control è quello di memorizzare gli eventi esterni, memorizzare il loro stato temporaneo, gestire la giusta sequenza di chiamate di operazioni sugli oggetti boundary e entity associati con gli eventi esterni
- Localizzare le decisioni sul flusso di controllo per uno use case in un singolo oggetto non solo consente di avere un codice più comprensibile, ma rende anche il sistema più flessibile ai cambiamenti nell'implementazione del flusso di controllo

System design: attività

- Mappare le componenti su piattaforme e processori
- Identificare e memorizzare informazioni persistenti
- Stabilire controllo di accesso
- Progettare il flusso di controllo globale
- Identificare le condizioni limite
- Rivedere il modello del system design

Attività del System design (cont.)

- Condizioni limite
  - **Come è avviato il sistema?**
  - **Come è interrotto?**
  - **Come sono individuati e gestiti i casi eccezionali?**

- L'avvio e l'interruzione del sistema spesso rappresentano molta della complessità di un sistema, specialmente nei sistemi distribuiti
  - **Incidono sulle interfacce di tutti i sottosistemi**

Condizioni limite

- Nella fase di system design bisogna determinare le condizioni limite per il sistema che si sta sviluppando:
  - **inizializzazione**
    - **Descrive come il sistema è portato da uno stato non inizializzato ad uno stato stabile ("startup use cases").**
  - **terminazione**
    - **Descrive quali risorse sono rilasciate e quali sistemi sono notificati della terminazione ("termination use cases").**
  - **fallimento**
    - **Molte possibili cause: errori, problemi esterni (e.g, alimentazione elettrica).**
    - **Buoni sistemi progettano i fallimenti fatali ("failure use cases").**
- Non vengono trattati nella fasi di analisi poichè molte condizioni limite sono determinate da decisioni di design

Questioni relative alle condizioni limite

- Inizializzazione
  - **Come il sistema parte?**
    - **Quali informazioni sono necessarie all'avvio del sistema?**
    - **Quali servizi devono essere registrati?**
  - **Cosa fanno le interfacce utenti all'avvio del sistema?**
    - **Come si presentano all'utente?**
- Terminazione
  - **Possono i singoli sistemi terminare?**
  - **Gli altri sistemi sono notificati se un sottosistema termina?**
  - **Gli aggiornamenti locali sono comunicati al database?**
- Fallimento
  - **Come il sistema si comporta quando un nodo o un link di comunicazione fallisce? Ci sono link di comunicazione di backup?**
  - **Come il sistema recupera da un fallimento (recovery)? È differente dall'inizializzazione?**

Indentificazione delle condizioni limite

- In generale, gli use case per le condizioni limite vengono identificati esaminando ogni sottosistema e ogni oggetto persistente:
  - **Configurazione.** Per ogni oggetto persistente, si esamina in quale use case è creato o distrutto. Per ogni oggetto non creato o non distrutto in uno degli use case, si aggiunge uno use case invocato dall'amministratore di sistema
  - **Avvio e terminazione.** Per ogni componente si aggiungono tre use case: start, shutdown, configure.
  - **Gestione eccezioni**. Per ogni tipo di fallimento di componente, si decide come il sistema debba reagire. Documentiamo ognuna di queste decisioni con uno use case eccezionale che estende lo use case di base.

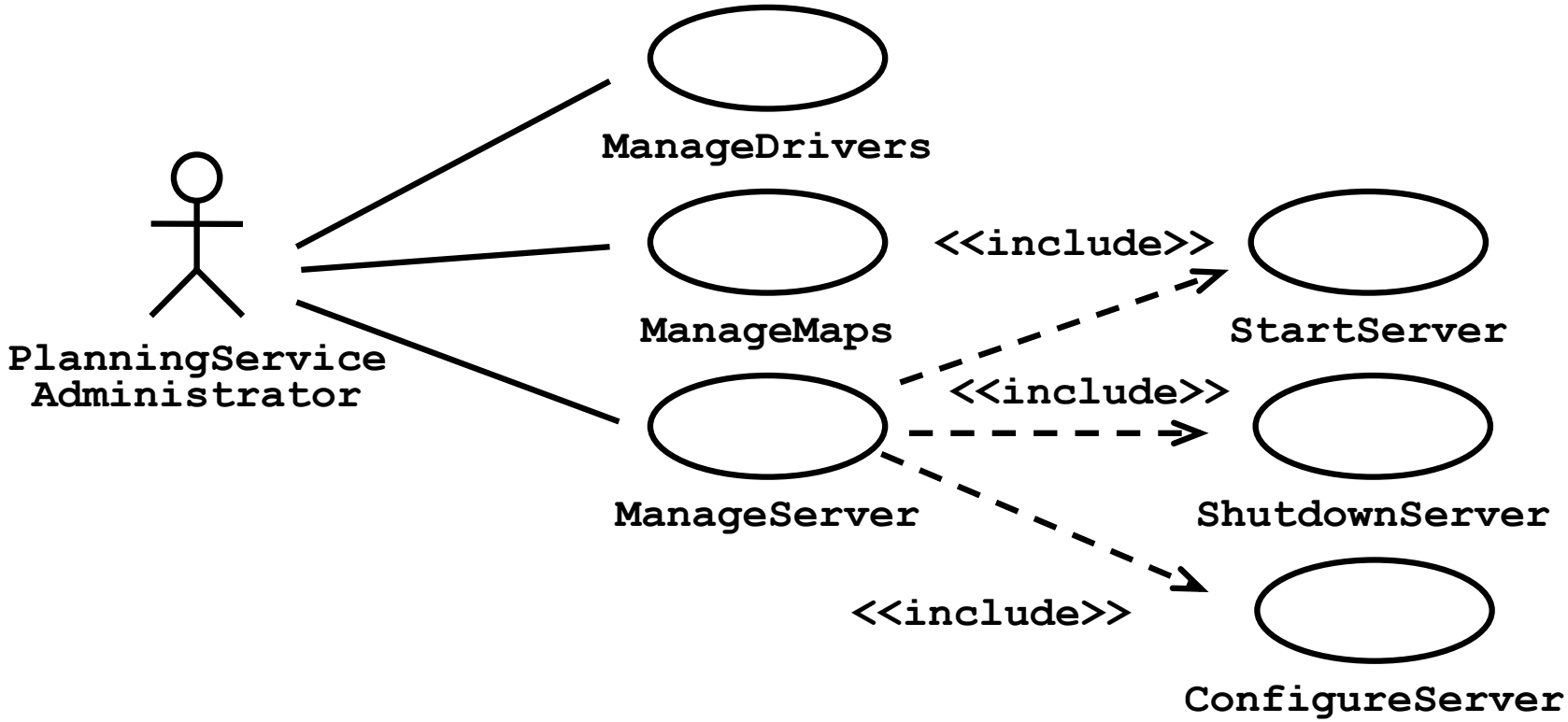
Eccezioni

- Un'eccezione è un evento o errore che si verifica durante l'esecuzione del sistema. Sono causate da tre differente risorse:
  - **Un fallimento hardware**
    - **L'hardware "invecchia" e fallisce. Crash dell'hard disk e perdita dei dati; fallimento di un link di rete provoca la disconnessione di due nodi**
  - **Un cambiamento nel sistema operativo**
    - **L'ambiente influenza il lavoro del sistema. Una pausa nell'erogazione della corrente elettrica può far andare giù il sistema, finché una batteria di back-up non è attivata.**
  - **Un fallimento del software**
    - **Un errore si verifica perché il sistema o una delle sue componenti contiene un errore commesso durante la fase di progetto**

Gestione delle eccezioni

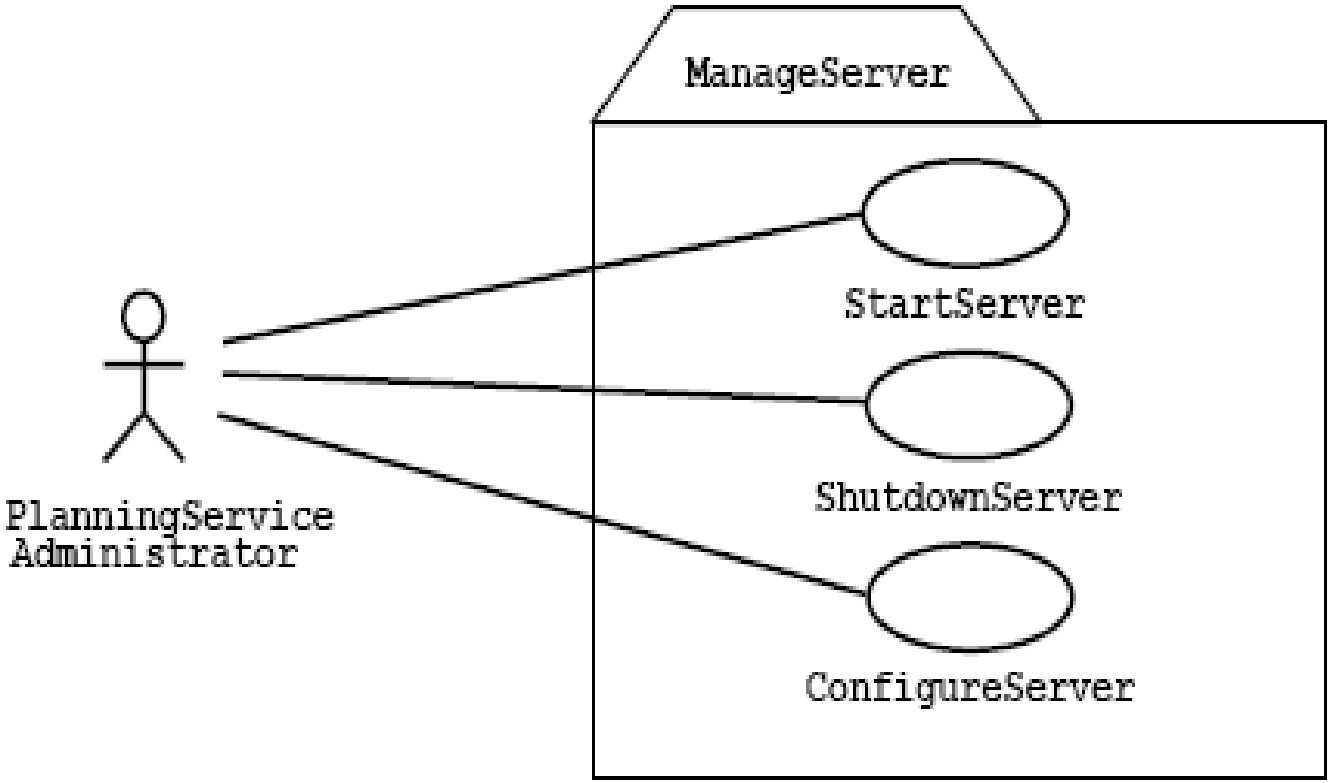
- Un meccanismo attraverso cui il sistema tratta le eccezioni.
- Nel caso di un errore utente, il sistema mostra all'utente un messaggio così che egli possa far fronte all'errore
- Nel caso di un fallimento di un link di comunicazione, il sistema dovrebbe salvare lo stato temporaneo così che possa essere recuperato quando la rete ritorna ad essere funzionante
- Si raffinano gli use case in modo da descrivere le situazioni in cui si possono verificare le eccezioni
- Durante il System design si gestiscono le eccezioni a livello di componente, mentre durante Object design a livello degli oggetti

Administration use cases for MyTrip (UML use case diagram).



***ManageDrivers** is invoked to add, remove, modify, or read data about drivers (e.g., user name and password, usage log, encryption key generation).*
***ManageMaps** is invoked to add, remove, or update maps that are used to generate trips.*
***ManageServer** includes all the functions necessary to start up and shutdown the server.*

ManageServer Use Case: Correct representation



Use case **StartServer**

1. The PlanningServiceAdministrator logs the server machine
2. Upon successful login, the PlanningServiceAdministrator executes the startPlanningService command.
3. If the PlanningService was previosly shutdown normally, the server reads the list of legitimate Drivers and the index of active Trips and Maps. If the PlanningService had crashed, it notifies the PlanningServiceAdministrator and performs a consistency check on the MapDBStore.
4. The PlanningService is available and waits for connections from RoutingAssistants.

Revised description for MapDBStoreSubsystem based on the additional StartServer use case

MapDBStoreSubsystem ***is responsible for storing maps and trips in a database for the PlanningSubsystem. This subsystem supports multiple concurrent drivers and planning agents.*** **When starting up, the MapDBStore detects if it was properly shutdown. If not, it performs a consistent check on the Maps and Trips and repairs corrupted data if necessary**

System design: attività

- Mappare le componenti su piattaforme e processori
- Identificare e memorizzare informazioni persistenti
- Stabilire controllo di accesso
- Progettare il flusso di controllo globale
- Identificare le condizioni limite
- Rivedere il modello del system design

Rivedere il System design

- Come l'analisi, il sistema design è un'attività iterativa.
- A differenza dell'analisi, non ci sono agenti esterni (come il cliente) per revisionare il lavoro e le scelte fatte
- Il manager del progetto e gli sviluppatori devono organizzare un processo di revisione per sostituirsi al cliente

- In aggiunta, per raggiungere gli obiettivi del progetto ("design goal"), bisogna assicurare che il sistem design è:
  - **Corretto**
  - **Completo**
  - **Consistente**
  - **Realistico**

Il system design è corretto?

- Ci si pone le seguenti domande:
  - **Ogni sottosistema può essere "tracciato" su uno use case o una richiesta non funzionale?**
  - **Ogni use case può essere mappato su un insieme di sottosistemi?**
  - **Ogni richiesta non funzionale è analizzata nel system design?**
  - **Ogni attore ha una politica di accesso?**

Il system design è completo?

- Ci si pone le seguenti domande:
  - **Le condizioni limite sono state gestite?**
  - **C'è una rivisitazione degli use case per identificare funzionalità non realizzate nel sistema?**
  - **Tutti gli use case sono stati esaminati ed è stato assegnato un oggetto control?**
  - **Ogni aspetto del system design è stato analizzato?**

Il system design è consistente?

- Ci si pone le seguenti domande:
  - **È stata assegnata una priorità ai design goal in conflitto?**
  - **Ci sono design goal che violano requisiti non funzionali?**
  - **Ci sono classi o sottosistemi con lo stesso nome?**
  - **Le collezioni di oggetti sono scambiate in modo consistente tra i sottosistemi?**

Il system design è realistico?

- Ci si pone le seguenti domande:
  - **Le nuove tecnologie sono incluse nel sistema?**
  - **I requisiti di performance e di affidabilità sono esaminati durante la decomposizione in sottosistemi**
  - **La concorrenza è stata analizzata?**

Il system design è leggibile?

- Ci si pone le seguenti domande:
  - **I nomi dei sottosistemi sono comprensibili?**
  - **Entità con nomi simili rappresentano concetti simili?**
  - **Tutte le entità sono descritte con lo stesso livello di dettaglio?**

Assegnare le responsabilità

- Il system design è una fase che è incentrata intorno al lavoro fatto dall' "architecture team"
  - **Il team responsabile della decomposizione del sistema in sottosistema e dell'assegnazione ad ogni team delle proprie responsabilità, e della selezione degli sviluppatori che devono realizzare le diverse componenti**
- **Architetto**, riveste il ruolo principale nel system design.
  - **Assicura la consistenza nelle decisioni di design e nello stile delle interfacce**
  - **Assicura la consistenza del design tra i team per la gestione delle configurazioni e del testing**
- **I liaisons** sono rappresentanti dei team che lavorano ai diversi sottosistemi.
  - **Raccolgono le informazioni da e verso il loro team e negoziano i cambiamenti nelle interfacce.**
  - **Durante il system design si concentrano sui servizi dei sottosistemi, durante la fase di implementazione si concentrano sui dettagli delle API**
  - **Il numero dei sottosistemi determina la grandezza di questo team**

Iterazione nel System design

- Come nel caso dei requisiti, la fase di system design è affrontata in modo iterativo e incrementale (successione di cambiamenti)
- I cambiamenti devono però essere controllati, per prevenire il caos, specialmente in progetti grossi dove il numero di partecipanti è elevato
- Tre tipi di iterazioni
  - **Cambiamenti relativi a decisioni che riguardano la decomposizione del sistema in sottosistemi**
  - **Cambiamenti relativi a decisioni che riguardano le interfacce (realizzazione di prototipi per valutare specifiche caratteristiche)**
  - **Cambiamenti relativi a decisioni da prendere per gestire condizioni di errore che sono scoperte in fasi avanzate del progetto**