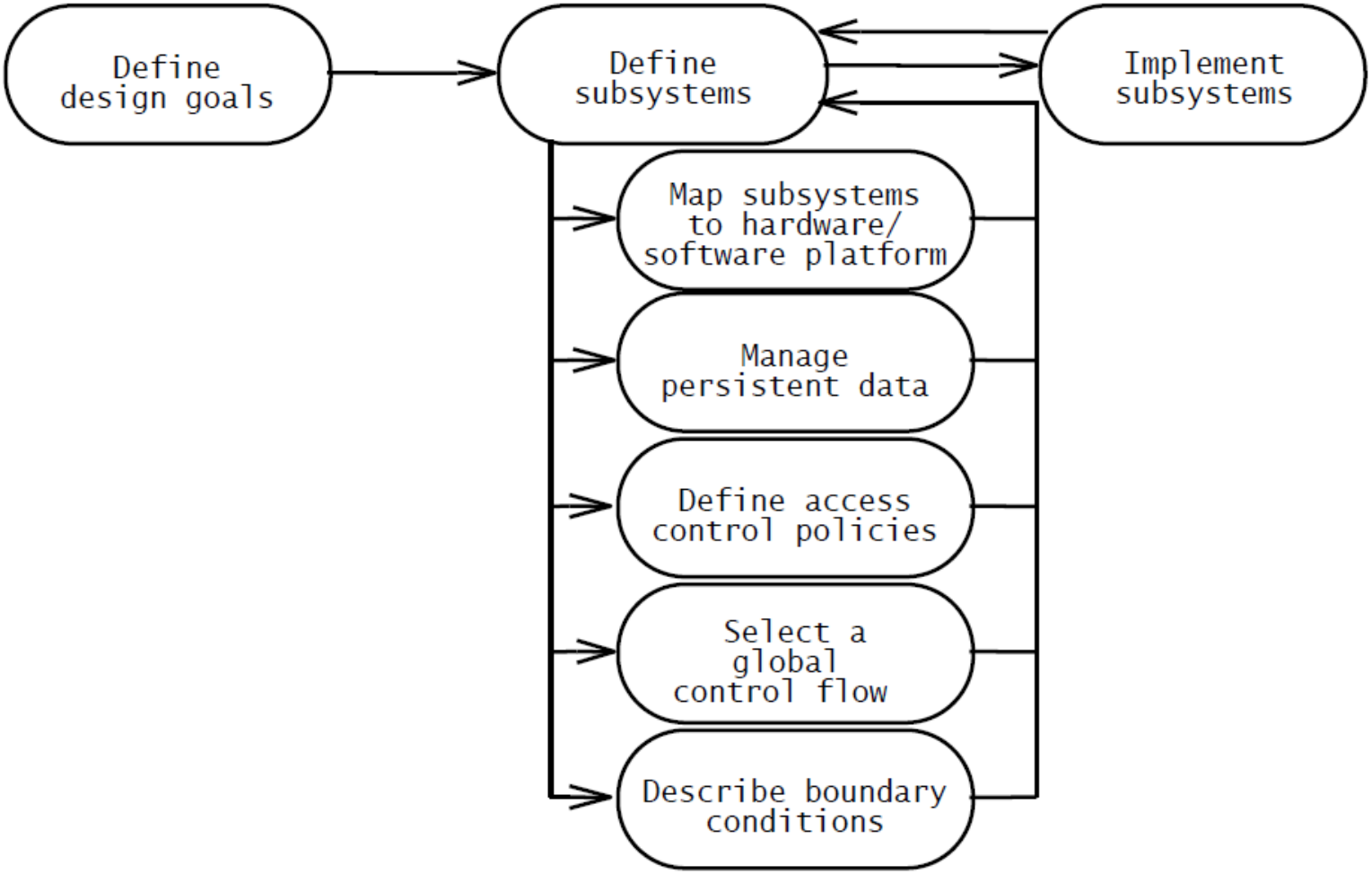
Chapter 7
System Design: Addressing Design Goals

The Activities of System Design



The Activities of System Design

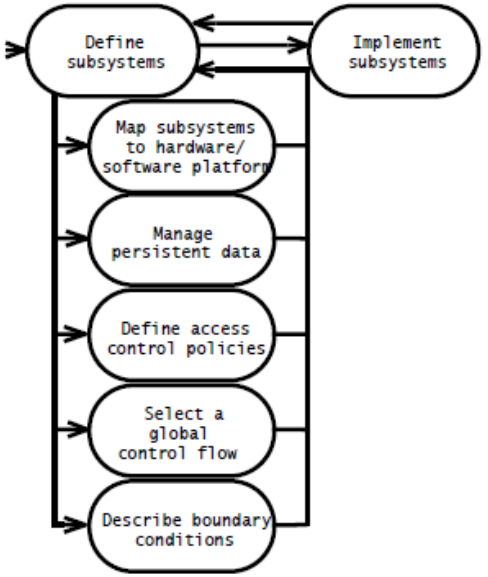
Hardware/software mapping
- What is the hardware configuration of the system
- How to map software (components) to hardware nodes?

Data Management
- Which data should be persistent?
- Where to store persistent data? How to access?

Access Control
- Who can access which data? Can access control change dynamically?
- How is access control specified? ...

Control Flow
- How does the system sequence operations?
- Is the system event-driven? Can it handle more than one user interaction? Etc.

Boundary Conditions
- How is the system initialized?
- How is it shut down?
- How are exceptional cases detected?
- Etc..



Hardware and Software Mapping

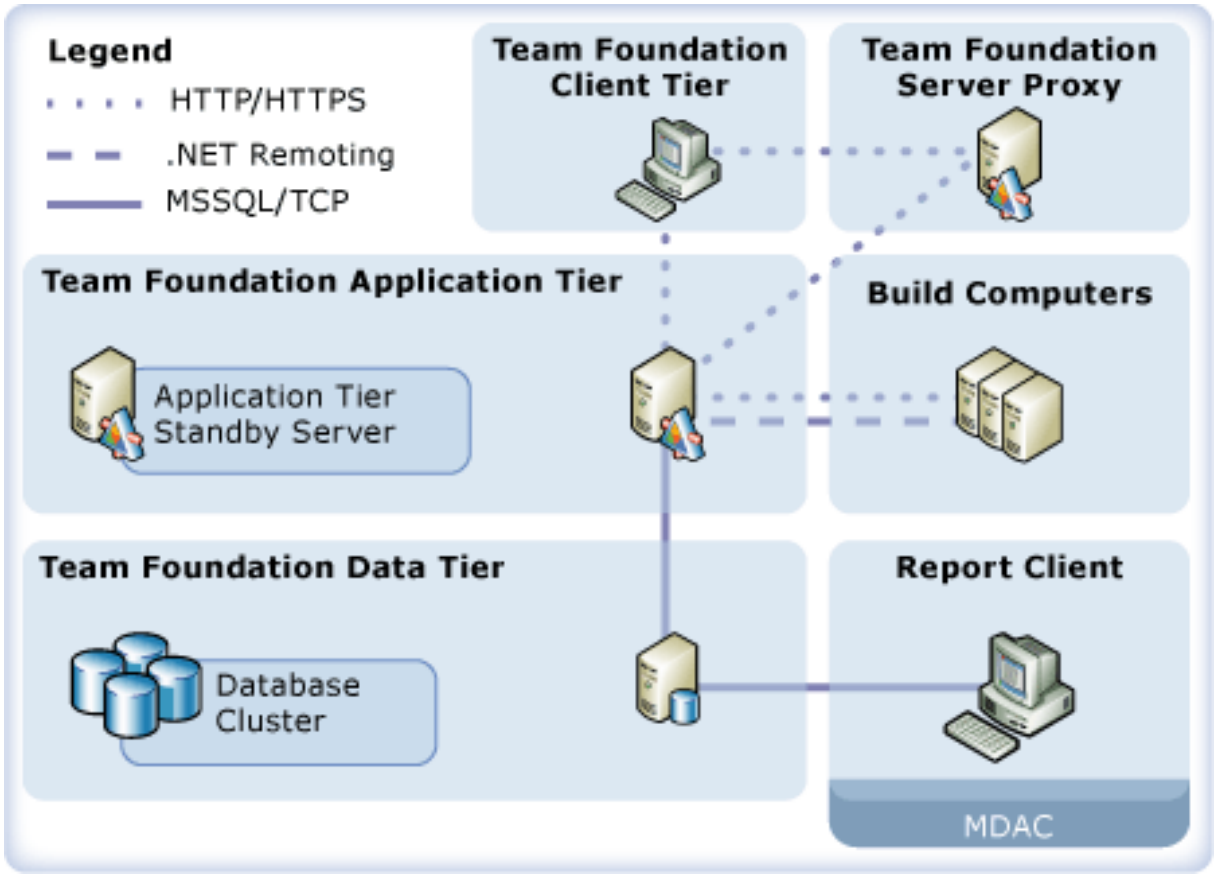
This activity addresses two questions:

- How shall we realize the subsystems: Hardware or Software?
- How is the object model mapped on the chosen hardware & software?
  - Mapping Objects onto Reality: Processor, Memory, Input/Output
  - Mapping Associations onto Reality: Connectivity

Much of the difficulty of designing a system comes from meeting externally-imposed hardware and software constraints.

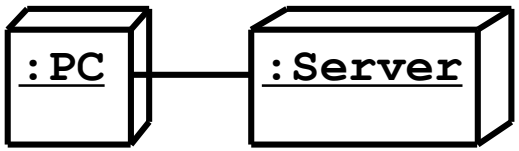
- Certain tasks have to be at specific locations

Real Life Example – Informal Notation



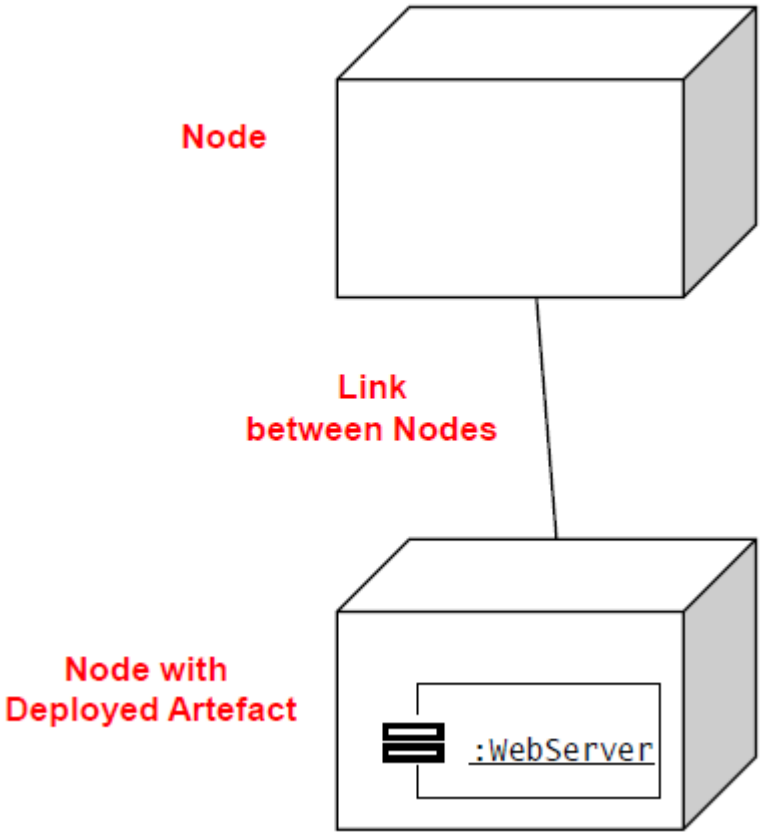
Deployment Diagram

- A **deployment diagram** serves to **model the physical deployment** of software artefacts on deployment targets
  - Illustrates the distribution of components at run-time.
  - Deployment diagrams use nodes and connections to depict the physical resources in the system
- Deployment diagrams are useful for showing a system design after these system design decisions have been made:
  - Subsystem decomposition
  - Concurrency
  - Hardware/Software Mapping

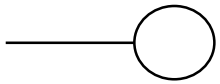
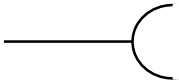


Deployment Diagram

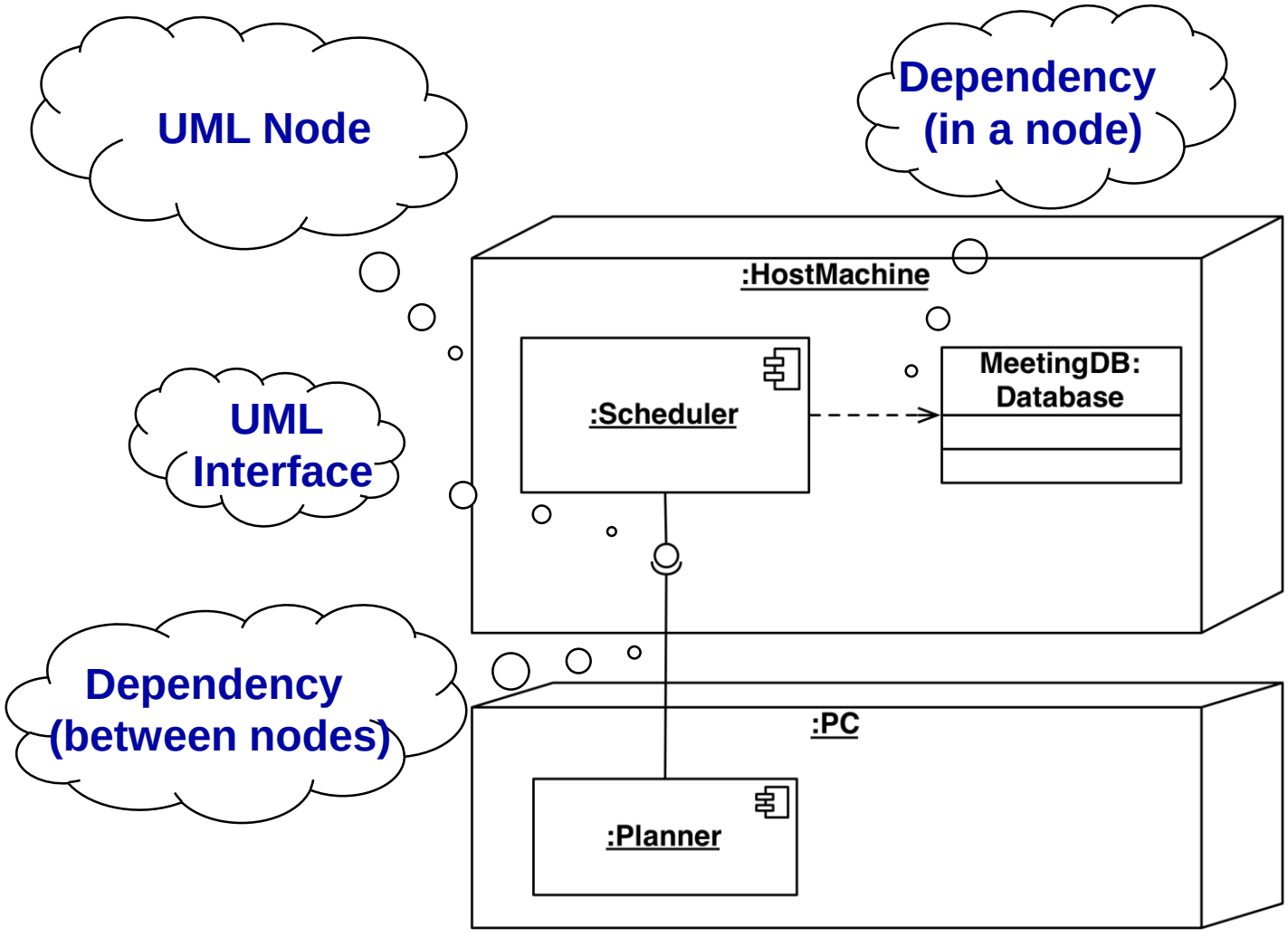
- A deployment diagram is a graph of nodes and connections ("communication associations")
  - Nodes are shown as 3-D boxes
  - Connections between nodes are shown as solid lines
  - Nodes may contain components
    - Components can be connected "lollipops" and "grabbers"
    - Components may contain objec (indicating that the object is pa of the component).



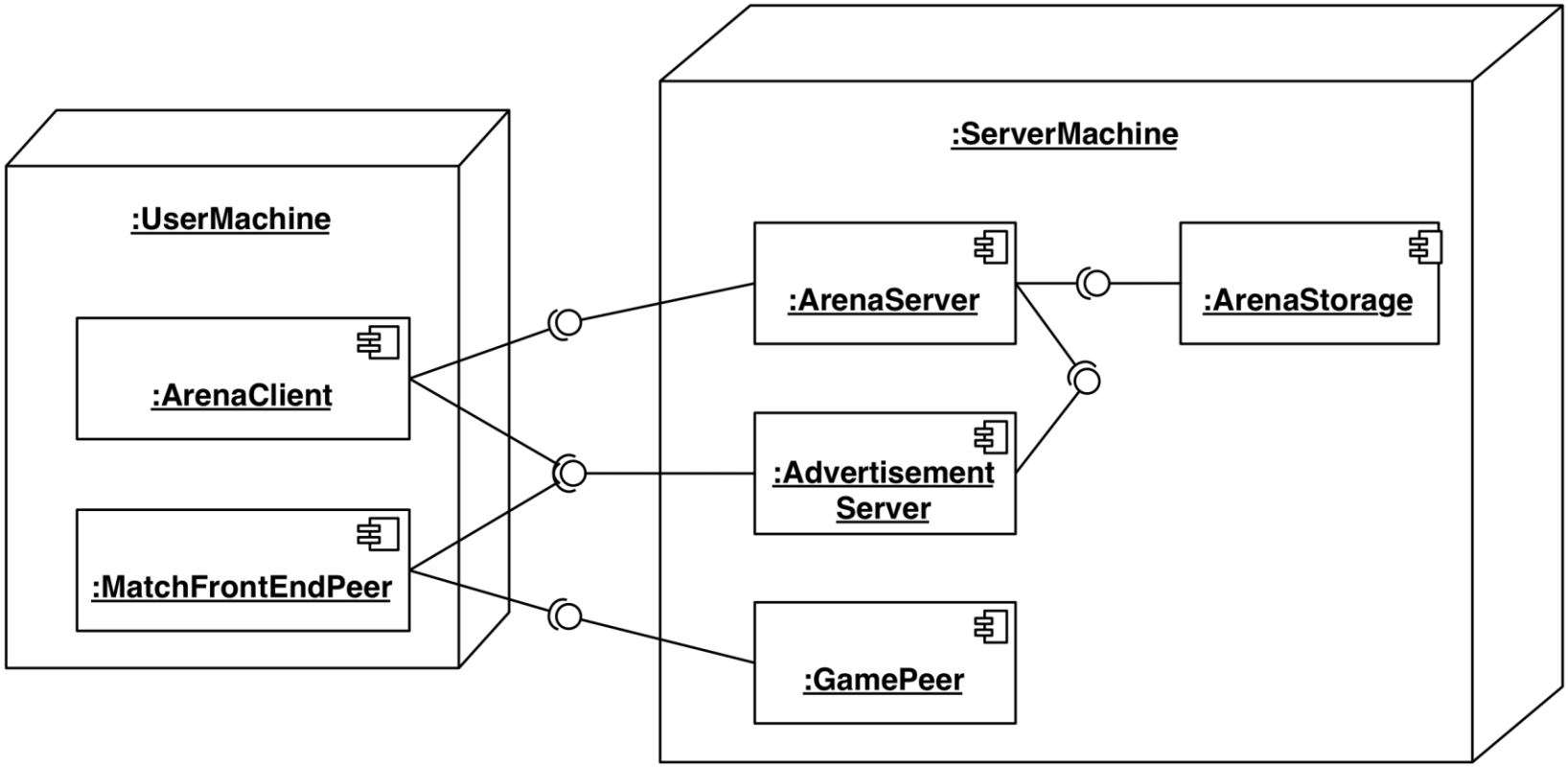
UML Interfaces: Lollipops and Sockets

- A UML interface describes a group of operations used or created by UML components.
  - There are two types of interfaces: provided and required interfaces.
    - A provided interface is modeled using the lollipop notation 
    - A required interface is modeled using the socket notation. 
- A port specifies a distinct interaction point between the component and its environment.
  - Ports are depicted as small squares on the sides of classifiers.

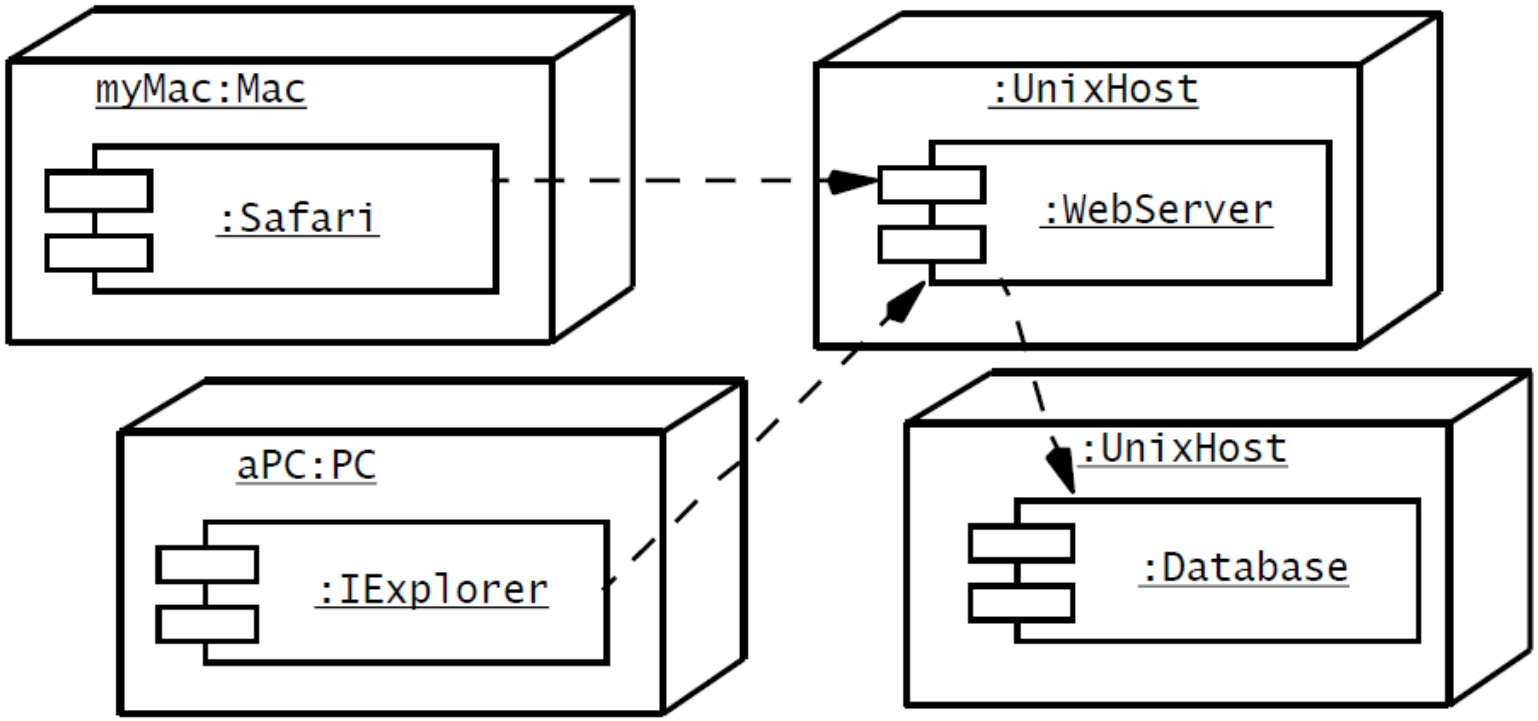
Deployment Diagram Example



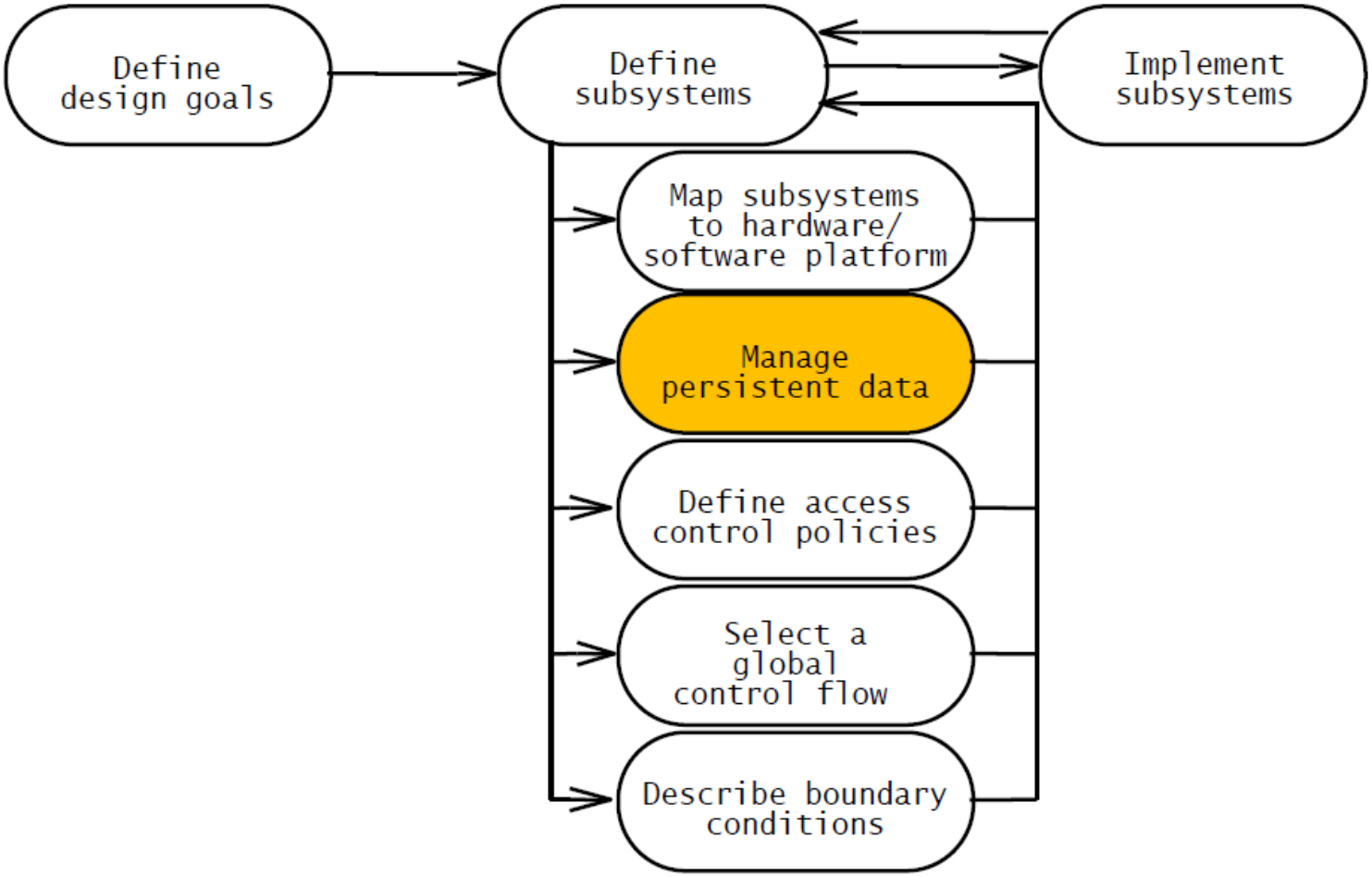
ARENA Deployment Diagram



Deployment Diagram - Example



Managing Persistent Data



5. Data Management

- Some objects in the system model need to be persistent:
  - Values for their attributes have a lifetime longer than a single execution
  - Without this capability, data would only exist in RAM, and would be lost when this RAM loses power, such as on computer shutdown.
  - This is achieved in practice by storing the data in non-volatile storage such as a hard drive or flash memory.
- A persistent object can be realized with one of the following mechanisms:
  - Filesystem:
    - If the data are used by multiple readers but a single writer
  - Database:
    - If the data are used by concurrent writers and readers.

Data Management Questions

- When should you choose a file?
  - Are the data voluminous (bit maps)?
  - Do you have lots of raw data (core dump, event trace)?
  - Do you need to keep the data only for a short time?
  - Is the information density low (archival files,history logs)?
- When should you choose a database?
  - Do the data require access at fine levels of details by multiple users?
  - Must the data be ported across multiple platforms (heterogeneous systems)?
  - Do multiple application programs access the data?
  - Does the data management require a lot of infrastructure?

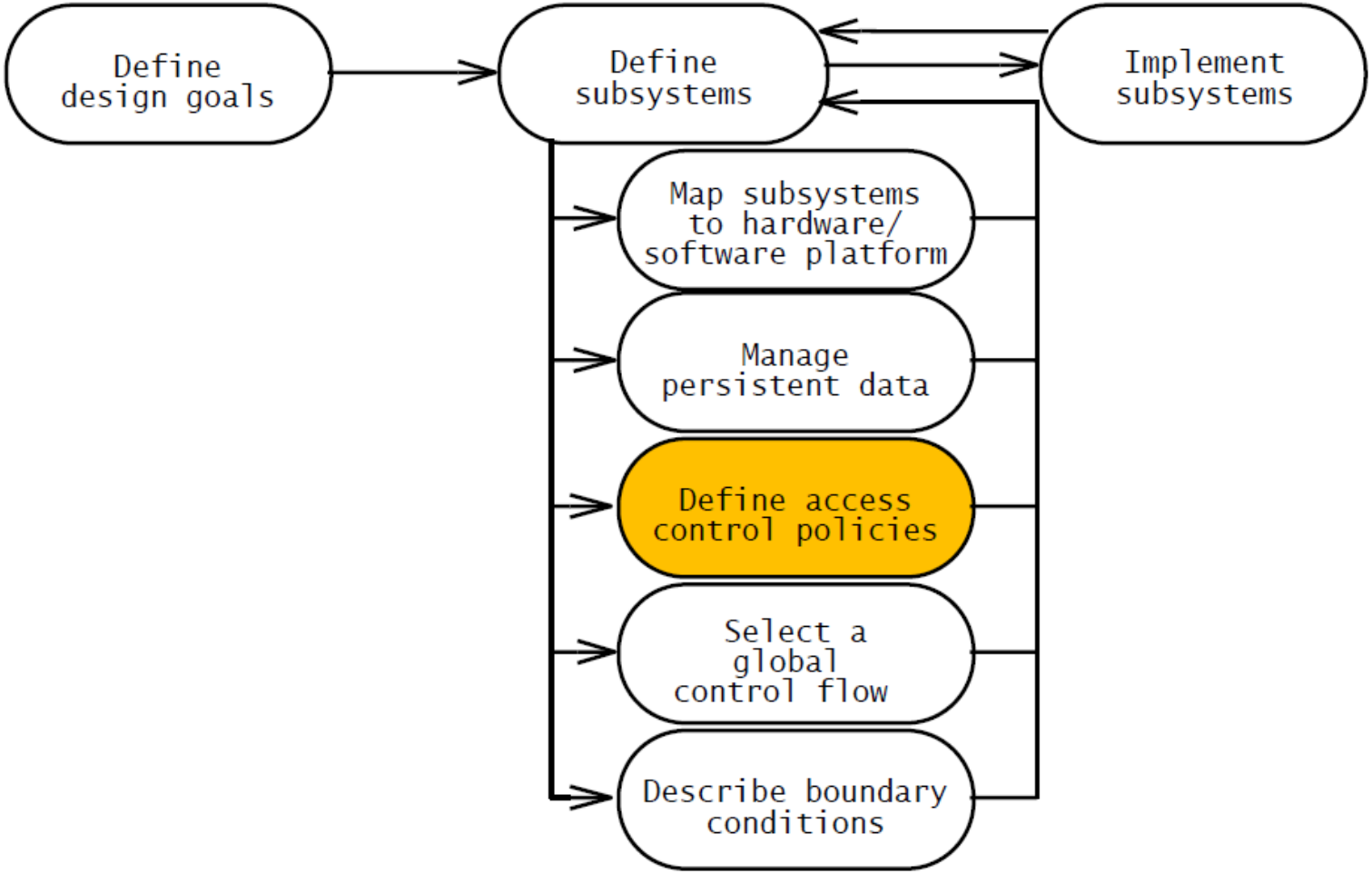
Issues to consider when selecting a database

- Storage space
  - Database require about triple the storage space of actual data
- Response time
  - Mode databases are I/O or communication bound (distributed databases). Response time is also affected by CPU time, locking contention and delays from frequent screen displays
- Locking modes
  - Pessimistic locking: Lock before accessing object and release when object access is complete
  - Optimistic locking: Reads and writes may freely occur (high concurrency!) When activity has been completed, database checks if contention has occurred. If yes, all work has been lost.
- Administration
  - Large databases require specially trained support staff to set up security policies, manage the disk space, prepare backups, monitor performance, adjust tuning.

Mapping Object Models

- UML object models can be mapped to relational databases
- The mapping:
  - Each class is mapped to its own table
  - Each class attribute is mapped to a column in the table
  - An instance of a class represents a row in the table
  - One-to-many associations are implemented with a buried foreign key
  - Many-to-many associations are mapped to their own tables
- Methods are not mapped

Define Access Control Policies



6. Global Resource Handling

- Discusses access control
- Describes access rights for different classes of actors
- Describes how object guard against unauthorized access.

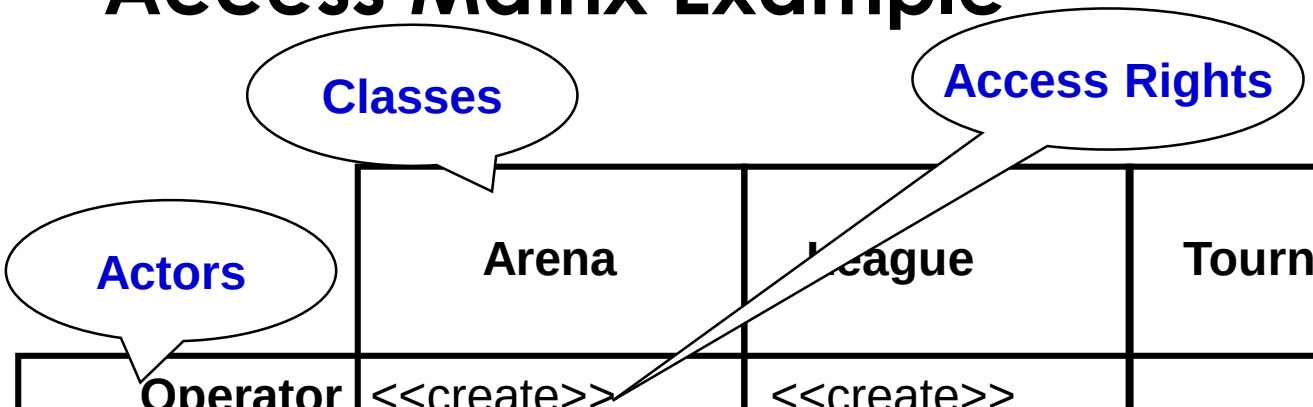
Defining Access Control

- In multi-user systems different actors usually have different access rights to different functionality and data
- How do we model these accesses?
  - During analysis we model them by associating different use cases with different actors
  - During system design we model them determining which objects are shared among actors.

Access Matrix

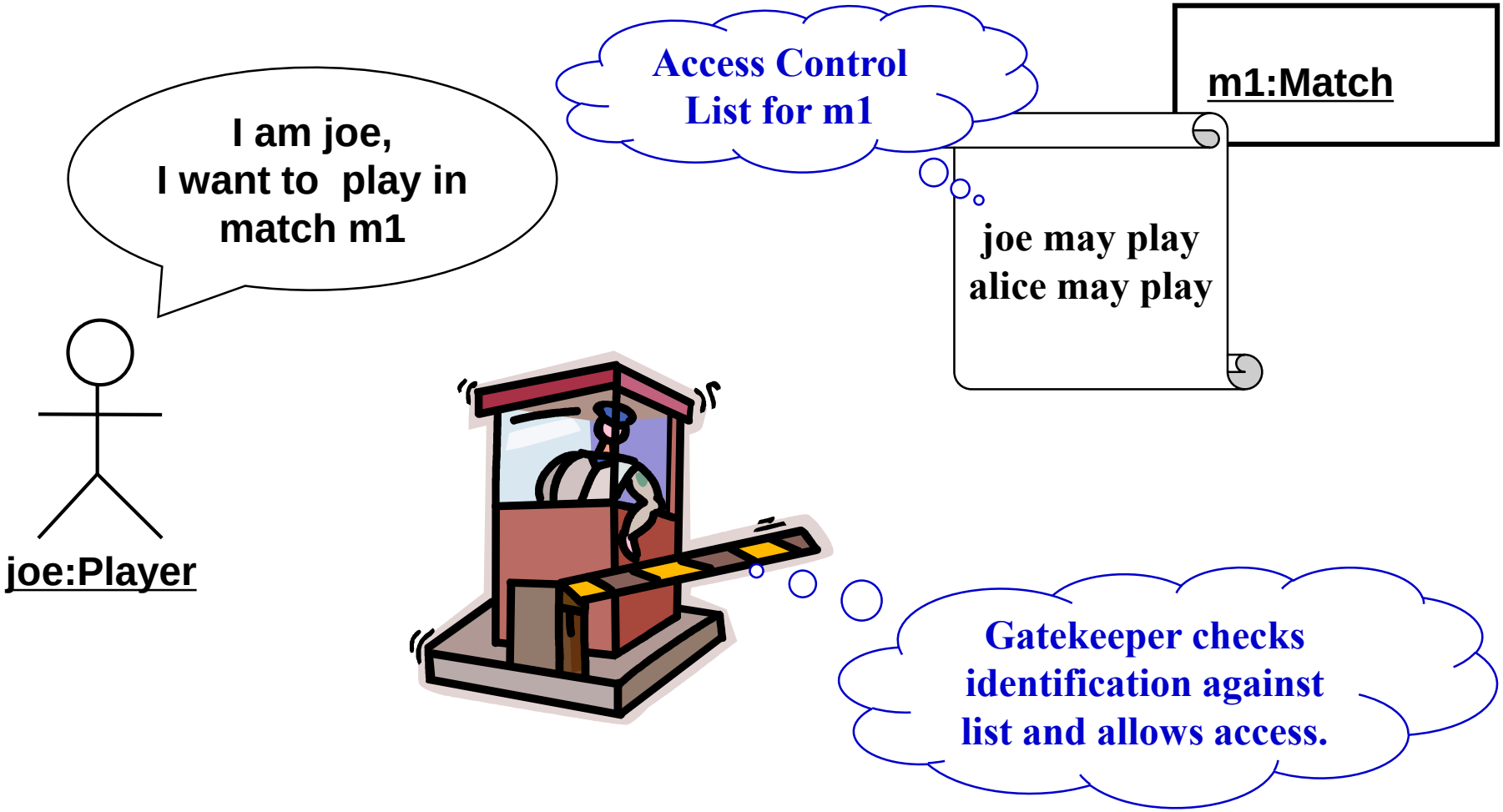
- We model access on classes with an access matrix:
  - The rows of the matrix represents the actors of the system
  - The column represent classes whose access we want to control
- Access Right: An entry in the access matrix. It lists the operations that can be executed on instances of the class by the actor.

Access Matrix Example

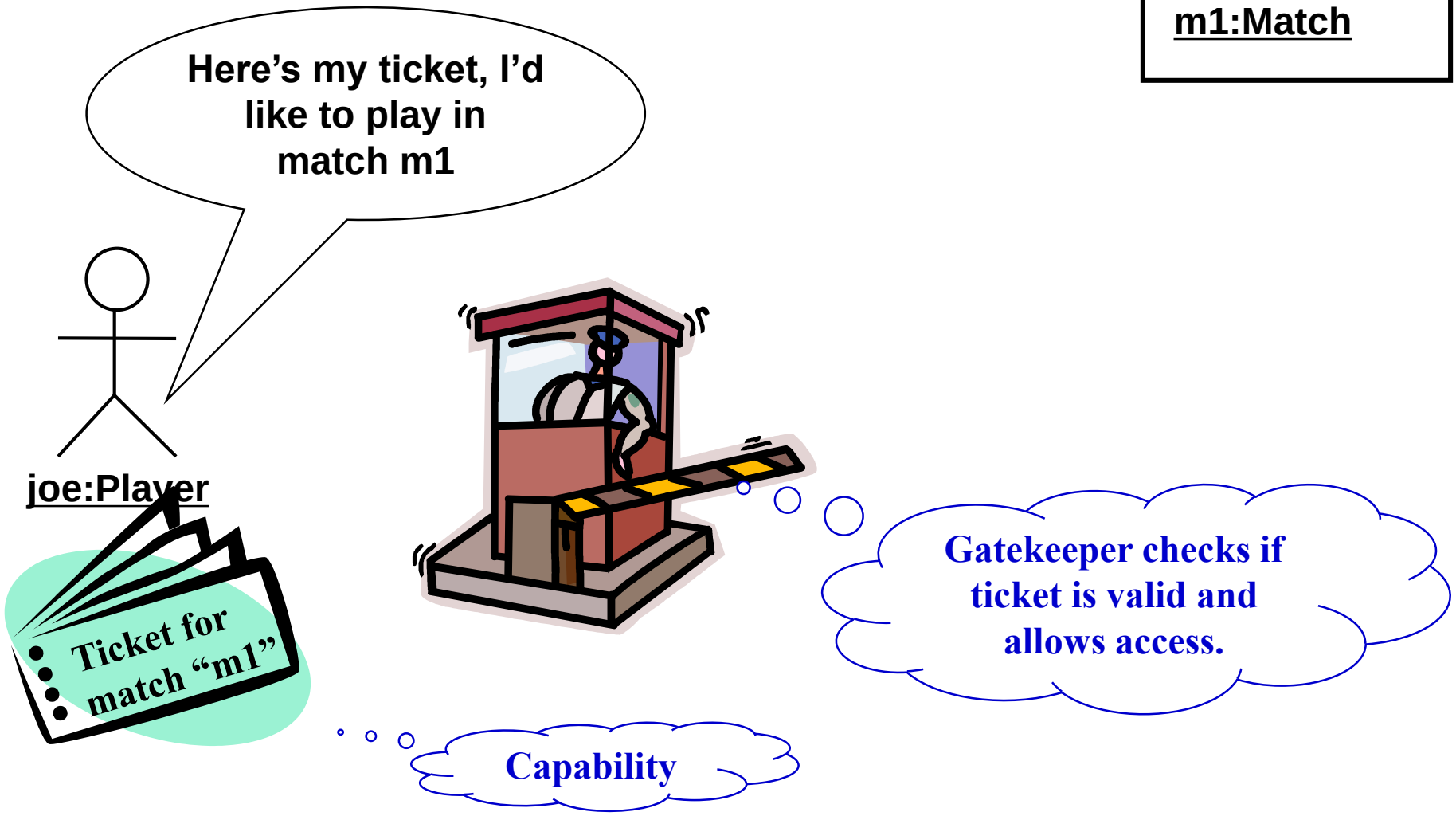


	Arena	League	Tournament	Match
Operator	<<create>> createUser() view ()	<<create>> archive()		
LeagueOwner	view ()	edit ()	<<create>> archive() schedule() view()	<<create>> end()
Player	view() applyForOwner()	view() subscribe()	applyFor() view()	play() forfeit()
Spectator	view() applyForPlayer()	view() subscribe()	view()	view() replay()

Access Control List Realization



Capability Realization



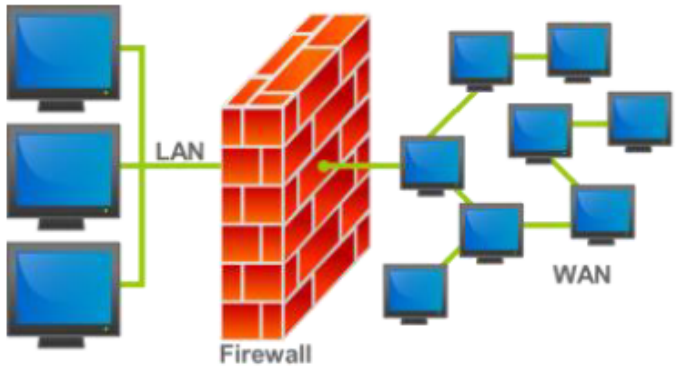
Rules Example

Often the number of actors and the number of protected objects are too large for either the capability or the access control list representations.

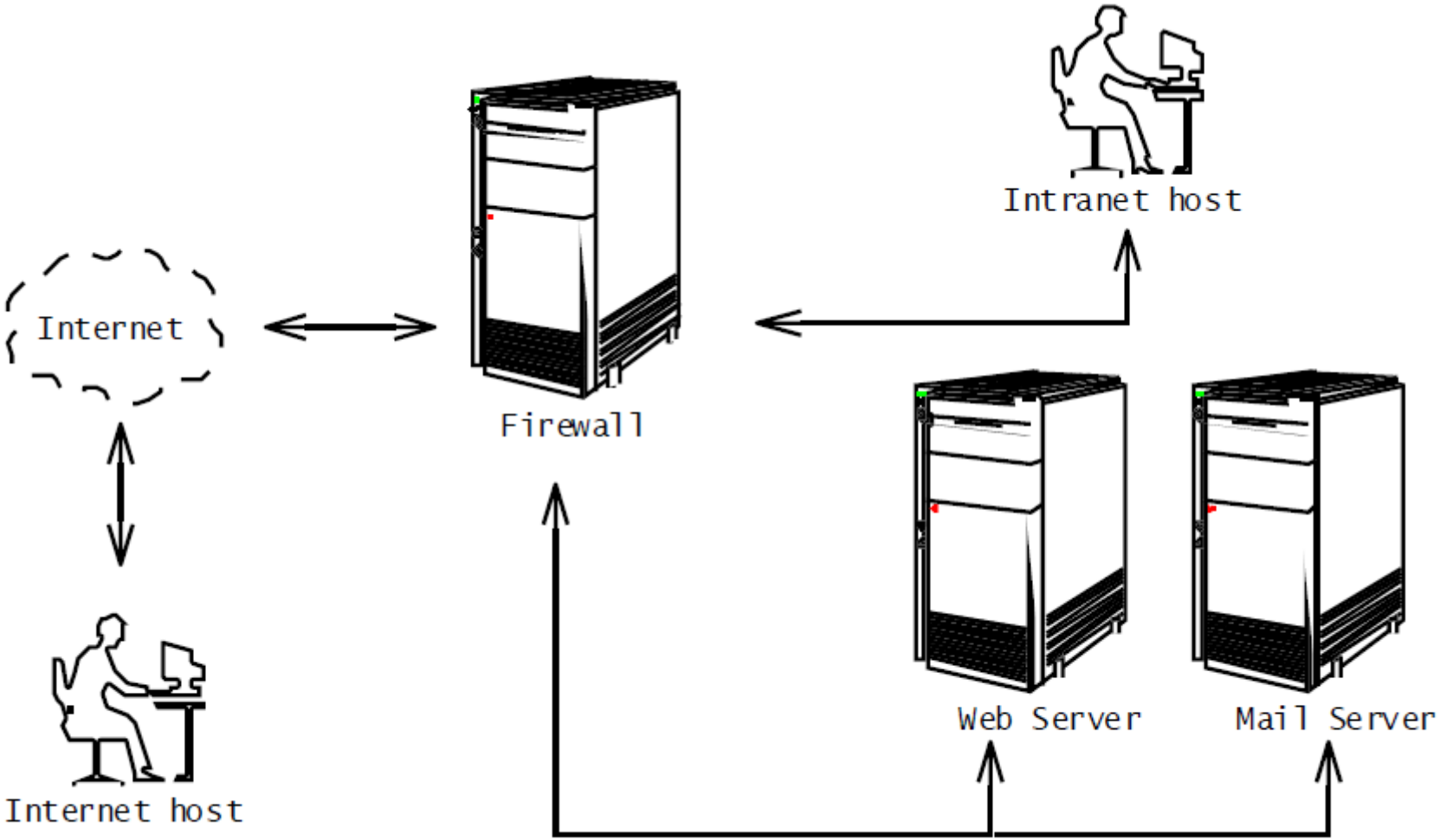
In such cases rules can be used as a compact representation of the global access matrix

A **firewall** is a part of a computer system or network that is designed to block unauthorized access while permitting authorized communications.

It is a device or set of devices configured to permit, deny, encrypt, decrypt, or proxy all (in and out) computer traffic between different security domains based upon a set of rules and other criteria.



Firewall Example



Firewall Example

Firewall access is defined in terms of a list of rules

The action of the matching rule dictates whether the current action chould be filtered or not

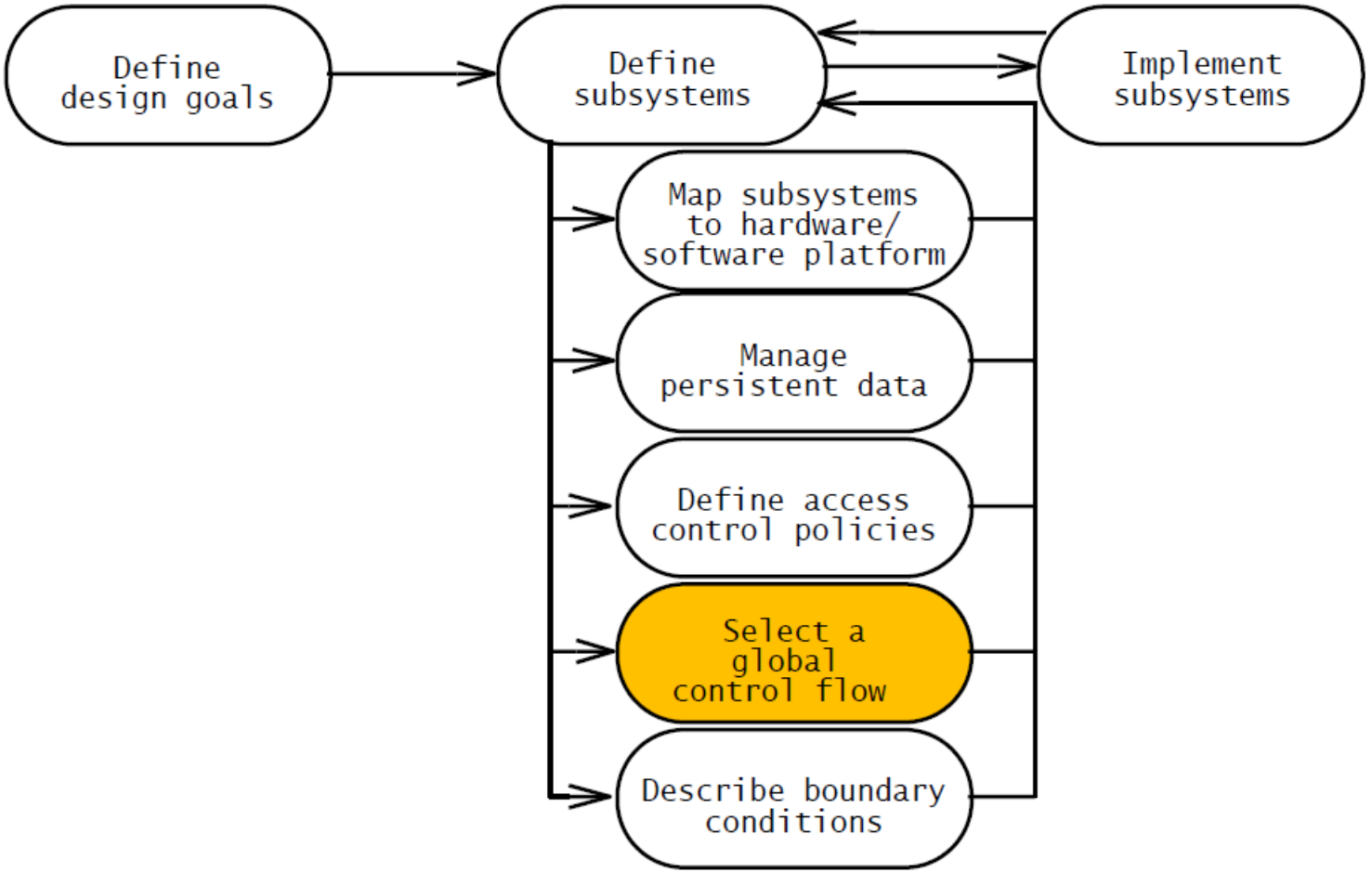
Static access control vs. Dynamic access control

Table 7-3 Simplified example of packet filtering rules for firewall of Figure 7-8.

Source Host	Destination Host	Destination Port	Action
any[a]	Web Server	http	allow
any	Mail Server	smtp	allow
Intranet host	Web Server	rsync	allow
Intranet host	Mail Server	pop	allow
Internet host	Web Server	rsync	deny
Internet host	Mail Server	pop	deny
Internet host	Intranet host	any	deny
any	any	any	deny

a. any means any one of Intranet host, Internet host, Web Server, or Mail Server.

Control Flow



Control Flow

Control Flow is the **sequencing of actions in the system**

In OO systems, sequencing actions includes deciding **which operations** should be executed and in **which order**

These decisions are based on **external events** generated by an actor or on **the passage of time**

Control flow is **a design issue**

- During Analysis control flow is not really an issue since we assume that all objects are running simultaneously, executing operations any time they need to execute them
- During system design we need to take into account that not every object can run on its own processor

Control Flow

There are two basic control flow mechanisms:

Procedure-driven control

- Control resides within program code.
- Example: Main program calling procedures of subsystems.

Event-driven control

- Main loop waits for an external event
- Whenever an event becomes available it will be dispatched to the appropriate object, based on information associated with the event
- Control resides within a dispatcher calling functions via callbacks.

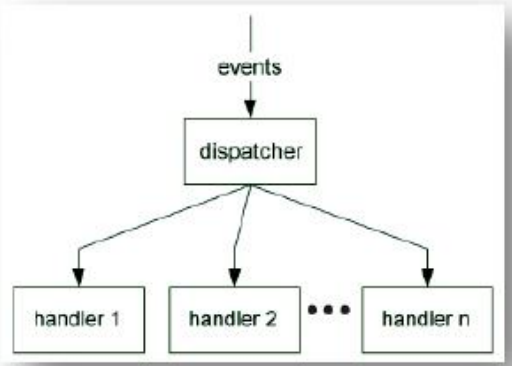
Control Flow

Procedure-driven

```
    Stream in, out;
    String userid, passwd;
/* Initialization omitted */
    out.println("Login:");
    in.readln(userid);
    out.println("Password:");
    in.readln(passwd);
    if (!security.check(userid, passwd)) {
       out.println("Login failed.");
       system.exit(-1);
    }
/* ...*/
```

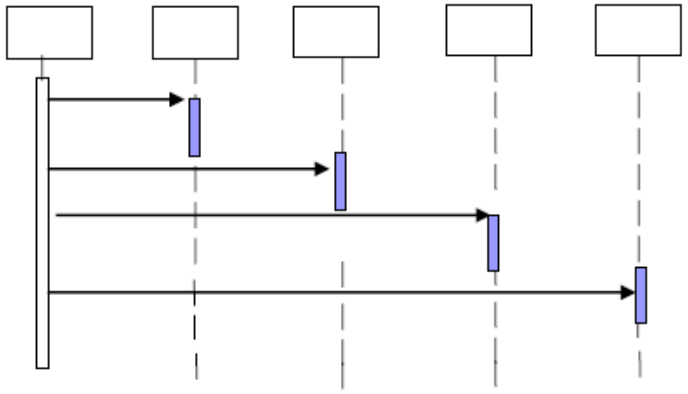
Event-driven

```
do forever:    # the event loop

    get an event from the input stream

    if   event type == EndOfEventStream :
         quit # break out of event loop

    if   event type == ... :
         call the appropriate handler subroutine,
         passing it event information as an argument

    elif event type == ... :
         call the appropriate handler subroutine,
         passing it event information as an argument

    else:   # handle an unrecognized type of event
         ignore the event, or raise an exception
```



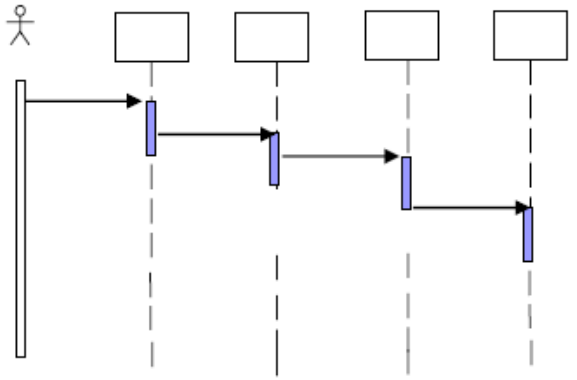
Centralized vs Decentralized

Centralized Control: Much of the dynamic behavior is **placed in a single object**, usually the control object. It knows all the other objects and often uses them for direct questions and commands.



Centralized Control Flow

Decentralized Control:
The dynamic behavior is **distributed**. Each object delegates some responsibility to other objects. Each object knows only a few of the other objects and knows which objects can hel with a specific behavior.



Decentralized Control Flow

Centralized vs. Decentralized Designs

- Centralized Design
  - One control object or subsystem ("spider") controls everything
    - Pro: Change in the control structure is very easy
    - Con: The single control object is a possible performance bottleneck
- Decentralized Design
  - Not a single object is in control, control is distributed; That means, there is more than one control object
    - Con: The responsibility is spread out
    - Pro: Fits nicely into object-oriented development

Centralized vs. Decentralized Designs (2)

- Should you use a centralized or decentralized design?
- Take the sequence diagrams and control objects from the analysis model
- Check the participation of the control objects in the sequence diagrams
  - If the sequence diagram looks like a fork => Centralized design
  - If the sequence diagram looks like a stair => Decentralized design.

8. Boundary Conditions

Most of the system design effort is concerned with steady-state behavior. However the system design must also address the initiation and finalization of the system. This is addressed by a set of new use cases called administration use cases

- Initialization
  - Describe how the system is brought from a non-initialized state to steady-state
- Termination
  - Describe what resources are cleaned up and other systems are notified upon termination
- Failure
  - Possible failures: Bugs, errors, external problems
  - Good system design foresees fatal failures and provides mechanisms to deal with them.

Boundary Condition Questions

- Initialization
  - What data need to be accessed at startup time?
  - What services have to registered?
  - What does the user interface do at start up time?
- Termination
  - Are single subsystems allowed to terminate?
  - Are subsystems notified if a single subsystem terminates?
  - How are updates communicated to the database?
- Failure
  - How does the system behave when a node or communication link fails?
  - How does the system recover from failure?

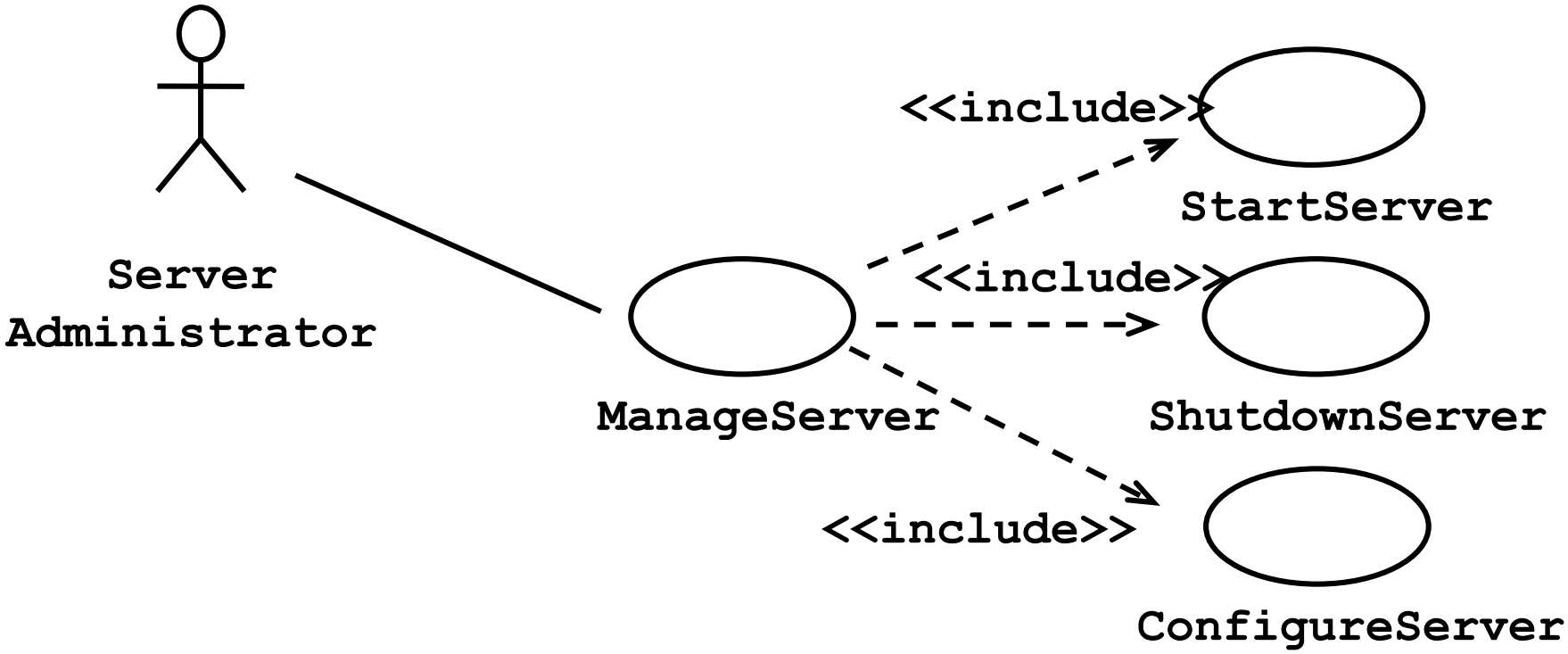
Modeling Boundary Conditions

- Boundary conditions are best modeled as use cases with actors and objects
- We call them boundary use cases or administrative use cases
- Actor: often the system administrator
- Interesting use cases:
  - Start up of a subsystem
  - Start up of the full system
  - Termination of a subsystem
  - Error in a subsystem or component, failure of a subsystem or component.

Example: Boundary Use Case for ARENA

- Let us assume, we identified the subsystem `AdvertisementServer` during system design
- This server takes a big load during the holiday season
- During hardware software mapping we decide to dedicate a special node for this server
- For this node we define a new boundary use case `ManageServer`
- `ManageServer` includes all the functions necessary to start up and shutdown the `AdvertisementServer`.

ManageServer Boundary Use Case



Summary

- System design activities:
  - Hardware/Software mapping
  - Persistent data management
  - Global resource handling
  - Software control selection
  - Boundary conditions
- Each of these activities may affect the subsystem decomposition
- Two new UML Notations
  - UML Component Diagram: Showing compile time and runtime dependencies between subsystems
  - UML Deployment Diagram: Drawing the runtime configuration of the system.

Documenting System Design

System Design Document

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

   Glossary

Reviewing System Design

Like analysis, system design is an evolutionary and iterative activity. Unlike analysis, there is no external agent, such as the client, to review the successive iterations and ensure better quality.

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?

Reviewing System Design (Completeness)

- 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?

Reviewing System Design (Readability)

- Are subsystem names understandable?
- Do entities (e.g., subsystems, classes) with similar names denote similar concepts?
- Are all entities described at the same level of detail?