

Modelli per la documentazione delle Architetture Software

Progettazione e Sviluppo di Sistemi Software

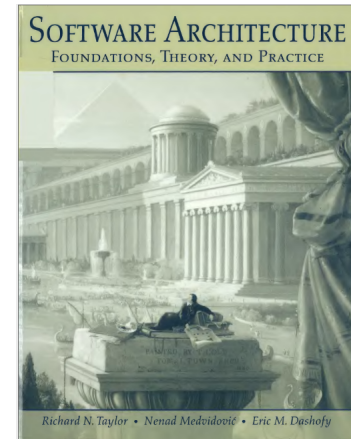
Prof. Anna Rita Fasolino

Ing. Domenico Amalfitano

Riferimenti



Documenting Software Architectures: Views and Beyond (2nd Edition)



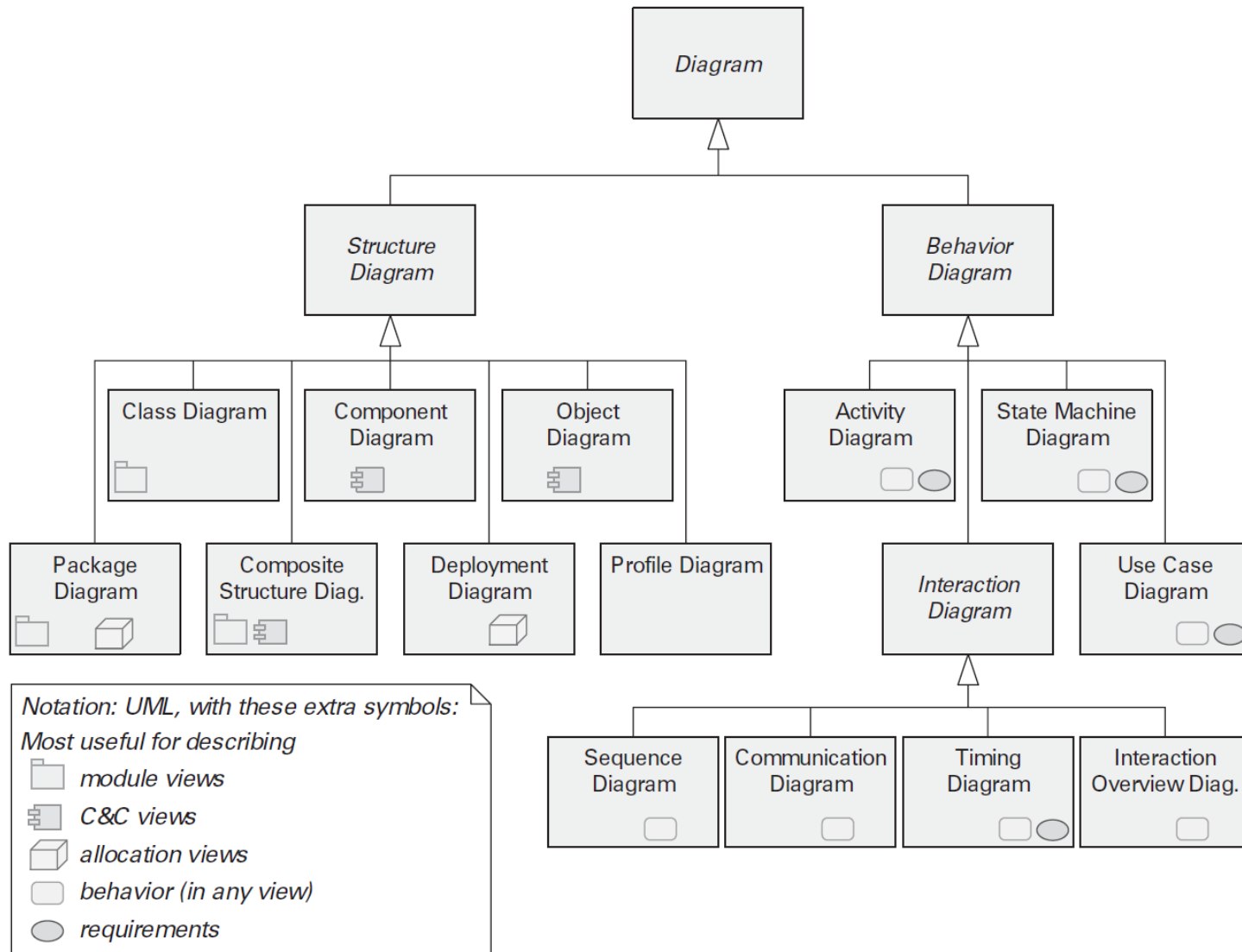
Software Architecture: Foundations, Theory, and Practice

- **Representing Explicit Attributes in UML**
 - http://dawis2.icb.uni-due.de/events/AOM_MODELS2005/Cepa.pdf
- **Documenting Component and Connector Views with UML 2.0**
 - <http://www.sei.cmu.edu/reports/04tr008.pdf>

Utilizzo di UML per la documentazione delle Views

- UML conserva molte delle caratteristiche che risalgono alle sue origini object-oriented
- Le astrazioni orientate agli oggetti non sono sempre i migliori strumenti per descrivere le architetture software.
 - Ad esempio, UML non ha notazione per i layers, diagrammi di contesto, o rich connectors.
- UML mette a disposizione altri strumenti che permettono di superare tali mancanze.

Mapping tra UML 2.2 diagram types e Views



Stereotypes (Stereotipi)

- Utilizzati per specializzare ulteriormente il significato di qualsiasi simbolo UML.
 - Elemento o relazione.
- È una etichetta (*label*) domain-specific o technology-specific.
 - Mostrata in all'interno di guillemets, "angle brackets, parentesi angolari, << >>.
- UML mette a disposizione stereotipi standard.
- Possiamo definire nuovi stereotipi.
 - ad esempio <<layer>>

Processo di documentazione

- Step 1 – Produrre documentazione di analisi
- Step 2 – Produrre documentazione di progetto/implementazione

Documentazione di analisi

- Ha come input la documentazione informale dei requisiti.
- Ha l'obiettivo di produrre una documentazione che sia del tutto indipendente dall'implementazione
- In genere contiene
 - Diagrammi dei casi d'uso (UC), Diagramma dei requisiti, Storie Utente (US), etc.
 - Sytem Domain Model (Modelli Statici)
 - Diagrammi di Contesto (Modelli Statici)
 - Prima bozza di diagrammi di architettura (Modelli Statici)
 - Per ogni requisito (UC, US) diagrammi dinamici che ne specificano il comportamento (Modelli Dinamici)
 - Activity Diagram, Sequence Diagram, Communication Diagram, State Chart Diagram

Documentazione di Progetto

- Scelta le tecnologie implementative, la documentazione di progetto ne recepisce le peculiarità
- Ha come input la documentazione di analisi
- Ne aggiunge dettagli di progettazione e di sviluppo delle diverse soluzioni implementative.
- In genere contiene
 - Diagrammi dei componenti che definiscono l'architettura generale del sistema software da sviluppare
 - Per ogni componente i diagrammi dei package e/o delle classi che ne dettagliano le scelte di design e di implementazione
 - Diagrammi dinamici che descrivono le interazioni tra le classi e/o componenti per specificare dettagli implementativi di ogni requisito, UC o US.
 - Diagrammi di Deploy

Tecniche di documentazione

- *Refinement.*
- *Descriptive completeness.*
- *Documenting context diagrams.*
- *Documenting variation points.*
- *Documenting architectural decisions.*
- *Combining views.*

REFINEMENT

Scopo del Refinement

- Presentare le informazioni in sezioni separate e più facilmente gestibili.
- Un raffinamento elabora (aggiunge informazioni) viste preesistenti.
- Permette di catturare e rappresentare informazioni con più o meno dettagli.
 - Meno dettagli *utili* nelle prime fasi della progettazione, eccellenti per le introduzioni ed overviews.

Tipi di refinements

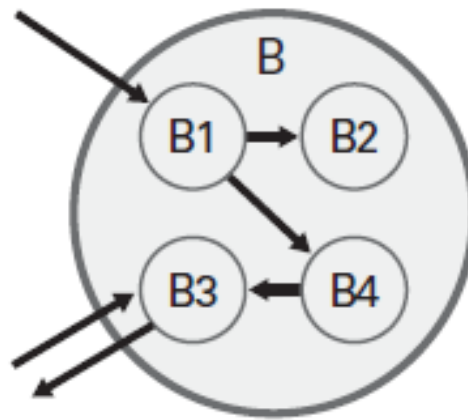
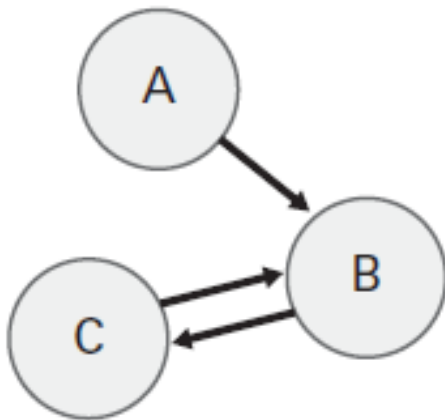
- Ci sono due importanti tipologie di refinement:

—**Decomposition refinement.**

—**Implementation refinement.**

Decomposition refinement

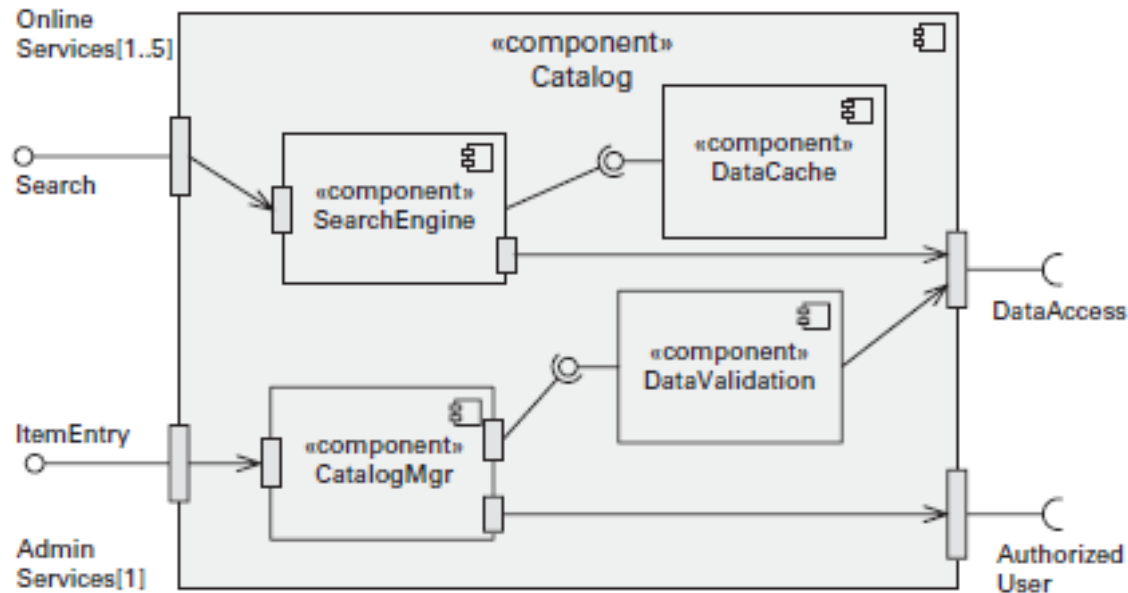
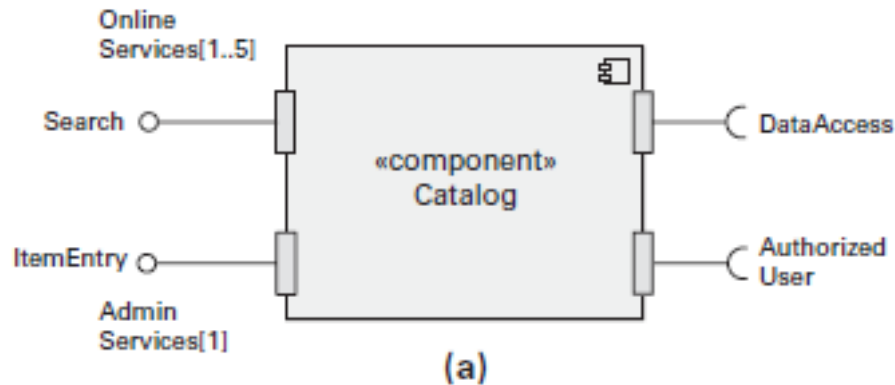
- Elabora un singolo elemento al fine di rilevarne la sua struttura interna.
- Bisogna mantenere la consistenza delle relazioni.



Poiché B ha due inputs e un output il refinement deve continuare a soddisfare questo vincolo di progetto

Refinement in UML

decomposition
refinement in UML 2.x.

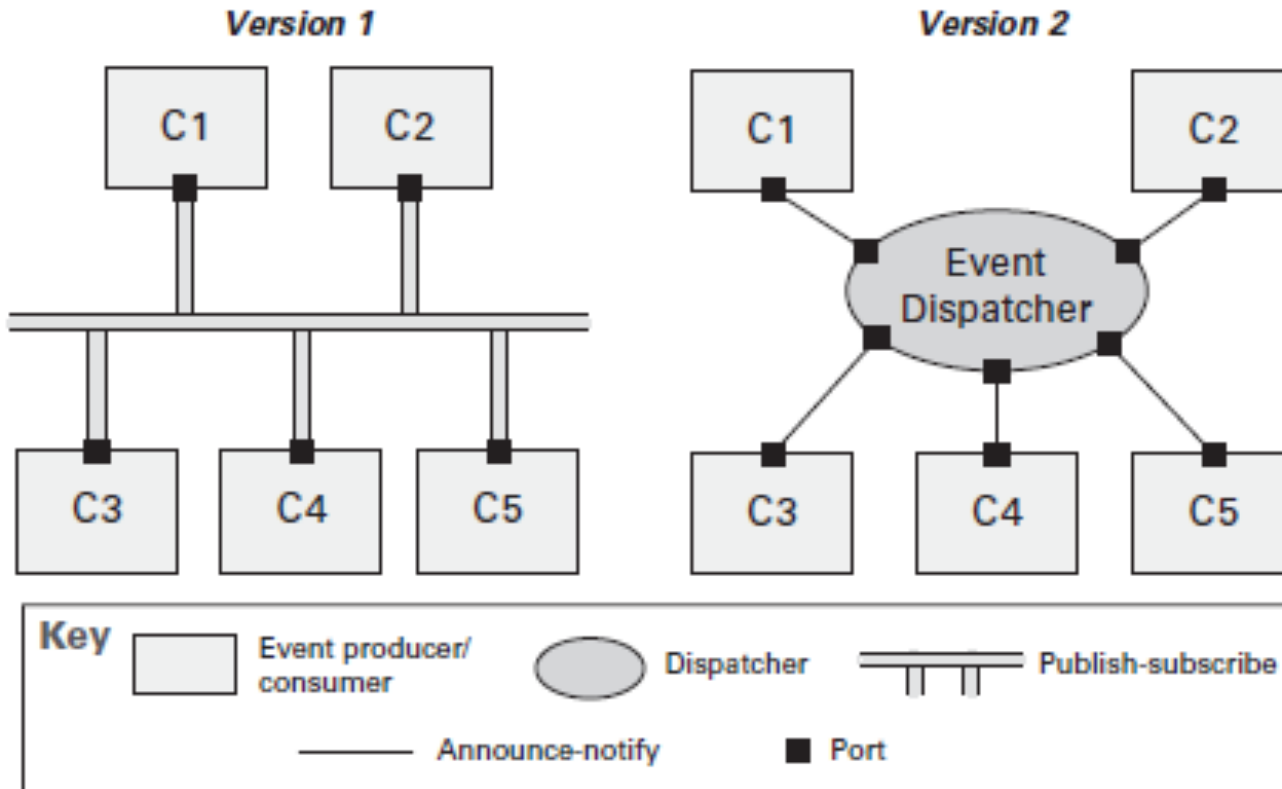


Implementation refinement

- Il livello di dettaglio resta lo stesso
- Molti o tutti gli elementi e relazioni sono sostituiti.
- Mostra delle informazioni che descrivono come gli elementi originali verranno realizzati.

Esempio

Il bus publish-subscribe è realizzato mediante un event dispatcher.



DOCUMENTING CONTEXT DIAGRAMS

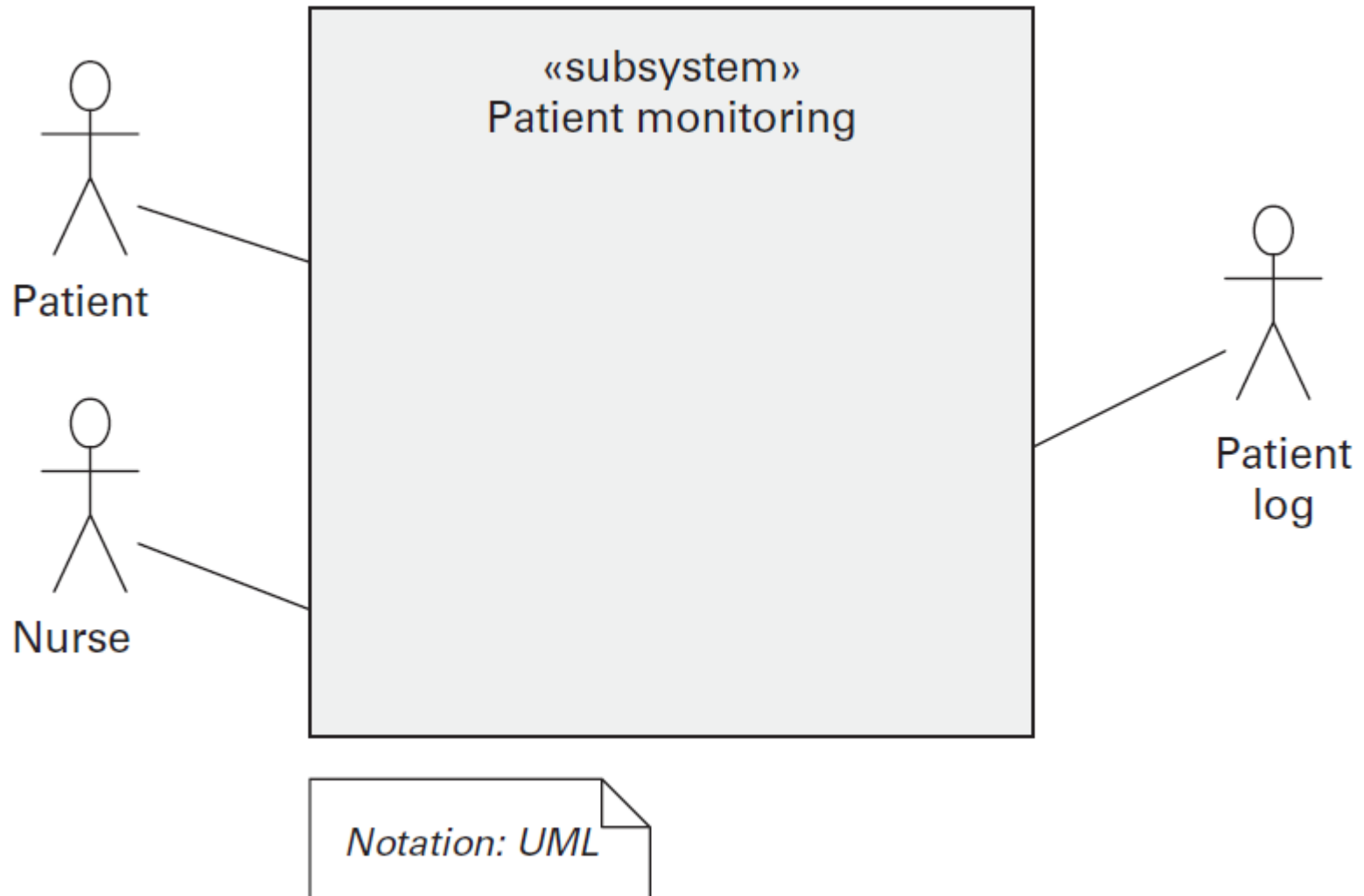
Scopo

- Mostrare il contesto di una View.
- Il *context* rappresenta l'ambiente con cui il sistema o una specifica parte del sistema interagisce.
 - Le entità del contesto possono essere umani, altri sistemi od oggetti fisici come sensori o devices.
- I context diagrams non sono sempre un **toplevel context diagrams** (TLCDs)

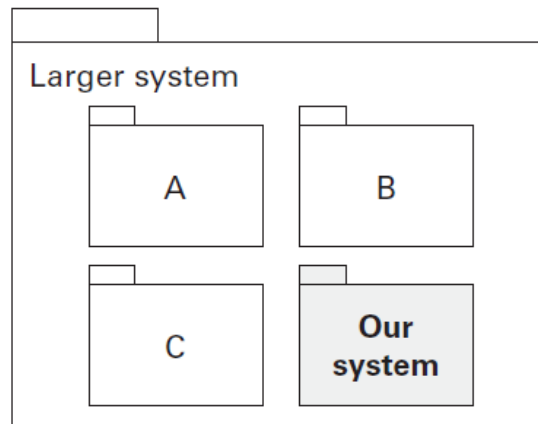
Come documentare un contex diagram

- UML non ha costrutti di base per descrivere un context diagram.
- Soluzioni:
 - Uso di Use case e Class diagrams con opportuni stereotipi.
 - Usare il vocabolario delle viste utilizzate per rappresentare il contesto del sistema.

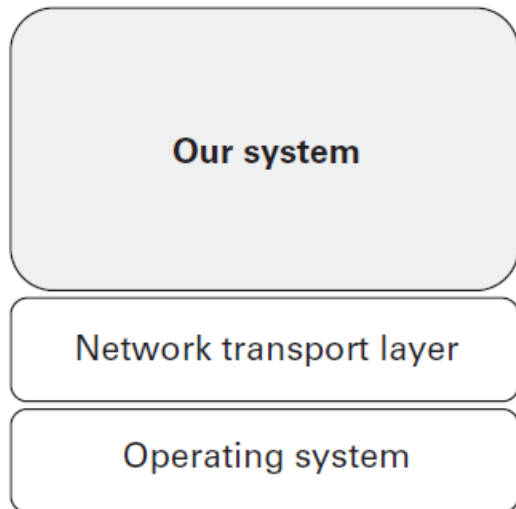
Esempio USE Case diagrams



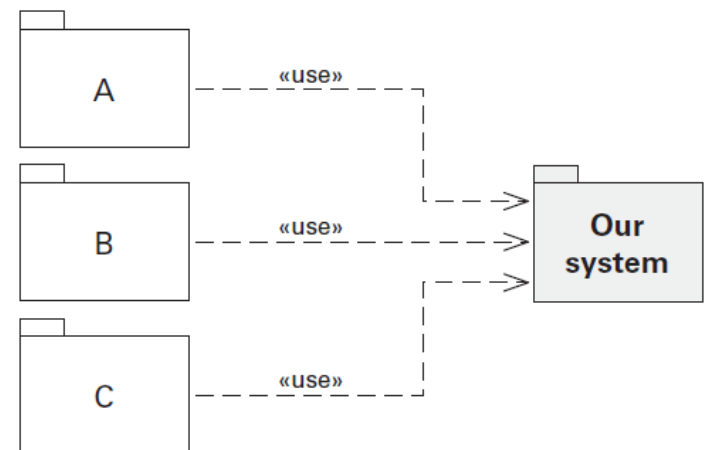
Esempi con vocabolari delle viste utilizzate – Module Styles



Decomposition Style



Layered Style



Usage Style

Diagramma di contesto con classi esterne e classi di input

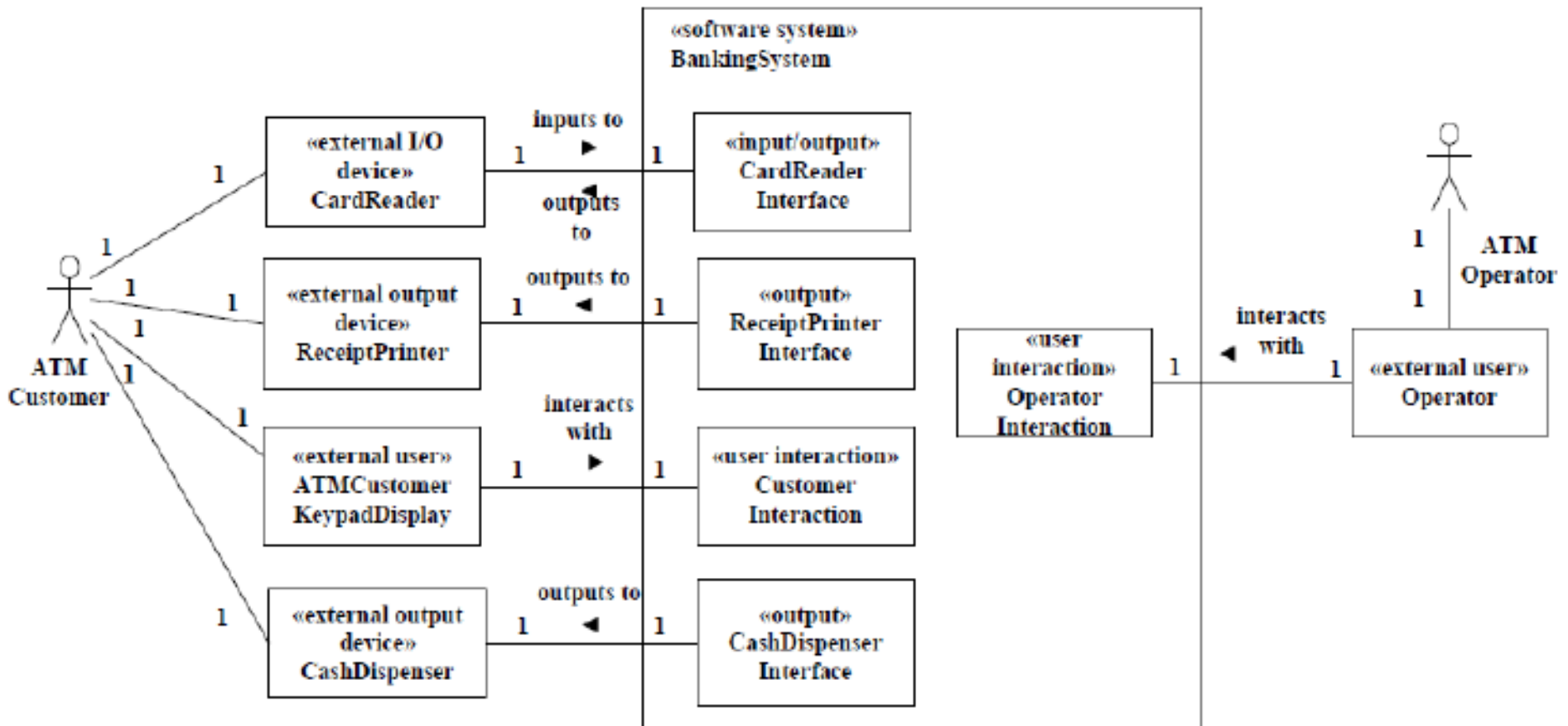
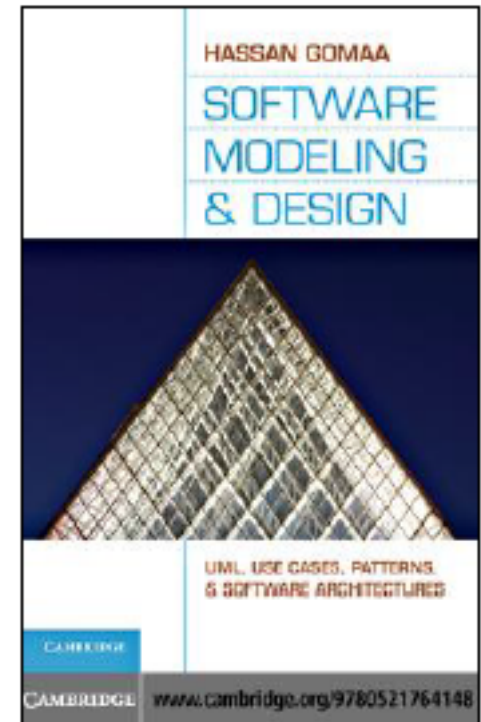
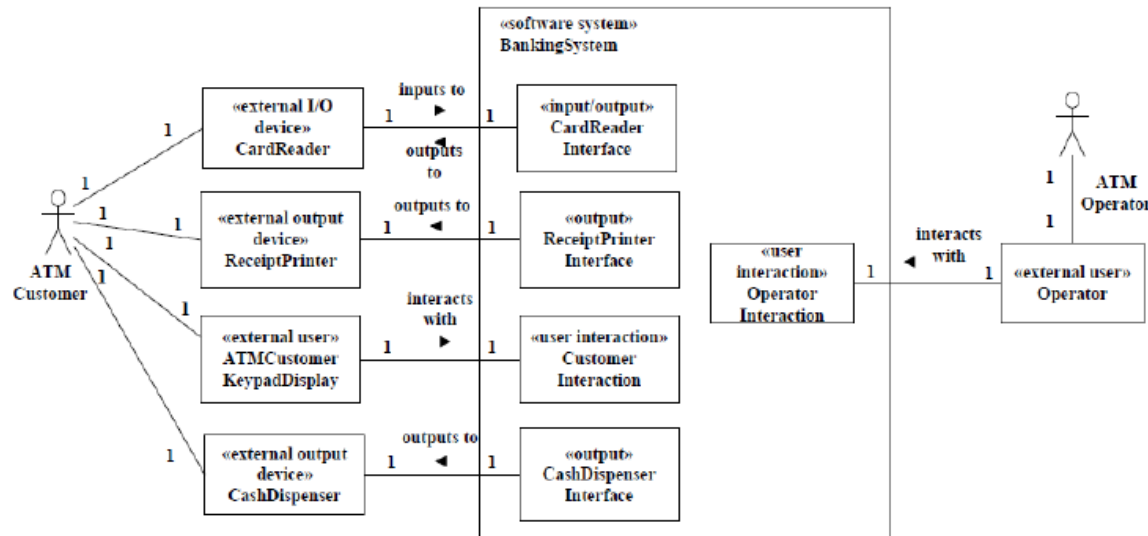


Diagramma di contesto con classi esterne e classi di input



Capitoli 5 e 6

H. Gomaa, Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures

DOCUMENTING ARCHITECTURAL DECISIONS

Scopo

- Bisogna descrivere il *rationale* per le scelte del progetto che sono ritenute rilevanti.
- Il rationale può essere descritto seguendo un template di documentazione.
- Il documento redatto seguendo tale template descriverà le decisioni architetturali.

COMBINING VIEWS

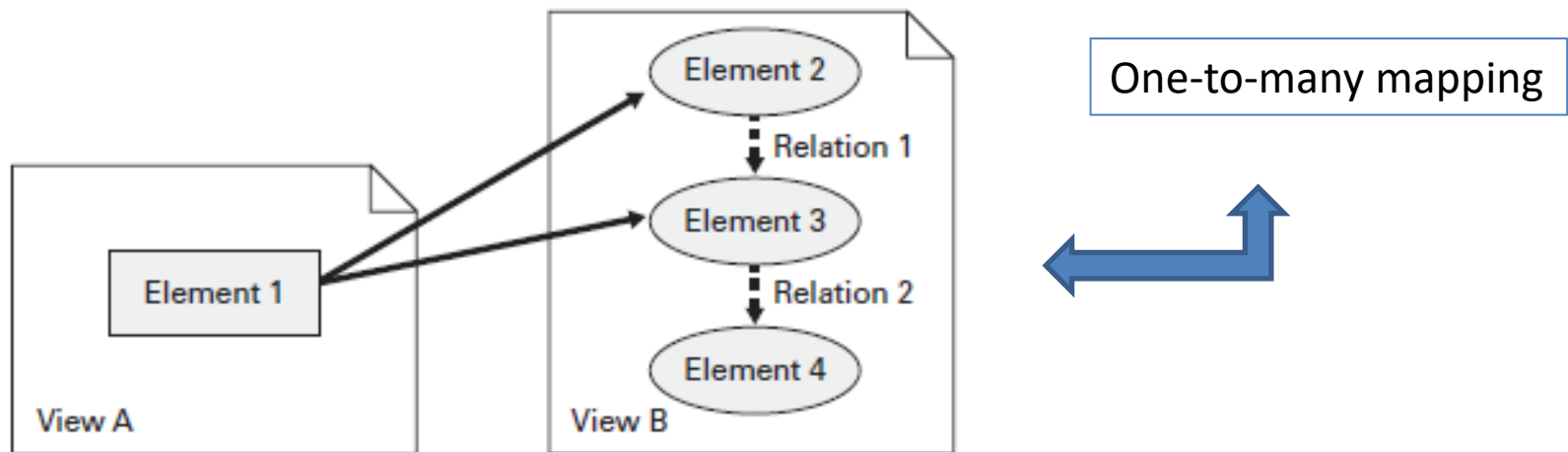
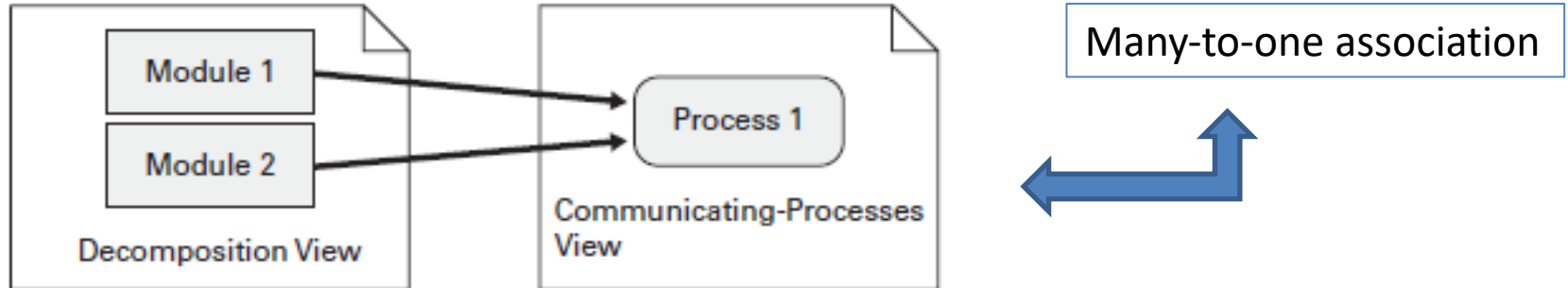
Scopo

- Poiché tutte le viste fanno parte di una stessa architettura e collaborano per raggiungere uno scopo comune → molte viste hanno forti associazioni tra loro.
- Bisogna
 - Gestire come le viste sono associate tra loro.
 - Documentare tali associazioni.

Tipi di associazioni tra Viste

- *many-to-one* – più elementi di una view sono associati ad un unico elemento di un'altra view.
- *one-to-many* – un singolo elemento di una vista è associato a più elementi di un'altra vista.
- *many-to-many* – un insieme di elementi di una vista sono associati a un insieme di elementi di un'altra vista.

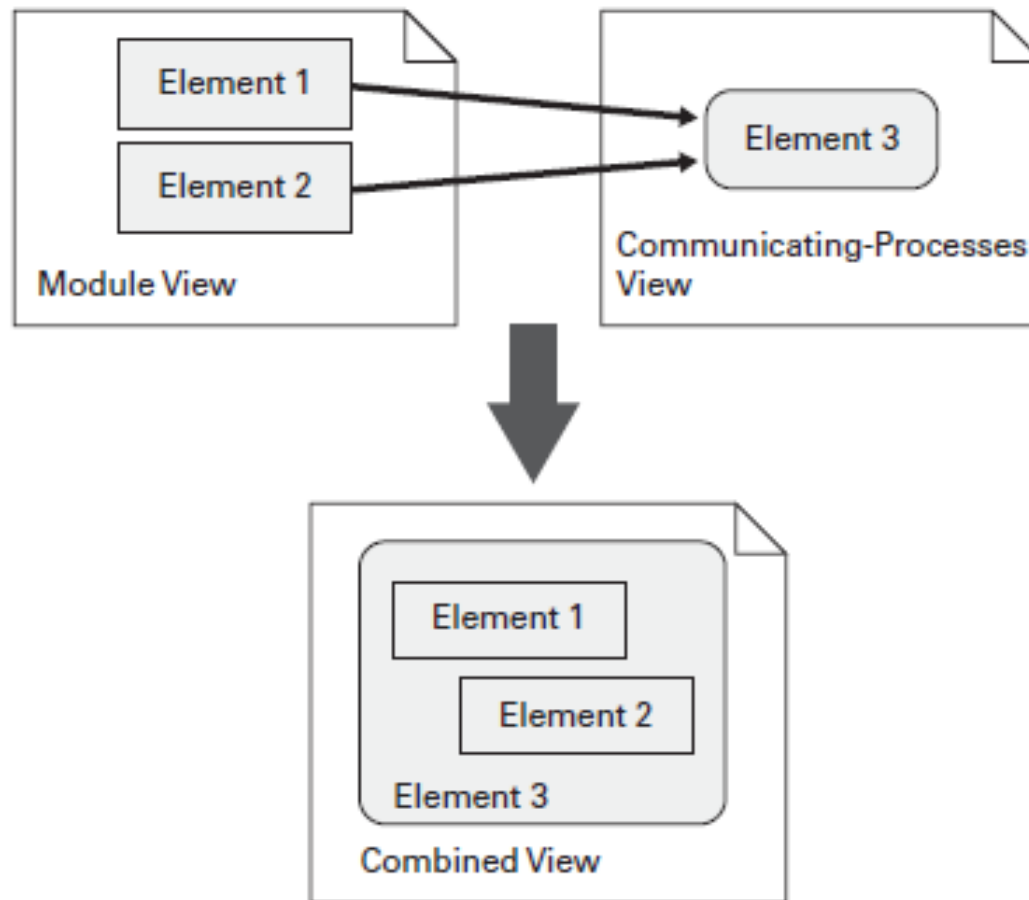
Esempi di associazioni



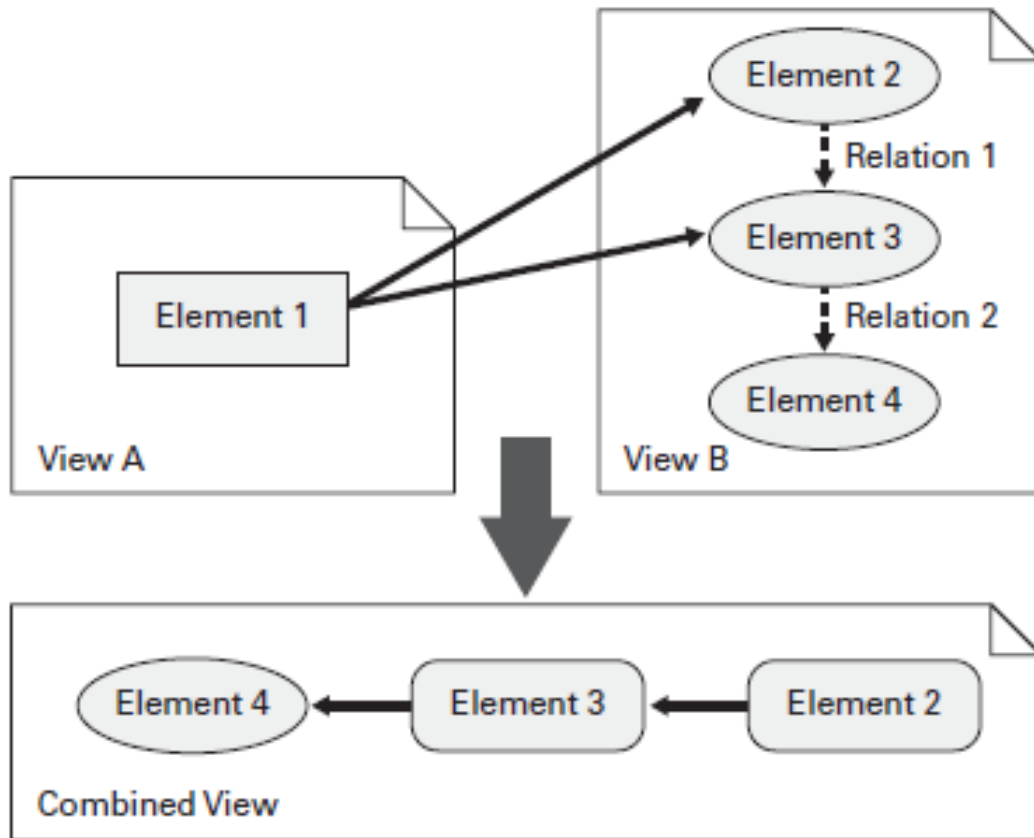
Combined Views

- Una combined view collassa più views in un'unica vista.
- Per rappresentare una associazione forte tra due o più viste.
- Serve a ridurre il numero di viste presenti in un documento di architettura.

Combined Views – Esempio 1



Combined Views – Esempio 2



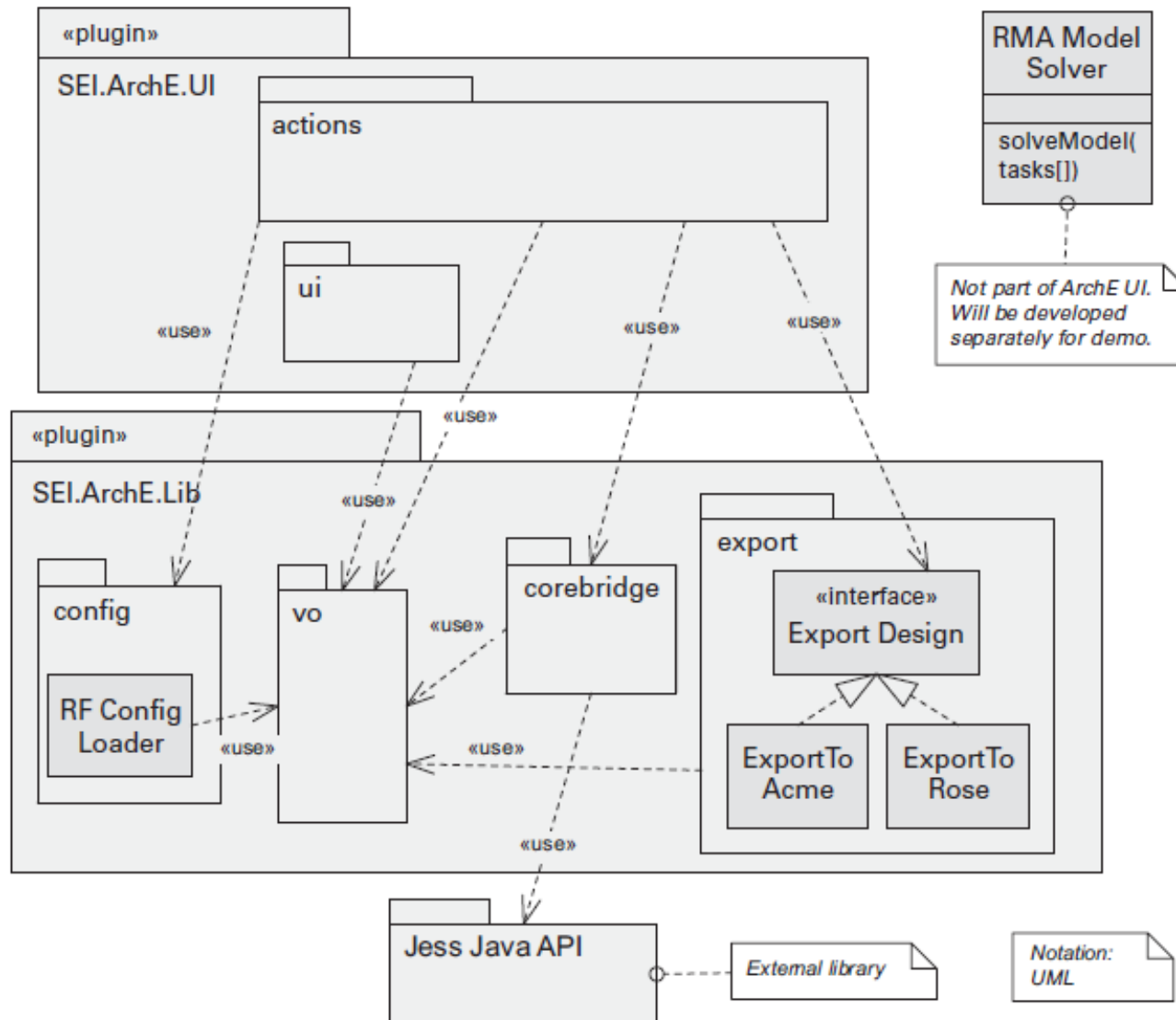
Esempio:

- Gli elementi 2, 3, e 4, sono componenti di una vista C & C
- Gli elementi 2 e 3 sono componenti che rispettivamente leggono e scrivono dati.
- L'elemento 1 è una classe di una vista di decomposizione che implementa queste due funzionalità.
- La combined view dell'elemento 1 sugli elementi 2 e 3 rende tali elementi "componenti persistenti."

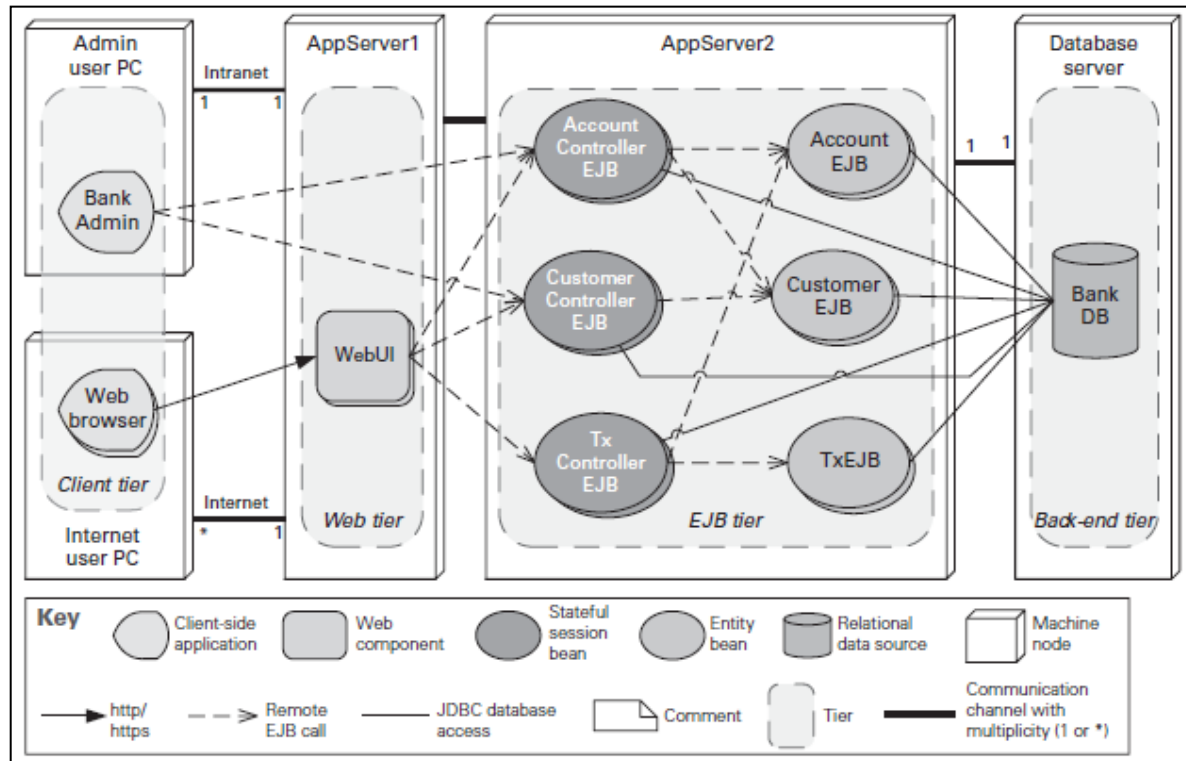
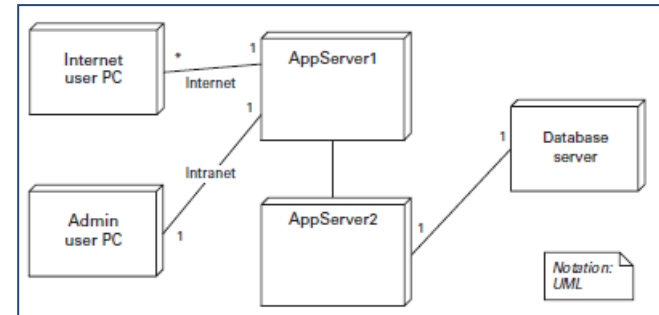
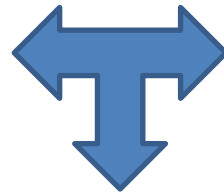
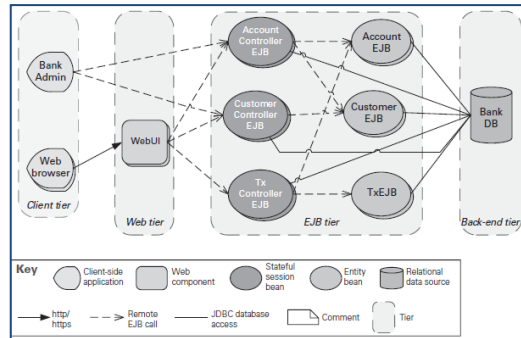
Come produrre una combined view

- Ci sono due modi per produrre una combined view.
 - Creare un overlay
 - Combinare le informazioni rappresentate in più viste separate.
 - Funziona bene se l'accoppiamento tra le viste è stretto.
 - Creare un hybrid style
 - Combinare due stili esistenti.
 - Creare una nuova style guide.
 - Da valutare se utilizzarlo

Esempio Combined View - Decomposition, Uses, and Generalization



Esempio Combined View - Tiered Client-Server and Deployment



DOCUMENTING SOFTWARE INTERFACES

Documentare le Interfacce

- Si documenta allo stesso modo sia le module interfaces che le component interfaces.
- Progettare una interfaccia significa decidere (e documentare con un interface document) quali servizi e proprietà sono visibili all'esterno e quali no.

Documentare le Interfacce

- Tutto ciò che è visibile all'esterno diventa un contratto.
- Una promessa per gli utenti che l'elemento è capace di adempiere ai propri obblighi.

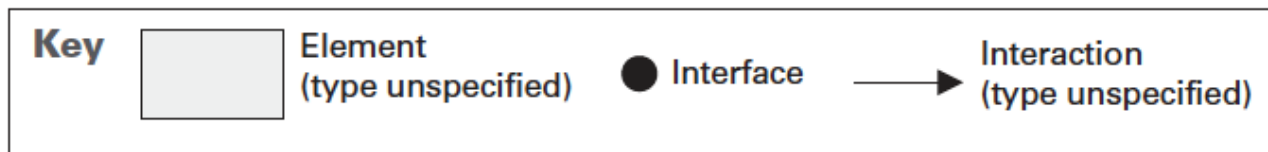
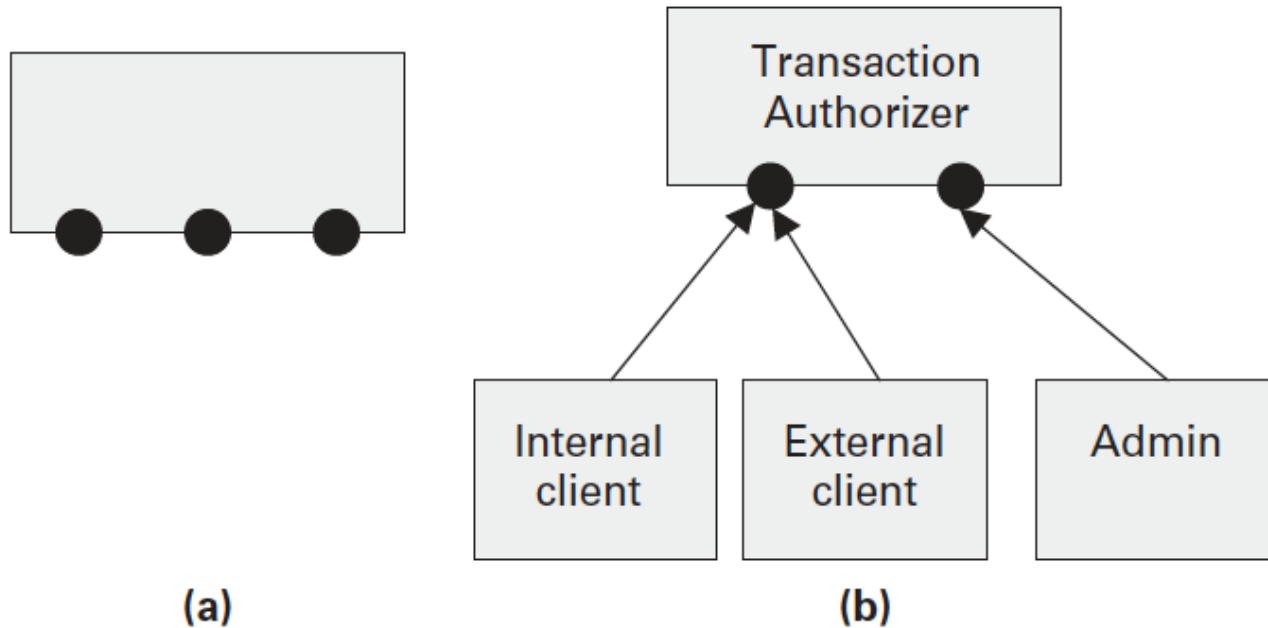
Documentare le Interfacce

- Un elemento è utilizzato da **attori (*actors*)**.
- Attori interagiscono con un elemento tramite le sue interfacce.
 - Attori possono essere sia interni che esterni al sistema documentato.
 - Le interazioni possono assumere diverse forme:
 - function o method calls, Web service requests, remote procedure calls, data streams, shared memory e message passing.
 - Molte interazioni implicano il trasferimento di controllo e/o di dati.
- I punti di interazione con un elemento sono definiti **risorse (resources)**.
 - Un interfaccia consiste di una o più risorse che possono essere consumate dagli attori.
 - Se l'elemento che fornisce una interfaccia è una classe → in genere le risorse sono chiamate metodi.

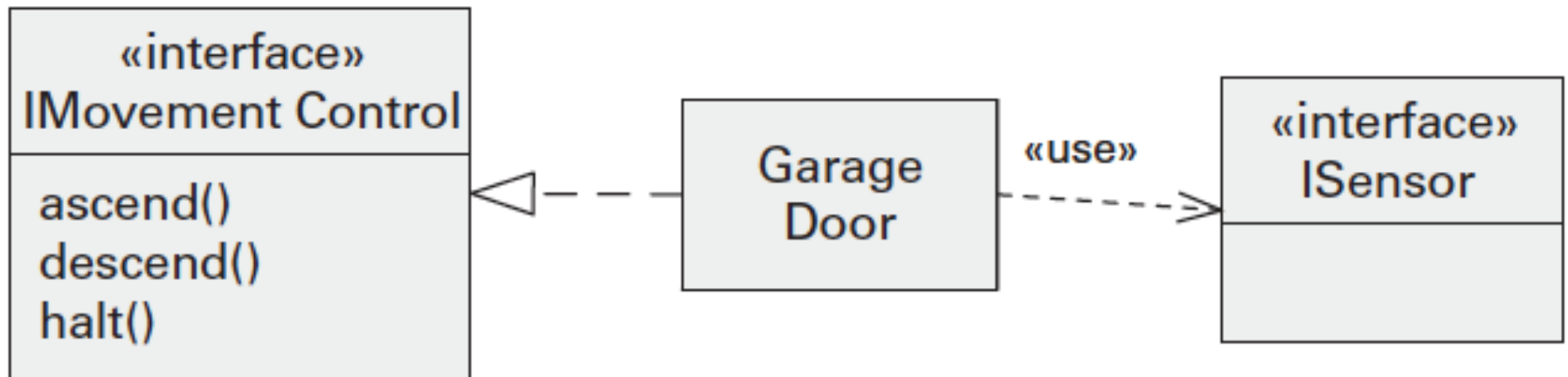
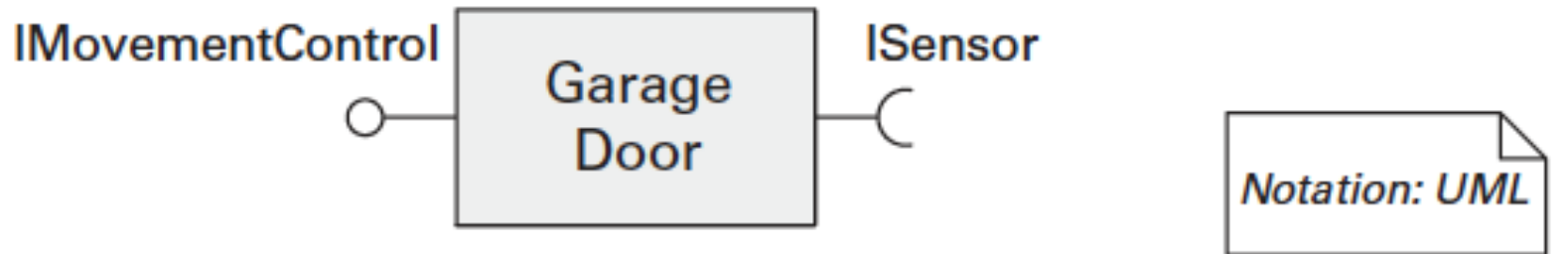
Documentare le Interfacce

- Impossibile praticamente documentare in un'unica rappresentazione ogni possibile caratteristica delle interfacce.
- Mostrare, piuttosto, solo quello che gli utilizzatori dell'interfaccia devono sapere per poter interagire con essa.
- Questa documentazione descrive quello che gli sviluppatori/tester hanno bisogno di sapere su una interfaccia in modo da poterla utilizzare in combinazione con altri elementi.

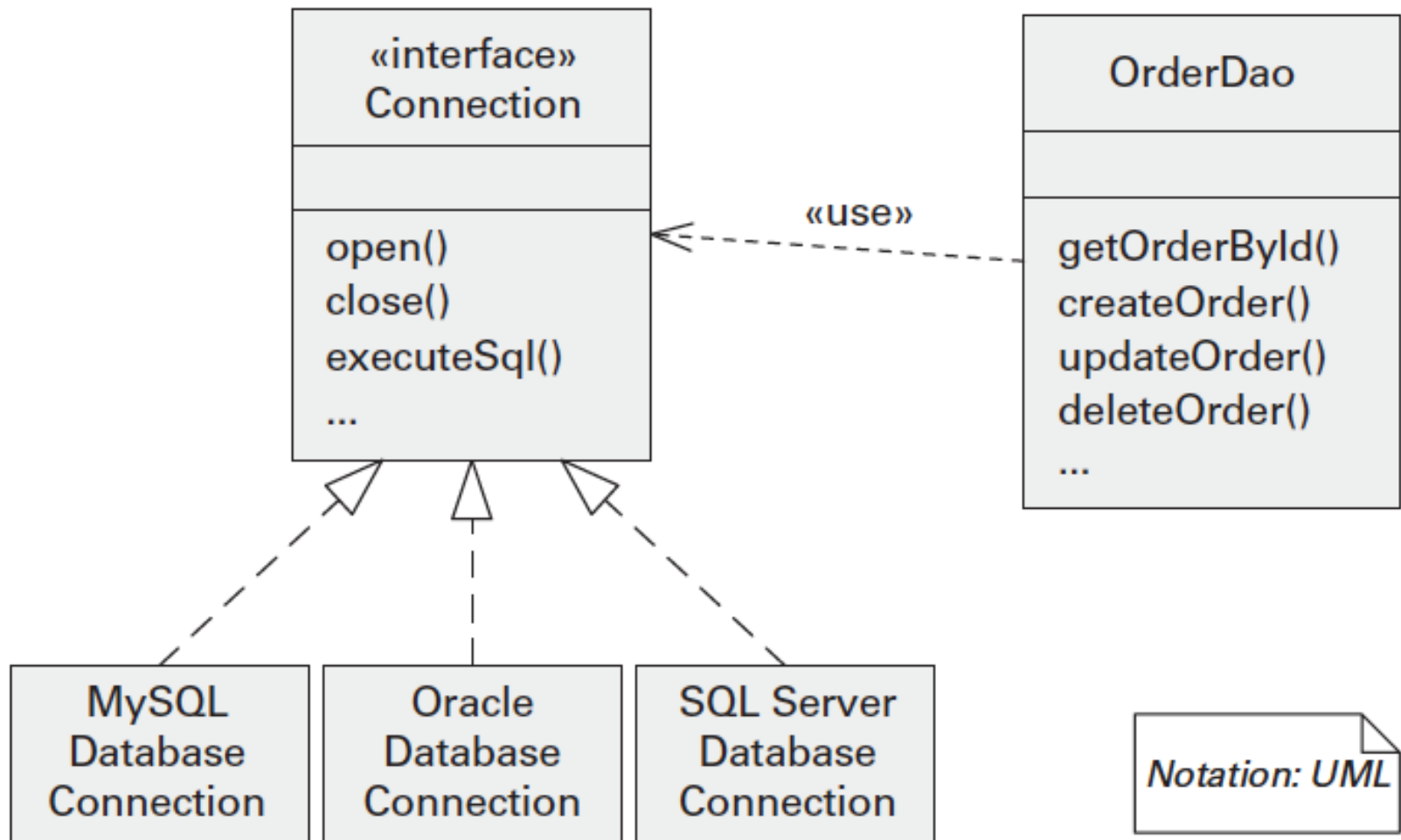
Diagrammi di Interfacce



Diagrammi di Interfacce



Diagrammi di Interfacce



Standard di Documentazione

- Un importante principio per produrre una buona documentazione prescrive l'adozione di uno standard.
- Lo standard può essere tradotto (se non lo prescrive) in uno standard organization (detto anche template)
- Presentiamo un template per la descrizione delle interfacce.
 - Ogni template è composto da sezioni (sections)

Template for interface documentation

Interface Documentation

Section 1. Interface Identity

Section 2. Resources

For each resource:

- Syntax
- Semantics
- Error Handling

Section 3. Data Types and Constants

Section 4. Error Handling

Section 5. Variability

Section 6. Quality-Attribute Characteristics

Section 7. Rationale and Design Issues

Section 8. Usage Guide

DOCUMENTING BEHAVIOR

Perché documentare il behavior

- Le relazioni strutturali rappresentano tutte le interazioni potenziali.
 - Poche di queste saranno attive ad ogni istante durante l'esecuzione.
- Bisogna descrivere come gli elementi architetture interagiscono mediante le loro strutture.

Perché documentare il behavior

- Documentare il comportamento aggiunge informazioni che fanno emergere aspetti quali:
 - L'ordine delle interazioni tra gli elementi.
 - Possibili concorrenze.
 - Dipendenze dal tempo;
 - a specifici istanti, o
 - dopo un certo periodo
 - Possibili stati del sistema o delle sue parti.
 - Usage patterns per differenti risorse del sistema (memory, CPU, database connections, network)
 - Etc.

Come documentare il comportamento

- Quando si modellano le viste comportamentali di un sistema si deve:
 1. decidere a quali domande la documentazione deve rispondere;
 2. determinare quali tipi di informazioni sono disponibili o possono essere vincolate,
 3. scegliere una notazione.

Tipi di comunicazione

- Un lettore della documentazione, osservando una relazione che lega due elementi in un diagramma strutturale potrebbe chiedersi:
 - “Cosa significa questa linea di connessione?”
 - “La linea rappresenta un flusso di dati o un flusso di controllo?”
- Un diagramma di behaviour dovrebbe poter rispondere a queste domande.

Esempi di tipi di comunicazione

Prima caratteristica:
General purpose of the
communication.

Terza caratteristica: indica se la
call è locale (cioè interna allo
stesso container o macchina)
oppure remota.

Seconda caratteristica:
come comunicano tra
loro gli elementi.

	Synchronous	Asynchronous
Data		<i>Local</i> : Shared memory <i>Remote</i> : Database
Stimulation	<i>Local</i> : Procedure call, semaphore <i>Remote</i> : RPC without parameters	<i>Local</i> : Interrupt, signal, or event without parameters <i>Remote</i> : Signal or event without parameters
Both	<i>Local</i> : Procedure call <i>Remote</i> : RPC with parameters	<i>Local</i> : Message, event with parameters <i>Remote</i> : Message, event with parameters

Notazioni per descrivere il comportamento

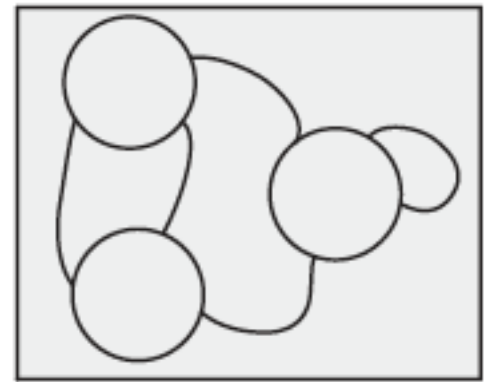
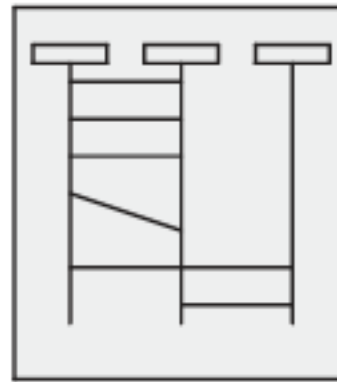
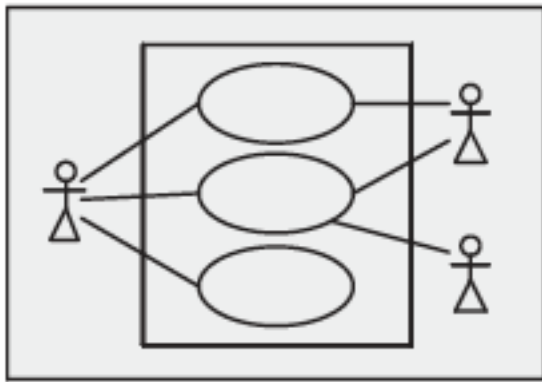
- I linguaggi che supportano la documentazione del comportamento devono includere costrutti per descrivere sequenze di interazioni.
- Devono mostrare time-based dependencies
 - Mostrando l'ordine di sequenze di interazioni e triggering activities.
- Nelle documentazioni di behavior si trovano concetti quali:
 - Stimulus and activity
 - Ordering of interactions
 - Structural elements with some relations the behavior maps to

Tipi di documentazione per il behavior

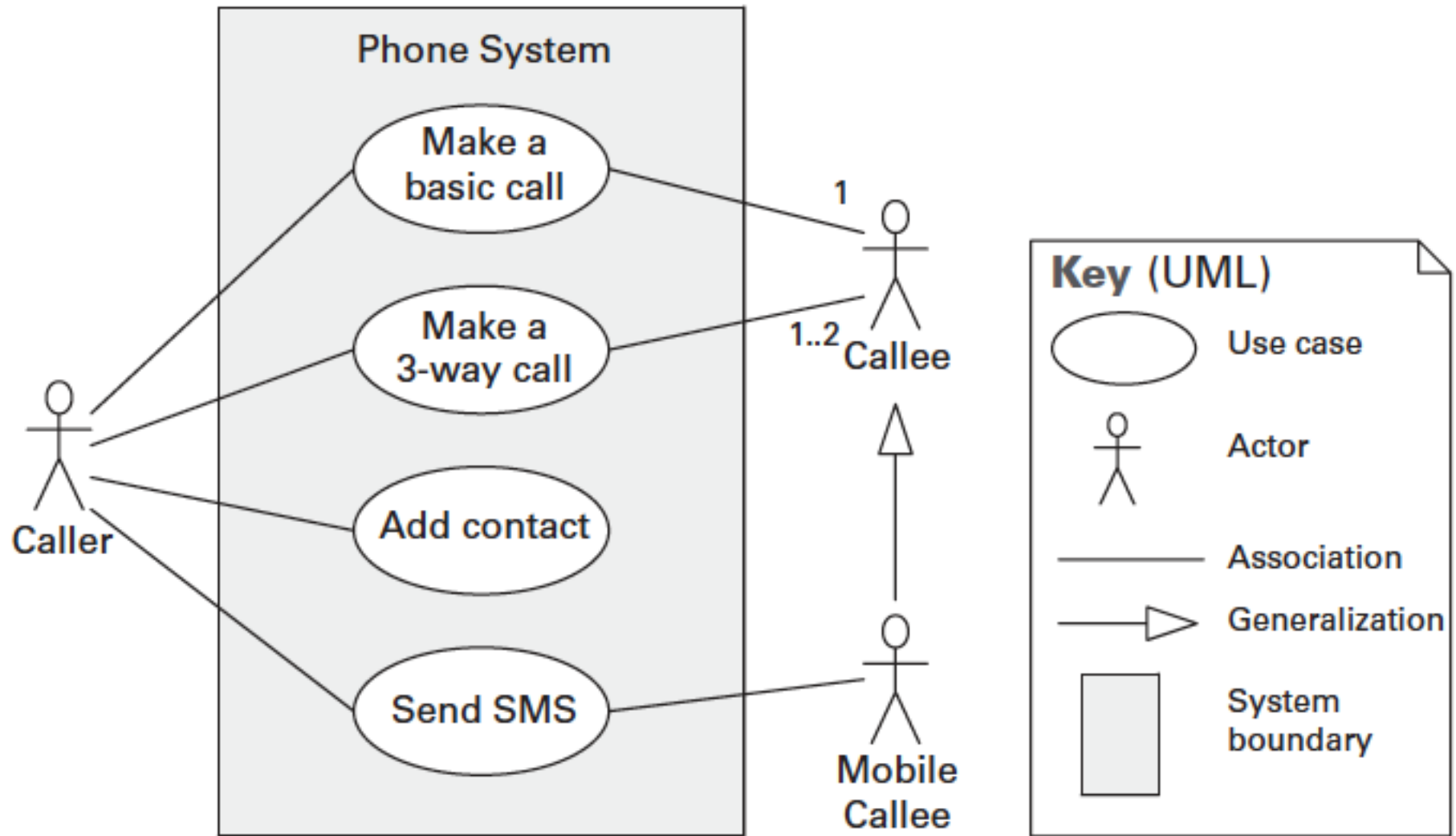
- Ne esistono due tipologie:
 - Una permette di descrivere ciò che accade tra gli elementi strutturali di un sistema durante uno **scenario**.
 - Descrizione avviene tramite **tracce** (*traces*).
 - Le tracce sono sequenze di attività o interazioni che descrivono la risposta del sistema ad uno stimolo.
 - Un'altra mostra il comportamento completo di un elemento strutturale oppure un insieme di elementi.
 - È spesso **state based**.
 - Definito **“comprehensive model of behavior”** perché grazie ad esso è possibile ricostruire tutti i possibili cammini da uno stato iniziale a uno finale.

Quali notazioni utilizzare

- Utilizzeremo UML 2.0.
- Mette a disposizione notazioni per:
 - Descrivere le *Traces*.
 - Descrivere *Comprehensive Models*.



Descrivere Traces – Use Case Diagrams



Descrivere Tracce – Use Case Diagrams

Name: Make a basic call

Description: Making a point-to-point connection between two phones.

Primary actors: Caller

Secondary actors: Callee

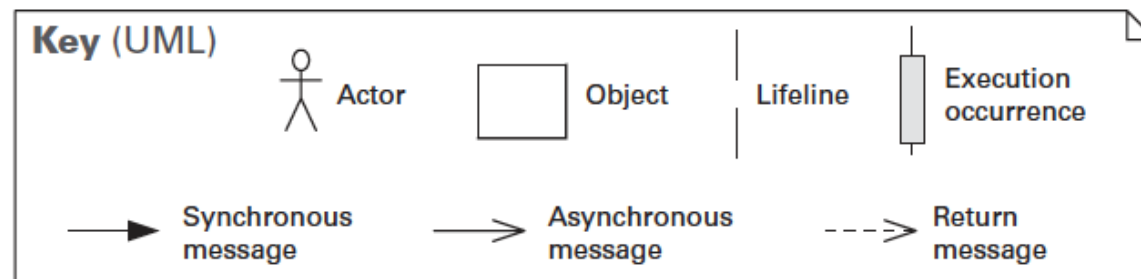
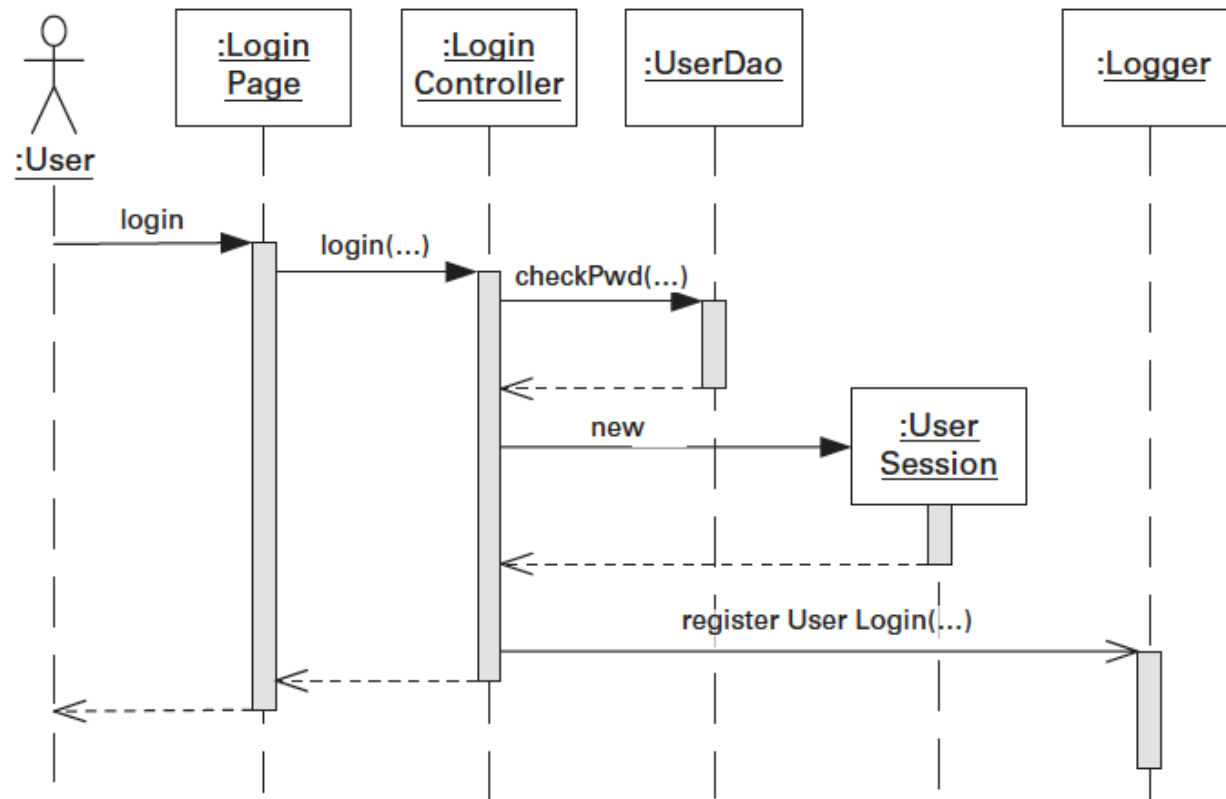
Flow of events:

The use case starts when a caller places a call via a terminal, such as a cell phone. All terminals to which the call should be routed then begin ringing. When one of the terminals is answered, all others stop ringing and a connection is made between the caller's terminal and the terminal that was answered. When either terminal is disconnected—someone hangs up—the other terminal is also disconnected. The call is now terminated, and the use case is ended.

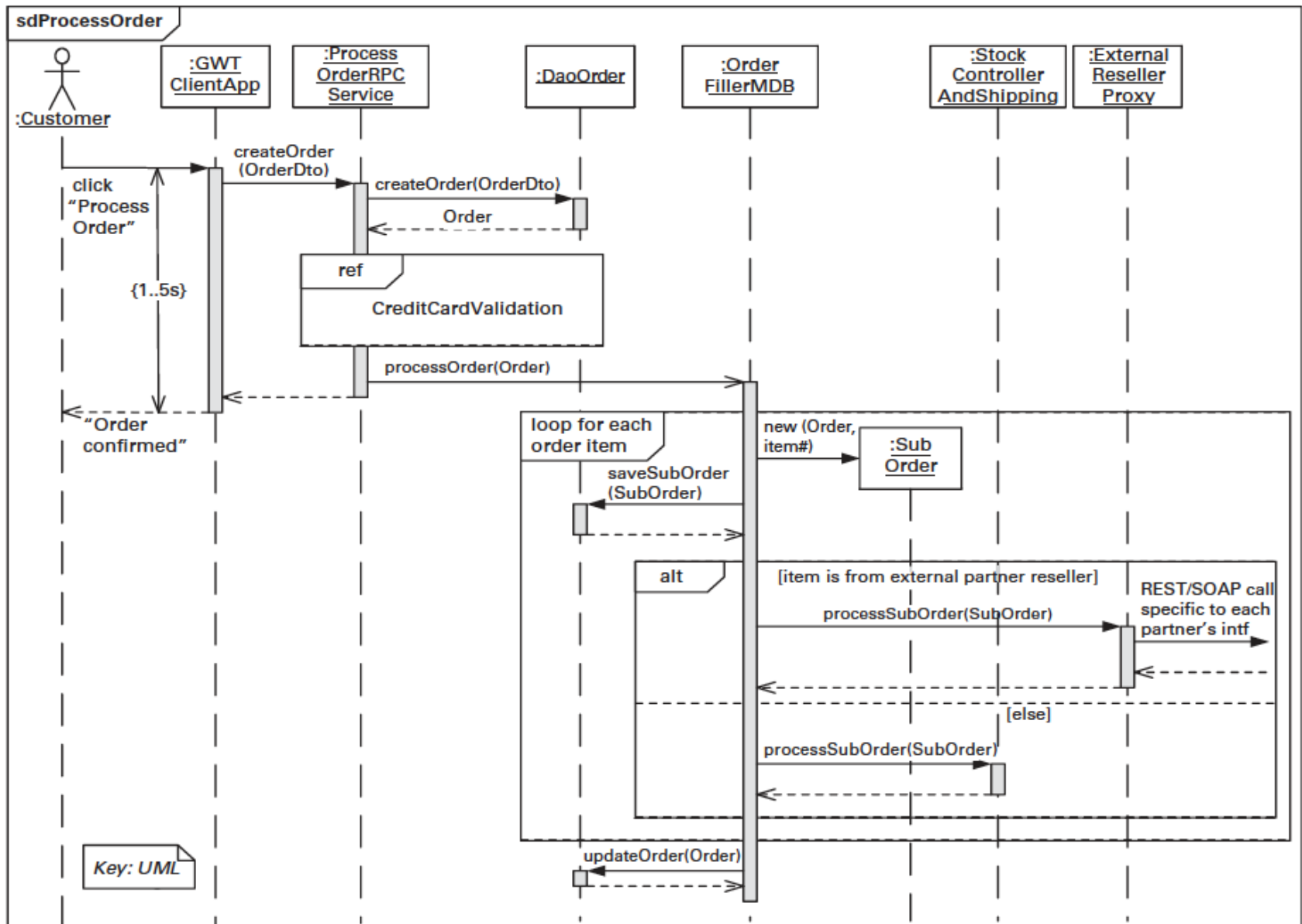
Exceptional flow of events:

The caller can disconnect, or hang up, before any of the ringing terminals has been answered. If this happens, all ringing terminals stop ringing and are disconnected, ending the use case.

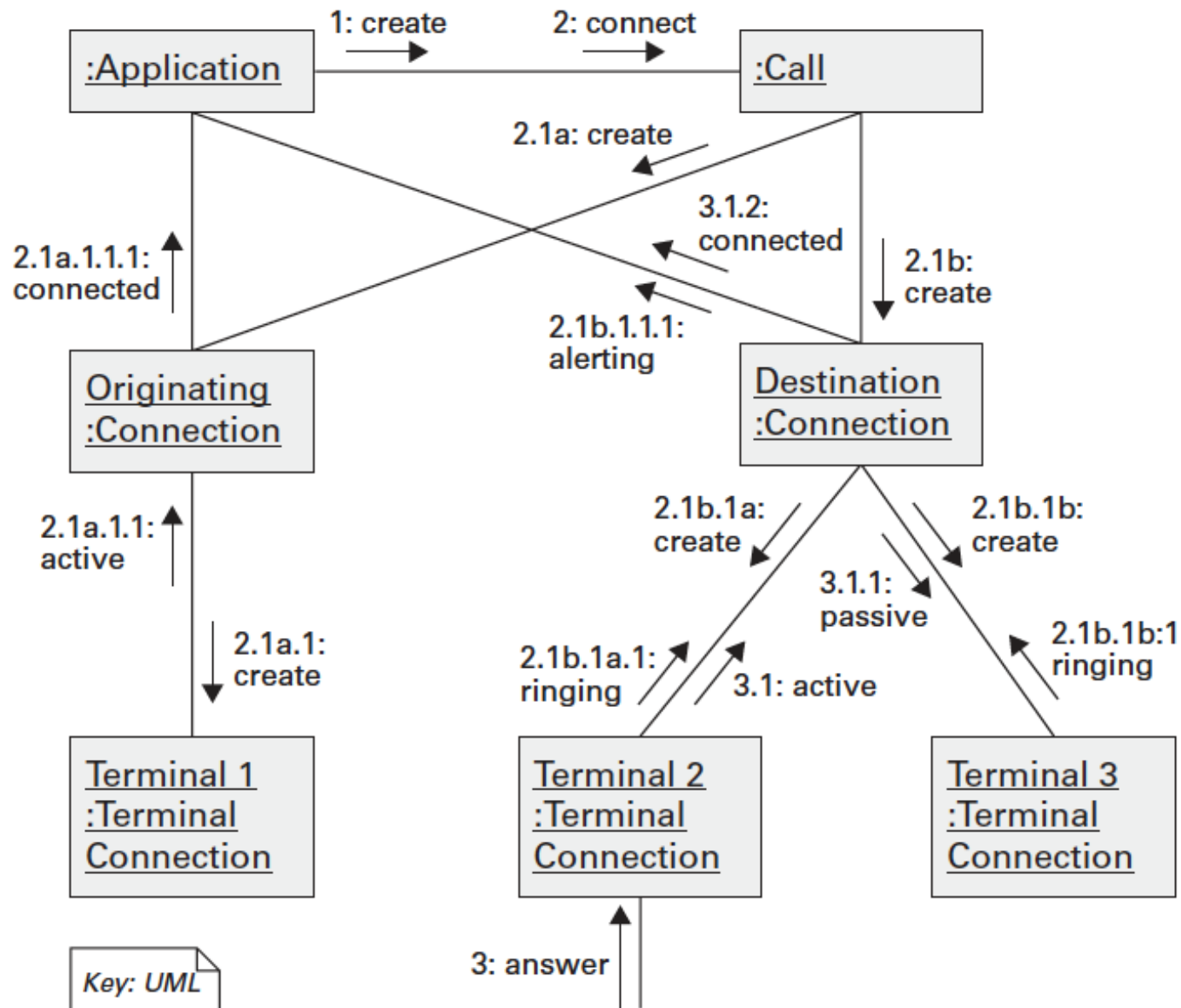
Descrivere Traces – Sequence Diagrams



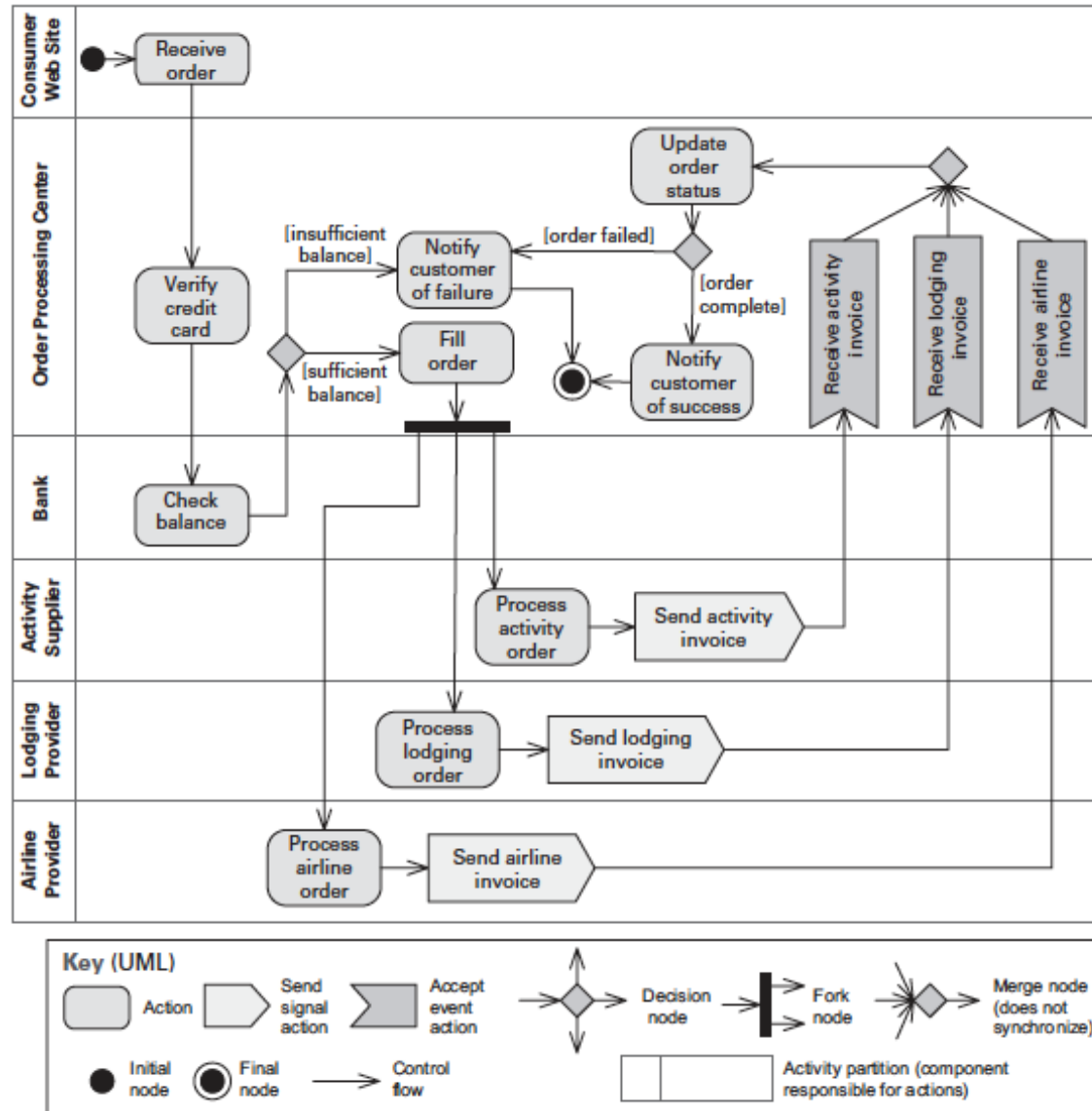
Descrivere Traces – Sequence Diagrams



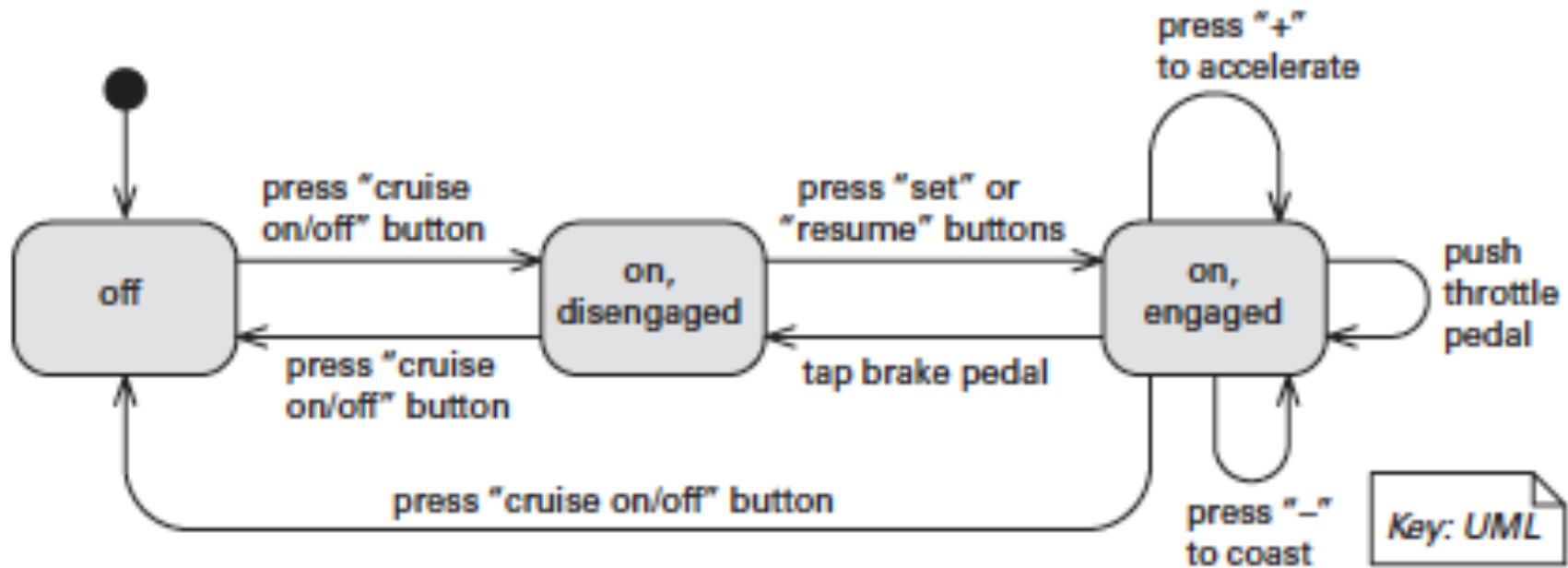
Describing Traces – Communication Diagrams



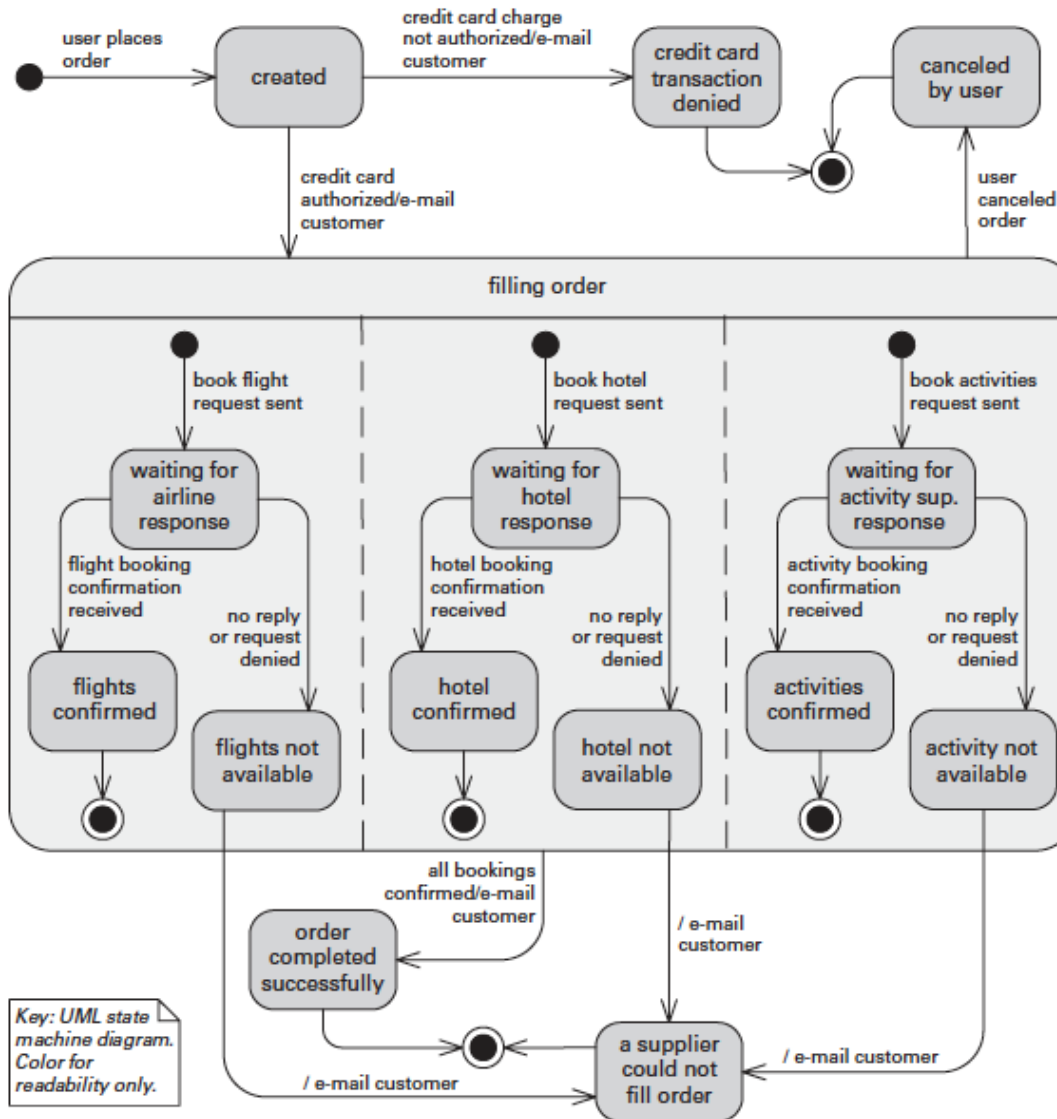
Descrivere Traces – Activity Diagrams



Describe Comprehensive Models – State Machine Diagrams



Descrivere Comprehensive Models – State Machine Diagrams



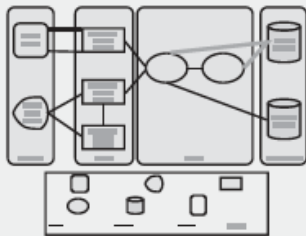
Dove documentare il behavior

- Può avere la propria sezione nell'*element catalog*.
 - Per descrivere una transazione complessa, possiamo mostrare come gli elementi interagiscono (es. sequence o communication).
- Può essere parte della documentazione di interfaccia.
 - Per rappresentare meglio, ad esempio, le sezioni *“semantics”* e *“usage guide”*.
- Può essere usata nella sezione *razionale* che include i risultati delle analisi.

Alcune sezioni del Template

Template for a View

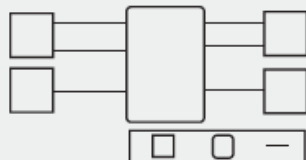
Section 1. Primary Presentation



Section 2. Element Catalog

- Section 2.A. Elements and Their Properties
- Section 2.B. Relations and Their Properties
- Section 2.C. Element Interfaces
- Section 2.D. Element Behavior

Section 3. Context Diagram



Section 4. Variability Guide

Section 5. Rationale

Template for Documentation Beyond Views

Architecture documentation information

- Section 1. Documentation Roadmap
- Section 2. How a View Is Documented

Architecture information

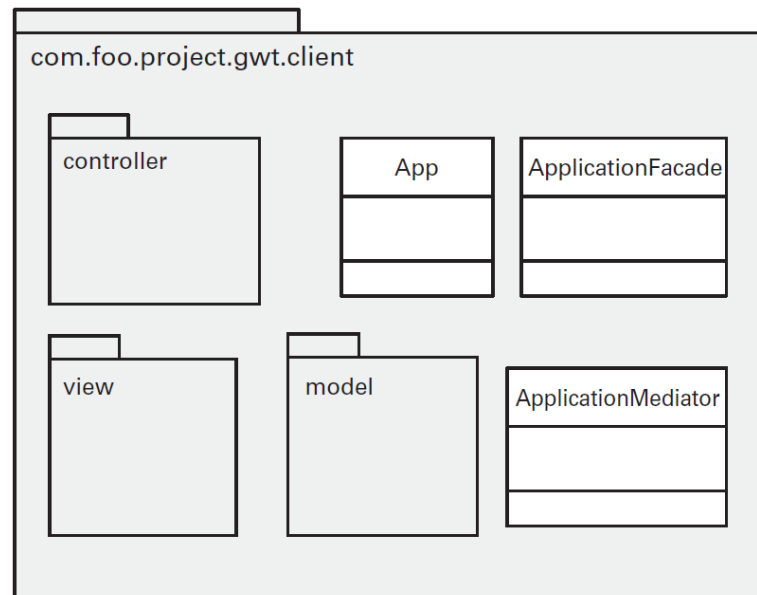
- Section 3. System Overview
- Section 4. Mapping Between Views
- Section 5. Rationale
- Section 6. Directory — index, glossary, acronym list

Documentare le Module Views

- Le Module Views servono a rappresentare la struttura dell'architettura.
- Moduli devono essere rappresentati come:
 - UML packages,
 - UML classes,
 - UML interfaces.

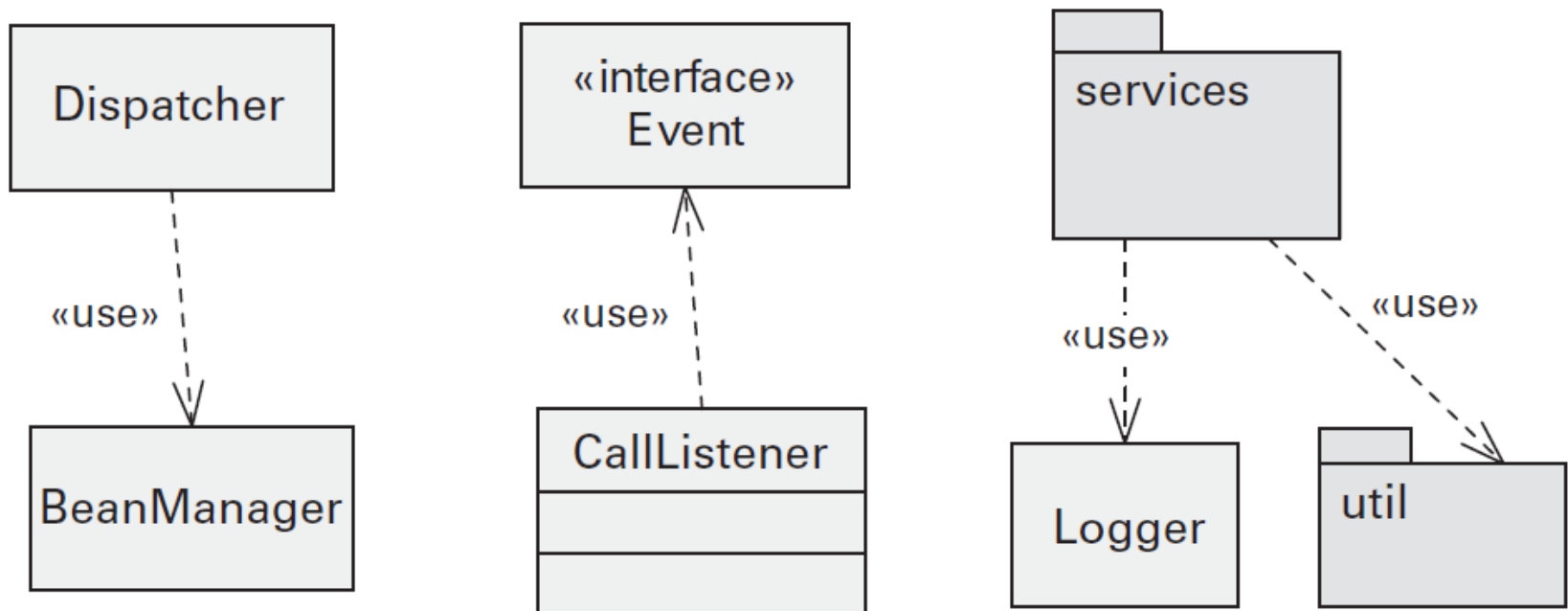
Decomposition Style

- I Moduli sono rappresentati come UML Packages o UML Classes.
- In UML la decomposizione di un modulo in sotto-moduli è rappresentata innestando packages, classes o interfaces all'interno di packages.



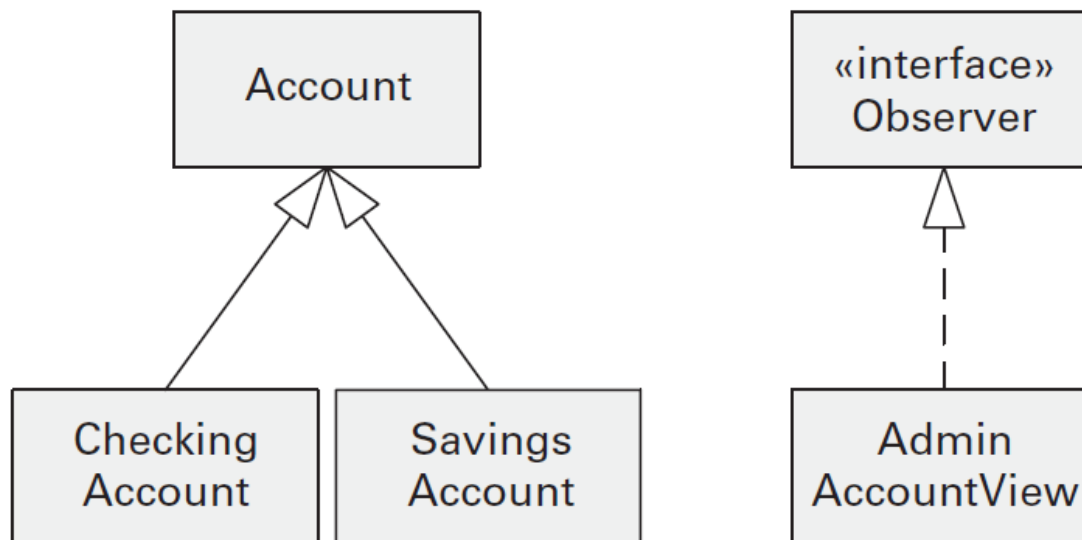
Uses Style

- Questo stile descrive le dipendenze d'uso tra i moduli.
 - UML Packages, UML Classes, UML Interfaces.
- In UML le dipendenze sono rappresentate mediante le frecce di dipendenza.



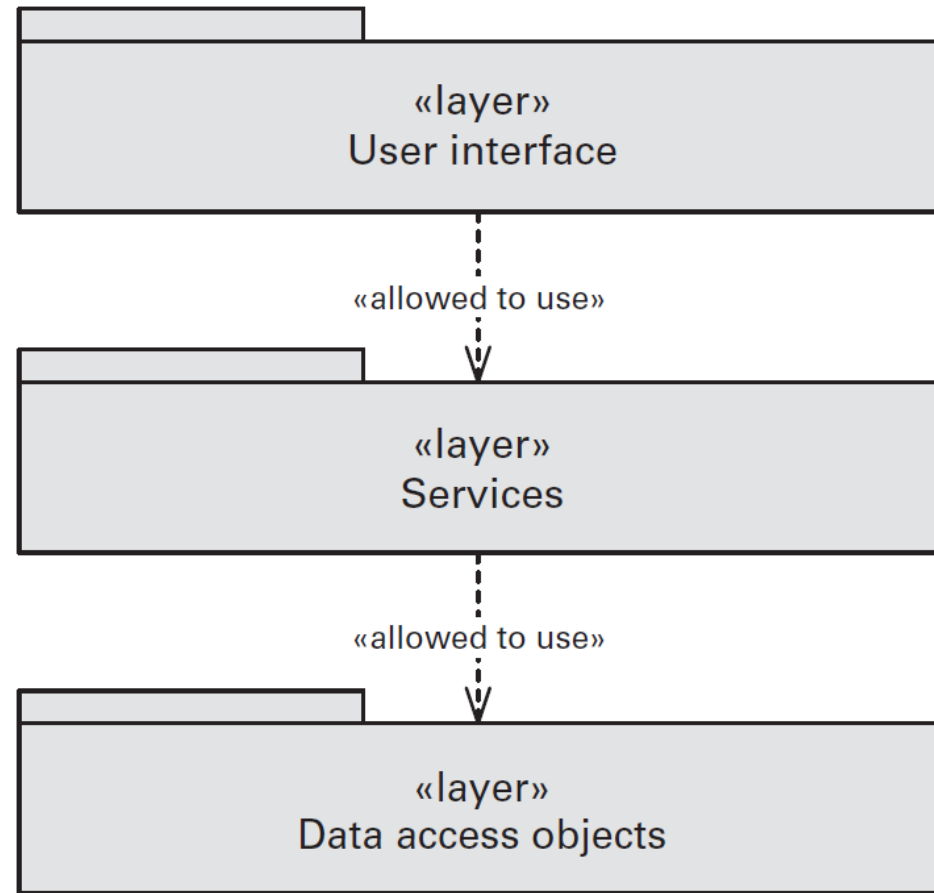
Generalization Style

- I moduli devono essere rappresentati utilizzando Classes e Interfaces.
- Generalization (relazione *is-a*) tra moduli è rappresentata in UML utilizzando la relazione UML di generalizzazione (*class inheritance*).
- Interface realization (relazione *is-a*) è rappresentata utilizzando l'omonima relazione UML.



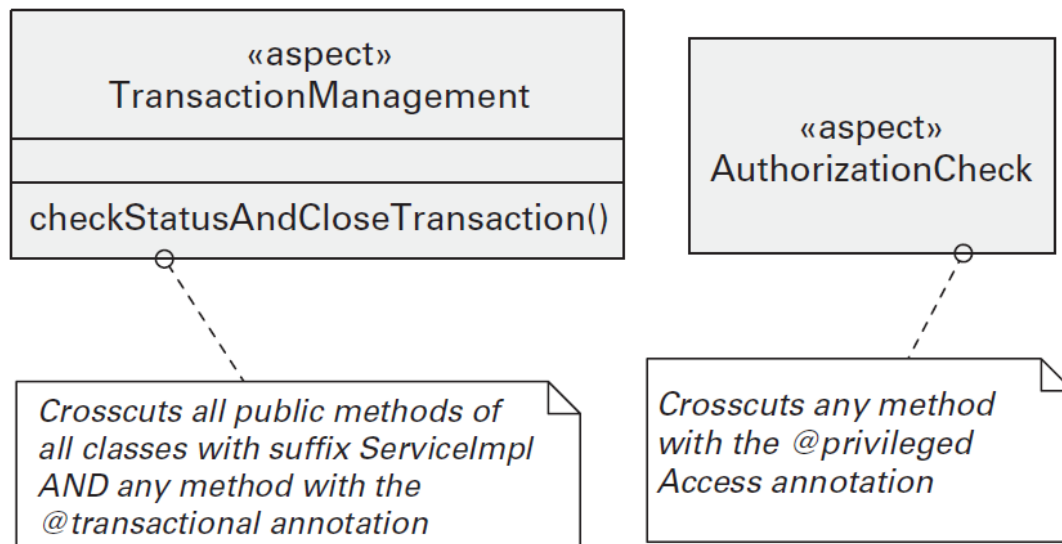
Layered Style

- UML non ha built-in notation per i layers.
- Un layer raggruppa moduli → layer rappresentato con un package stereotipato come <<layer>>.
- La relazione *allowed-to-use* tra i layers deve essere rappresentata con una relazione stereotipata.



Aspects Style

- UML non ha built-in notation per gli aspects.
- Usare una classe stereotipata come <<aspect>>.
- La relazione *crosscut* tra un aspect e classi può essere rappresentata con dipendenze UML stereotipate.
- Per limitare la complessità del modello si possono usare le *annotations*.



Annotations

- Java annotations e .NET attributes danno la possibilità di arricchire direttamente elementi di codice sorgente con metadati personalizzati.

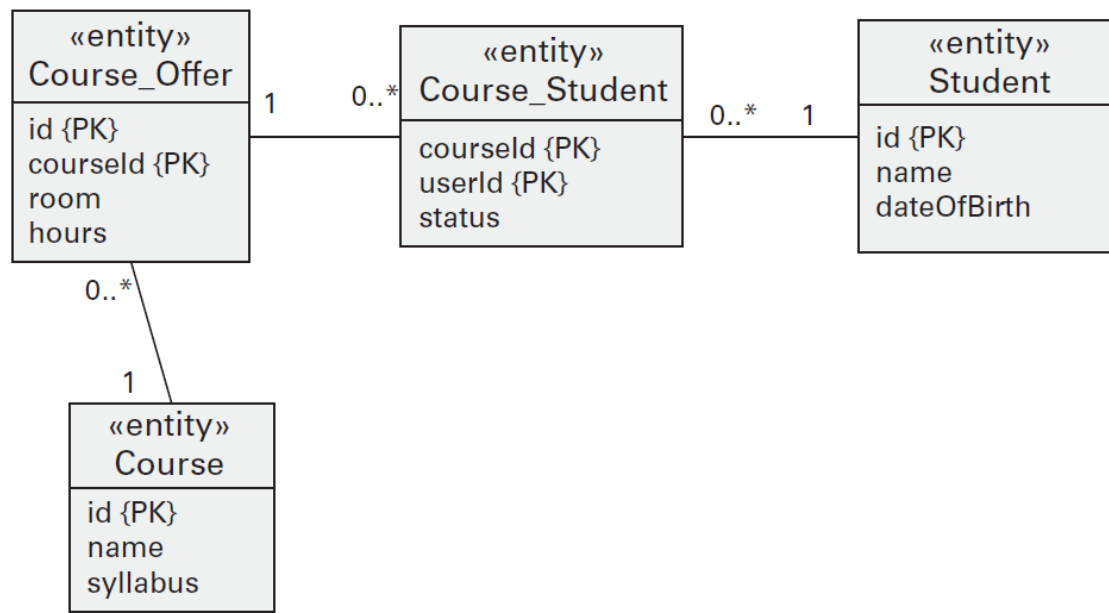
```
1  @Author(  
2      name="Vasian Cepa",  
3      name="Sven Kloppenburg")  
4  @DAO  
5  public class Account {  
6  
7      @PrimaryKey  
8      private string accountID;  
9  
10     ...  
11  
12     @Transactional(kind=Required)  
13     public void credit(float amount) {...}  
14  
15     @Transactional(kind=Required)  
16     public void debit(float amount) {...}  
17  
18     public float getBalance() {...}  
19     ...  
20 }
```

Account	
{@ DAO}	
{@ Author=name~Vasian Cepa:name~Sven Kloppenburg}	
- accountID : string	
{@ PrimaryKey}	
+ credit (amount : float) : void	
{@ Transactional=kind~Required}	
...	



Data Model Style

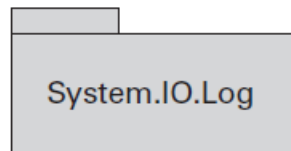
- I Data Model devono essere documentati con UML class diagram.
- Le classi devono avere lo stereotipo <<entity>>.
- Si possono usare *constraints* per indicare gli attributi che definiscono le PK.



Sommario dei simboli UML usati nelle Module Views



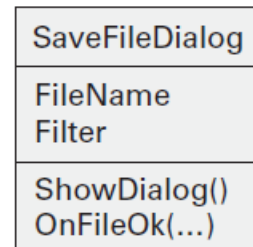
class:
used to represent a module, aspect, or data entity



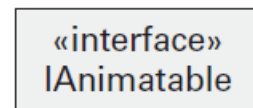
package:
used to represent a module or layer



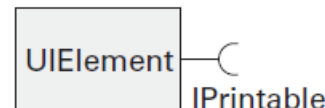
class with provided interface:
used to represent an interface provided by a module




class with attributes and operations:
used to represent a module, aspect, or data entity





interface:
used to represent an interface of a module





class with required interface:
used to represent an interface required by a module

 **dependency (with a stereotype):**
used to represent relations such as *use* and *allowed to use*

 **association:**
used to represent logical associations between data entities, often annotated with multiplicities

 **aggregation:**
used to represent aggregation of data entities into an aggregate entity

 **generalization:**
used to represent a generalization relation

 **interface realization:**
used to represent a realization relation between an interface and a module realizing that interface

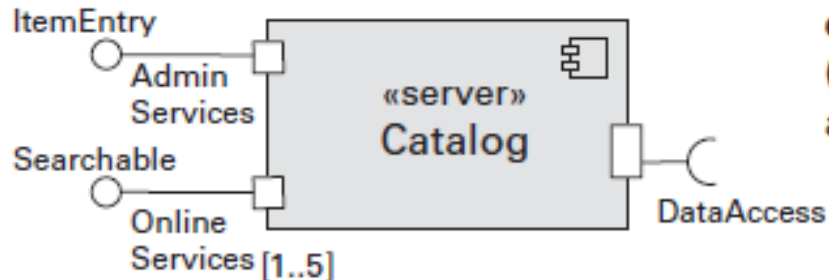
Documentare le C&C Views

- **C&C component instances** devono essere rappresentate utilizzando istanze di UML components in *object diagrams* oppure *component diagrams*.
- **C&C component types** devono essere rappresentati utilizzando *UML components* in *component diagrams*.
- Instances e types non devono mai essere rappresentati nello stesso diagramma.

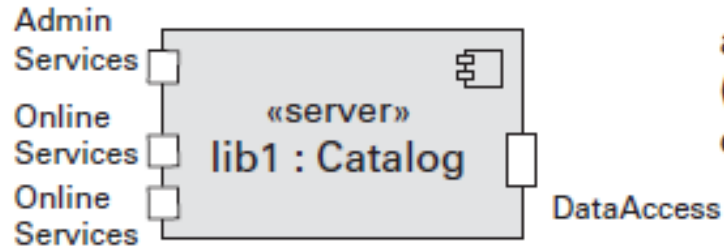
Component types and instances

- In UML vengono distinte con la stessa convenzione utilizzata per distinguere le classi e gli oggetti.
- I componenti devono essere stereotipati per indicare il nome del corrispondente tipo utilizzato nello style guide.
- Component ports devono essere rappresentati utilizzando UML ports.
 - Devono avere un identificatore e possono avere un indicatore di molteplicità.
- UML provided e requested interfaces possono essere collegate ai ports.
 - In genere questo viene fatto per i types e non per le instances.

Esempi di Componenti



component type
(with UML ports
and interfaces)



a component instance
(with ports explicitly
documented)



a component
instance

lib2 non documenta
esplicitamente i suoi ports.

Connectors

- Possono essere rappresentati in modi differenti in UML.
 - Trade off tra la quantità di informazioni che si vuole documentare, la lunghezza del documento.
- Si possono rappresentare essenzialmente in due modi.



using a UML connector to represent a C&C connector



using a UML component to represent a C&C connector

Prima scelta

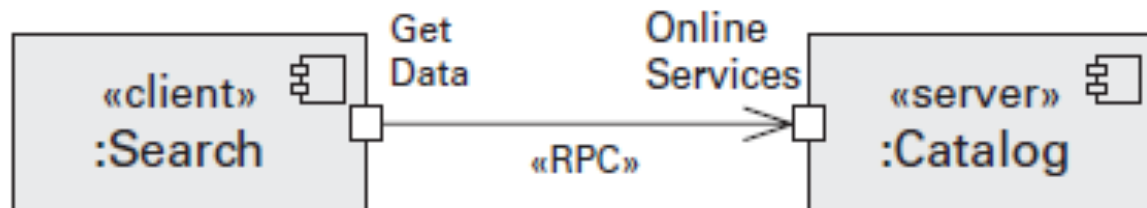
- Undecorated line
 - Il tipo di connettore deve essere identificato tramite uno stereotipo.
 - Questa scelta è limitante perché il connettore può avere substructure, properties, behavioral descriptions, etc.
- Roles del connettore indicati:
 - etichettando le estremità del connettore;
 - Implicitamente quando il connettore è collegato a due ports.

Seconda scelta

- Rappresentare un connettore come un componente.
 - I componenti possono avere associati substructure, properties, behavioral descriptions, etc.
- Roles rappresentati tramite ports del componente.

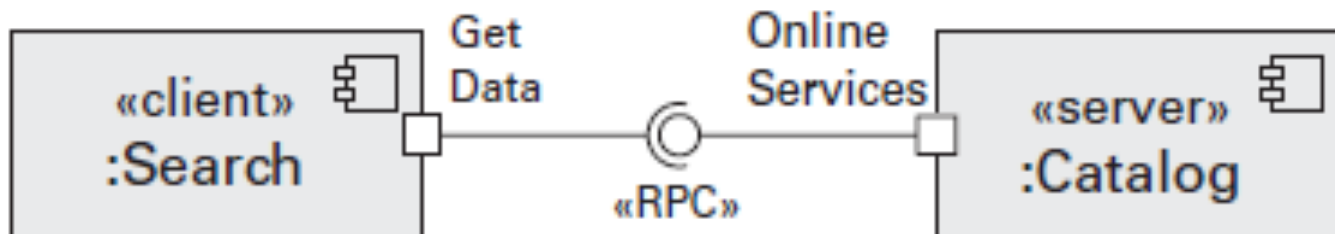
UML connector strategies

- Esistono due varianti
 - Il connettore presenta una estremità navigabile per rappresentare una direzione associata con l'interazione.
 - La documentazione deve identificare il significato della navigabilità.
 - Esempio: rappresenta l'inizio di una interazione o la direzione in cui i dati sono passati?
 - Questa scelta è poco utile quando si vogliono rappresentare interazioni bidirezionali come i protocolli.



UML connector strategies

- UML assembly connector.
 - Rappresentato utilizzando una notazione ball-and-socket.
 - Mappa naturalmente connettori tra interfacce provided e required.
 - Esempio semplici connettori call-return
 - Poco utili quando
 - i connettori non sono del tipo provided/required,
 - si vogliono rappresentare comunicazioni bidirezionali.



Tagged values

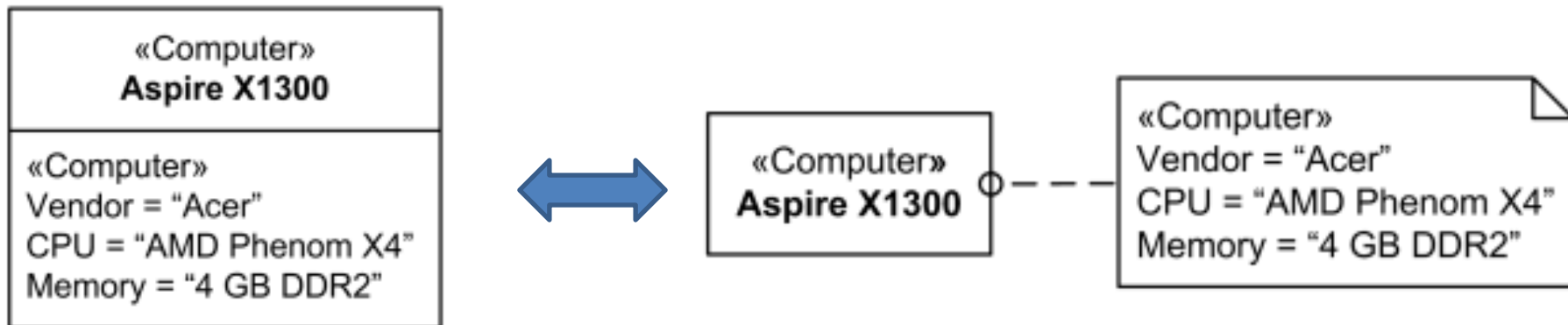
- Definisce proprietà aggiuntive per elementi del modello.
- Possono essere definiti per elementi esistenti a per stereotipi.
- Rappresentati come una coppia *tag-value*
 - Tag rappresenta la proprietà e il valore rappresenta il valore della proprietà.

Tagged values

- Utilizzati per aggiungere proprietà aggiuntive riguardanti:
 - code generation
 - version control
 - configuration management
 - authorship
 - etc.

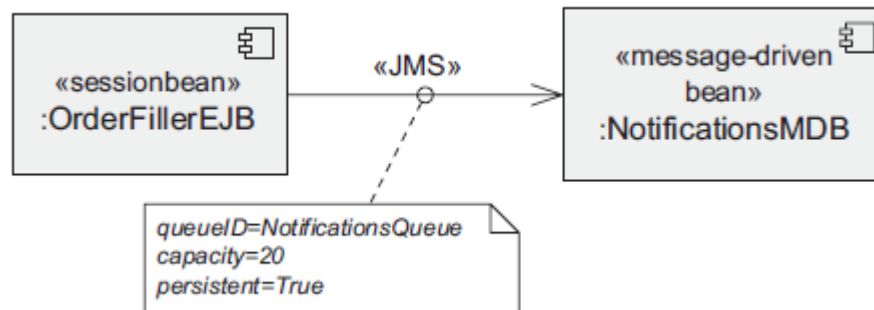
Tagged values

- Possono essere mostrati in class compartments sotto il nome dello stereotipo.
- A volte è richiesto un compartimento separato per visualizzare i valori dei tags.
- Possono essere mostrati in commenti collegati.



Utilizzo dei Tagged Values

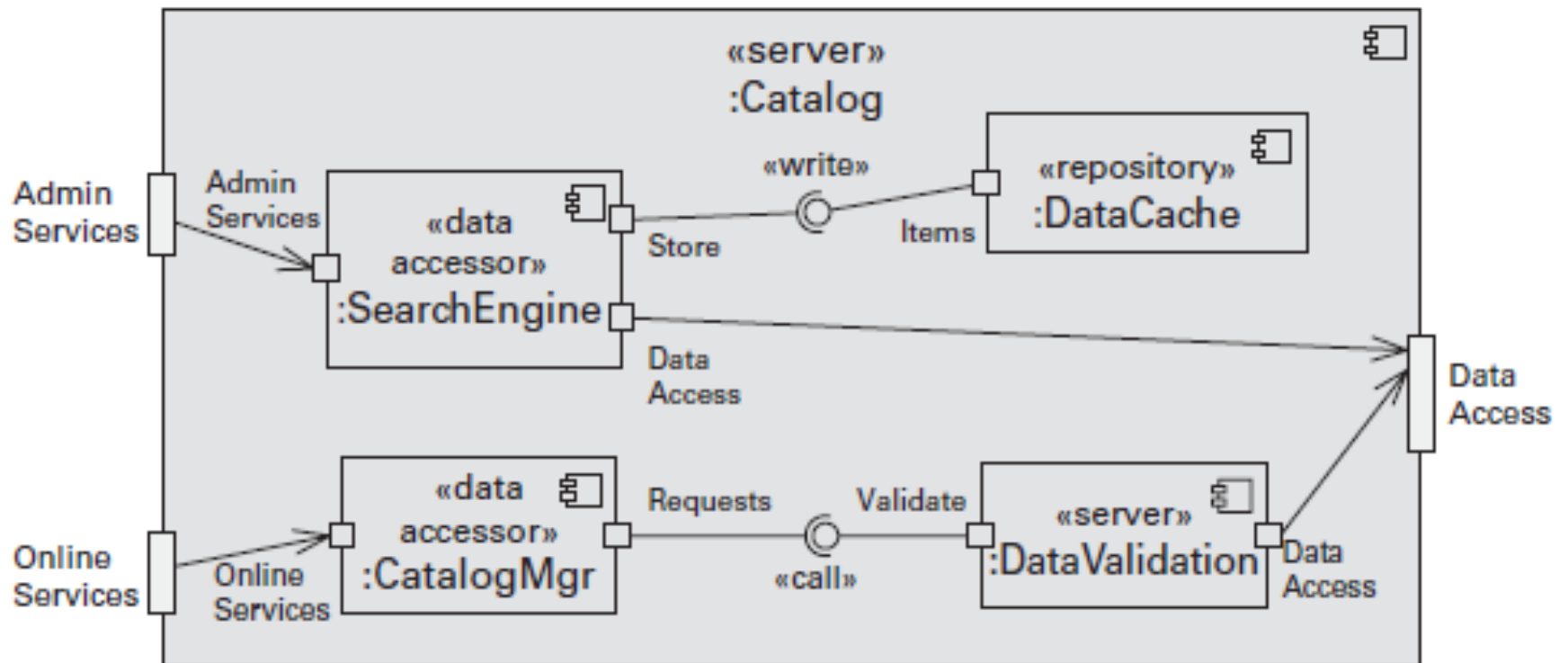
- Possono essere utilizzati per associare attributi aggiuntivi ai connettori UML.
- Si può quindi:
 - Definire uno stereotipo del connettore.
 - Definire tagged values per quel connettore.
 - Indicare Name e dataType
 - Esempio: `queueID : String; capacity : integer; persistent : Boolean`



Substructure

- Substructures devono essere rappresentate in UML utilizzando:
 - UML components;
 - UML delegation connectors;
 - Connettori che collegano ports del componente più esterno con i ports dei componenti più interni.
 - Questi devono essere innestati in un macro componente.

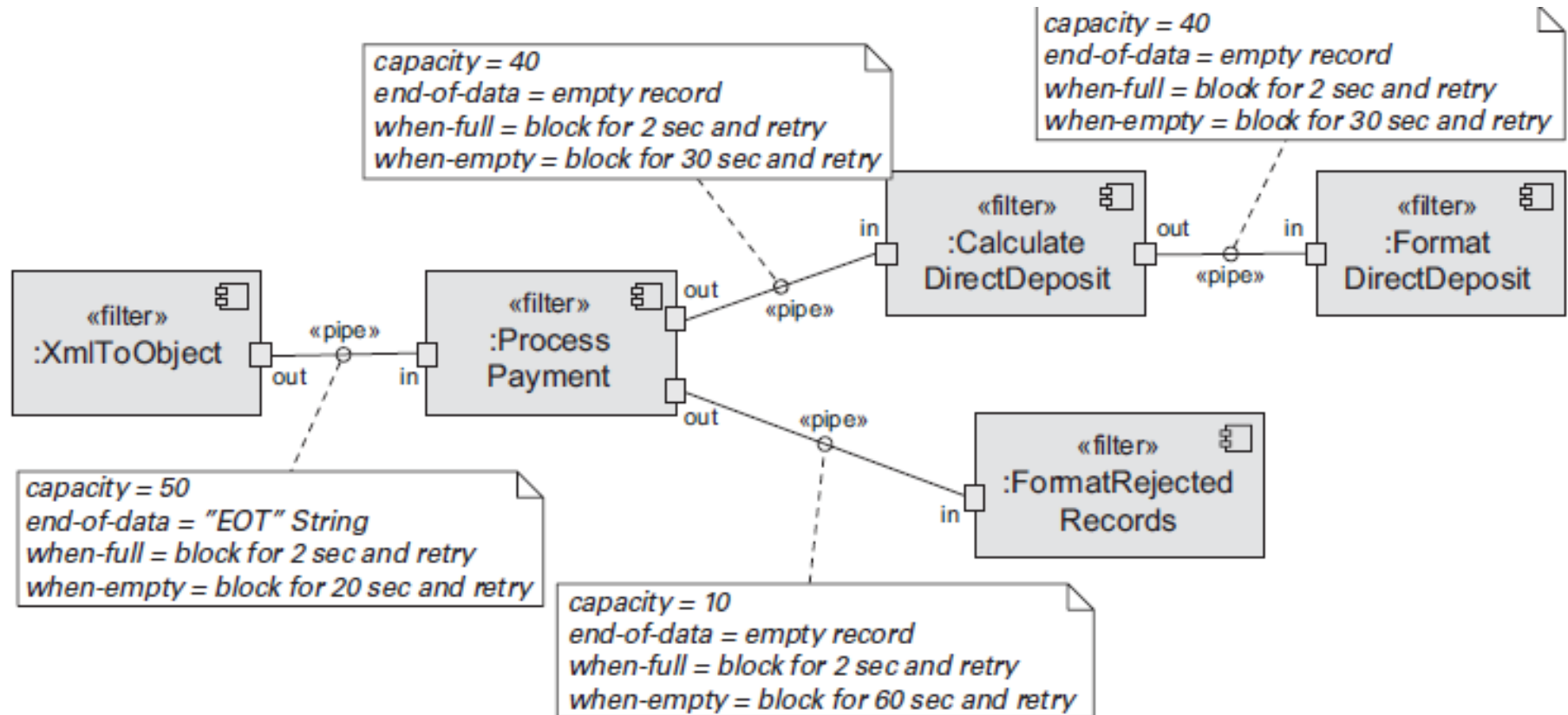
Substructure



Esempio pipe-and-filter Style

- Quando si documenta una data C&C View in UML si deve utilizzare uno stereotipo per identificare il tipo di ogni componente e connettore
 - Assicura una chiara relazione con i tipi di componenti e connettori definiti nello style guide usato per creare la view.
- Se sono stati definiti application-specific subtypes per questi tipi → I subtypes devono essere identificati col nome delle istanze dei componenti.

Esempio pipe-and-filter Style



Documentare le Allocation views

- Allocation views presentano mapping tra elementi software ed elementi di environment.
- Environmental elements sono nonsoftware elements (come nodi hardware) che rappresentano entità che vengono associate agli elementi software del sistema progettato.

UML Deployment Diagram

- Deployment diagrams mostrano architetture di esecuzione di sistemi software.
- Rappresentano l'assegnazione (*deployment*) di *software artifacts* a *deployment targets* (generalmente *nodes*).

Elements

- **Nodes** rappresentano o hardware devices oppure software execution environments.
- Possono essere connessi mediante **communication paths** per creare sistemi di diversa complessità.
- **Artifacts** rappresentano elementi concreti risultanti da un processo di sviluppo e che sono deployed su nodi.
- In UML 2.* artifacts sono deployed su nodes.
 - Artifacts possono implementare (**manifest**) components.
 - Components sono indirettamente deployed su nodes mediante artifacts.

Nodes

- **Node** è un **target di deploy** che rappresenta una risorsa computazionale su cui gli artefatti possono essere installati (deployed) per l'esecuzione.
- Può essere
 - *Device*
 - *Execution environment*



Application Server Node

Device

- È un nodo che rappresenta una risorsa fisica computazionale con capacità di elaborazione.
- È rappresentato come un nodo con stereotipo «device».
- UML fornisce stereotipi non standard per i device.
 - «application server»
 - «client workstation»
 - «mobile device»
 - «embedded device»



Application Server device

Device

- Può essere rappresentato con un'icona personalizzata.
 - In genere gli Stereotipi possono essere avere un'icona personalizzata
- Profili, stereotipi e tagged values forniscono alle icone proprietà aggiuntive.

Device Esempi



«application server»

IBM System x3755 M3

Application Server device depicted using custom icon

«Computer»
Vendor = "Acer"
CPU = "AMD Phenom X4"
Memory = "4 GB DDR2"



*Computer stereotype with tags applied to **Device** clas*



«database server»

Sun SPARC Server

Database Server device depicted using custom icon



«mobile device»

smartphone

Mobile smartphone device depicted using custom icon

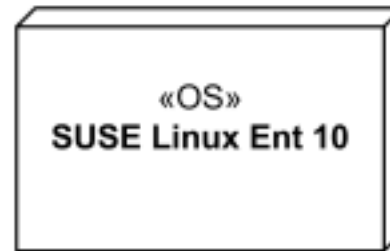
Execution Environment

- È un (software) node che offre un ambiente di esecuzione per specifici tipi di componenti che vengono distribuiti su di esso in forma di artefatti eseguibili.
- Implementa un insieme di servizi necessari ai componenti a tempo di esecuzione.
 - Questi servizi vanno documentati.
- Componenti del tipo appropriato vengono distribuiti in ambienti di esecuzione specifici.
- Rappresentato come un nodo con stereotipo «executionEnvironment»

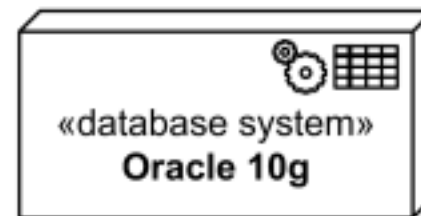


Execution Environment

- UML fornisce stereotipi non standard per gli execution environment.
 - «OS»
 - «workflow engine»
 - «database system»
 - «J2EE container»
 - «web server»
 - «web browser»



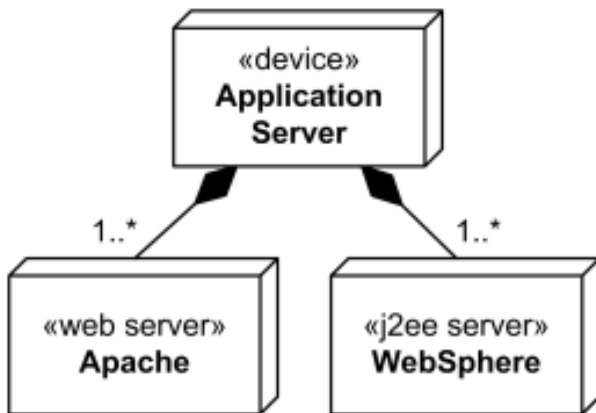
Linux Operating System Execution Environment



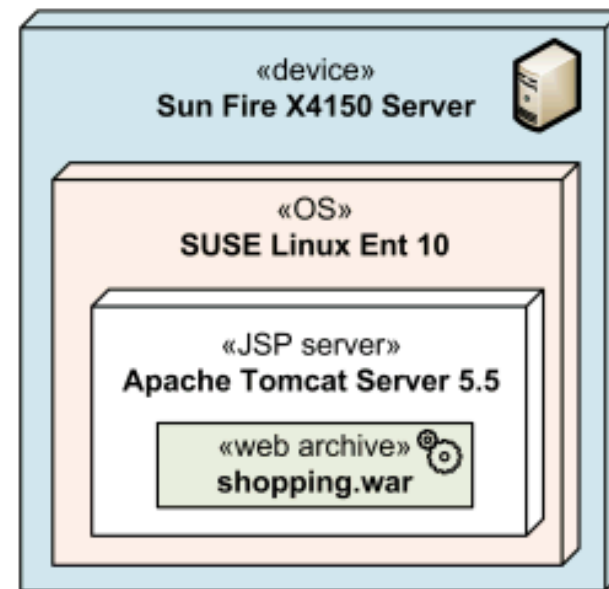
Oracle 10g DBMS Execution Environment

Hierarchical Node

- **Hierarchical nodes** possono essere modellati utilizzando la **composition** oppure definendo una **internal structure**.



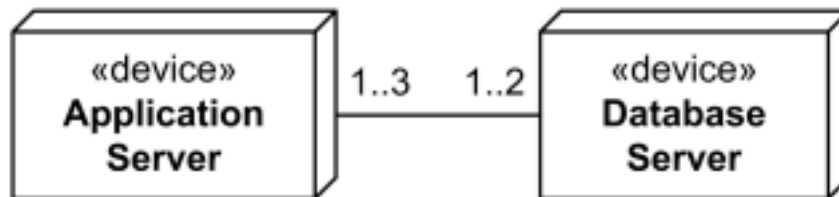
Application server box runs several web servers and J2EE servers



Several execution environments nested into server device

Communication Path

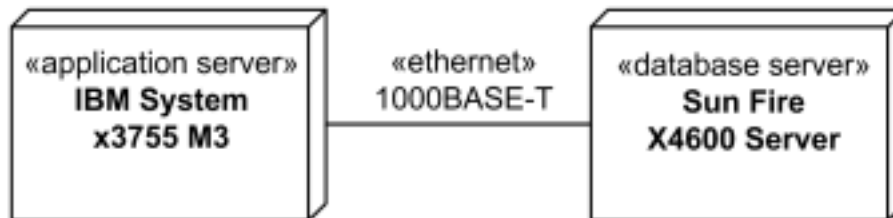
- È una associazione tra due deployment targets
- Permetto lo scambio di *signals* e *messages*.
- È rappresentata come associazione, e non ha notazioni addizionale rispetto alle associazioni.



Communication path between several application servers and database servers.

Communication Path

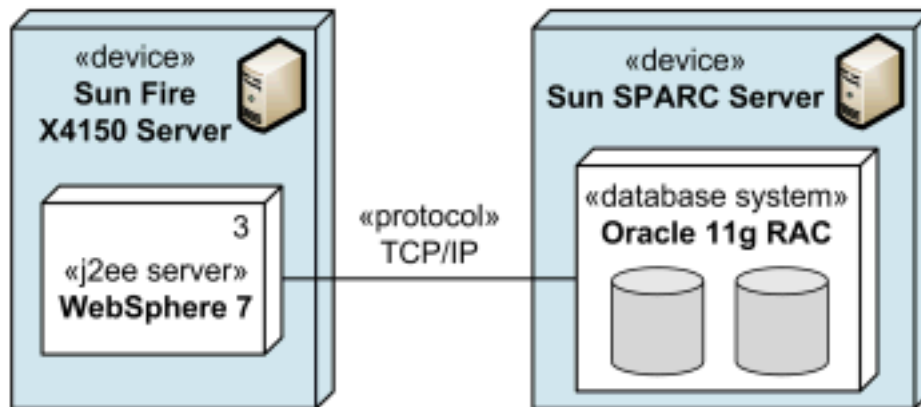
- Nel caso di physical devices → communication path in genere rappresentano un tipo di connessione fisica tra i nodes.



Gigabit Ethernet as communication path between application and database servers.

Communication Path

- Nel caso di execution environments → communication path in genere rappresentano un tipo di protocollo tra i nodes.



TCP/IP protocol as communication path between J2EE server and database system.

Artifact

- È un classifier che rappresenta un'entità fisica di informazione che:
 - è usata o è prodotta da un processo di sviluppo software,
 - è distribuita o eseguita su un sistema.
- È una sorgente del deployment verso un node.
- Una particolare istanza (o copia) di un artifact è distribuita su una istanza di un node.

Artifact

- Possono avere:
 - *proprietà* che rappresentano caratteristiche del manufatto,
 - *operazioni* che possono essere eseguite sulle istanze.
- Hanno un attributo, `fileName`, che viene utilizzato per fare riferimento al manufatto in un contesto fisico.
 - Esempio il nome del file o URI.
- Un artefatto potrebbe avere artefatti nidificati.

Artifact

- Esempi di artefatti comunemente utilizzati:
 - text document
 - source file
 - script
 - binary executable file
 - archive file
 - table in a database

Artifact

- Il Profilo Standard UML Standard definisce diversi stereotipi che si applicano agli artefatti.

«file»	A physical file in the context of the system developed.
--------	---

- Sottoclassi di «file»

«document»	A generic file that is not a «source» file or «executable».
«source»	A source file that can be compiled into an executable file.
«library»	A static or dynamic library file.
«executable»	A program file that can be executed on a computer system.
«script»	A script file that can be interpreted by a computer system.

- Stereotipi standard possono essere ulteriormente specializzati con stereotipi dipendenti dalla piattaforma e/o dal linguaggio di implementazione.
 - Ad esempio, il profilo di Java potrebbe definire «jar» come una sottoclasse di «executable» per gli archivi Java eseguibili.

Artifact

- Un artifact è rappresentato come una classe con stereotipo «artifact».
 - Può mostrare anche un'icona di un documento in alto a destra.
- Può essere rappresentato anche tramite un'icona.



Artifact web-app.war



C# source file artifact UserService.cs



Library commons.dll

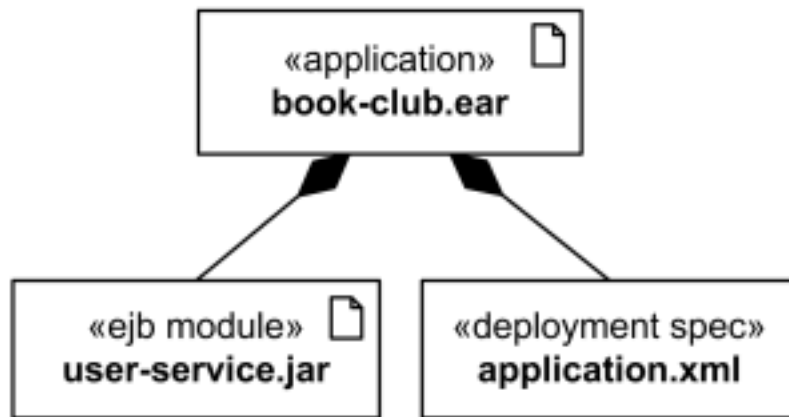


web-tools-lib.jar

Artifact web-tools-lib.jar

Le associazioni tra Artefatti

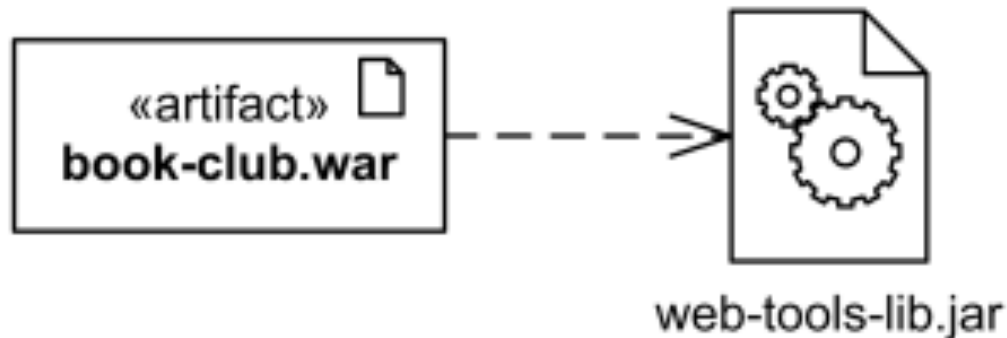
- Gli artefatti possono essere coinvolti in associazioni ad altri manufatti.
 - Esempio: associazioni di composizione.



Application book-club.ear artifact contains EJB user-service.jar artifact and deployment descriptor.

Dipendenza tra Artefatti

- Gli artefatti possono essere coinvolti in relazioni di dipendenza con altri artefatti.
- La dipendenza tra artefatti è rappresentata nello stesso modo della dipendenza generica.



The book-club.war artifact depends on web-tools-lib.jar artifact.

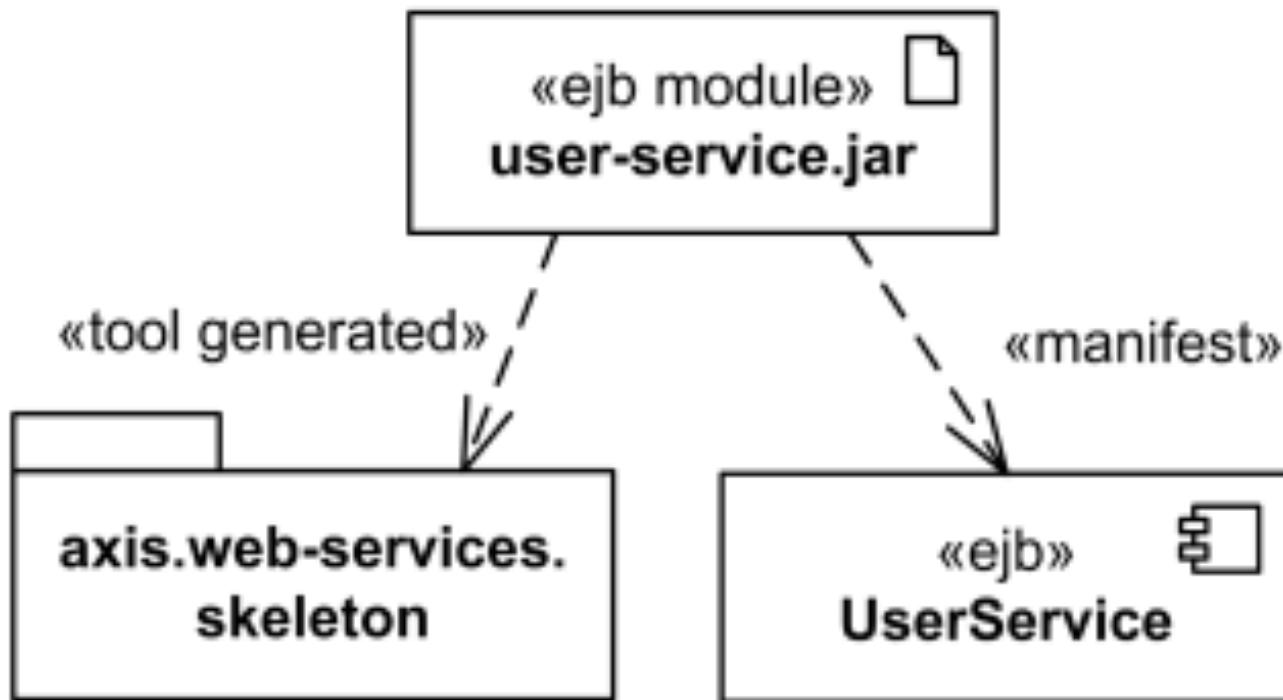
Manifestation

- Manifestation è una abstraction relationship che rappresenta
 - l'implementazione di uno o più elementi di modello da parte di un artefatto, oppure
 - L'utilizzo degli elementi di modello nella costruzione o generazione dell'artefatto.
- Un artifact «manifests » uno o più model elements.

Manifestation

- In alcuni profili è possibile indicare particolari forme di manifestation.
 - Esempio: utilizzando stereotipi quali «tool generated» e «custom code».
- È rappresentata come una abstraction, cioè come una freccia tratteggiata rivolta dal manufatto all'elemento di modello ed è etichettato con lo stereotipo «manifest».

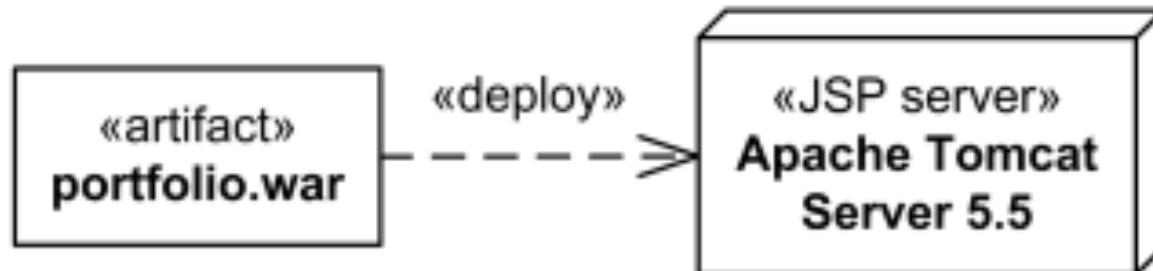
Manifestation



EJB component UserService and skeleton of web services are manifested (implemented) by EJB module user-service.jar artifact

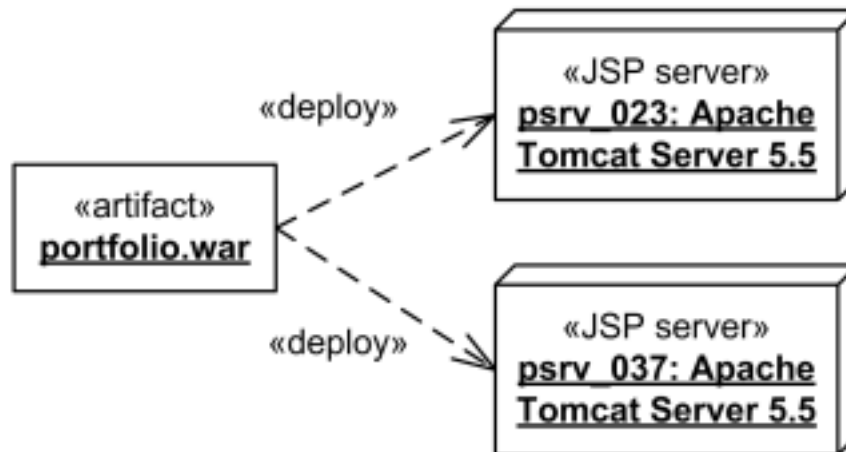
Deployment

- È una dependency relationship che descrive l'allocazione (distribuzione, deployment) di un artefatto su una destinazione di distribuzione (deployment target).
- Può essere rappresentata come una dependency che va dall'artefatto (**supplier**) verso il deployment target (**client**) ed è stereotipata con il label «deploy».



Deployment

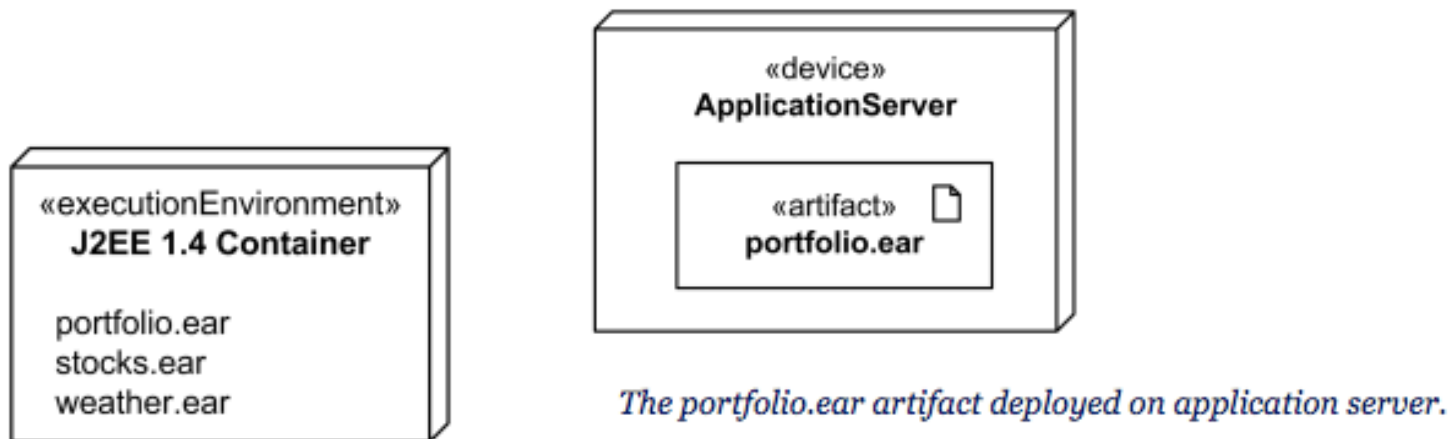
- A "instance level", istanze di artefatti possono essere allocati su istanze di deployment target.



*J2EE web application archive portfolio.war deployed
on two instances of Apache Tomcat JSP server - psrv_023 and psrv_037.*

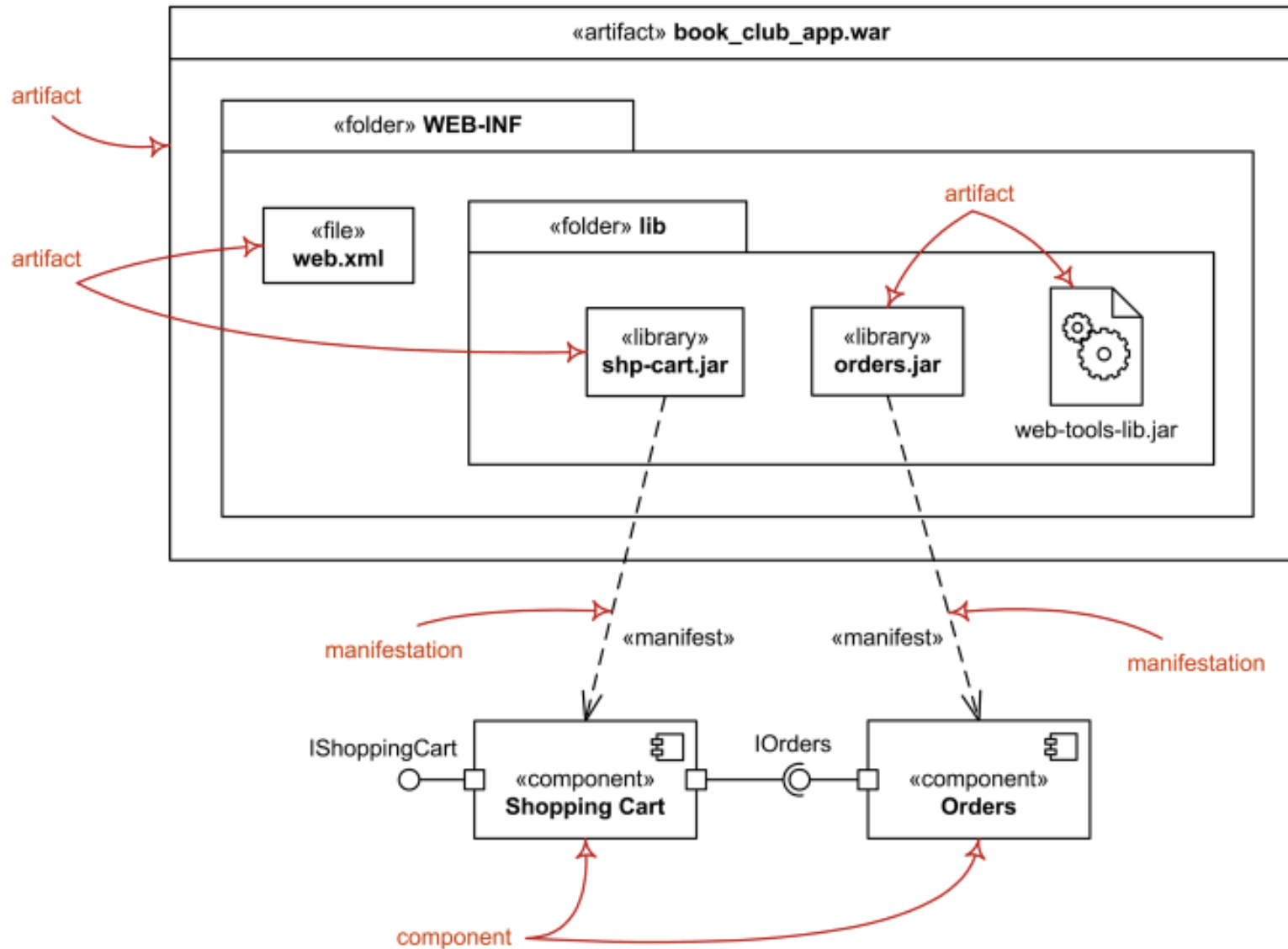
Deployment

- Per modellare complessi diagrammi di deployment è possibile utilizzare una rappresentazione di contenimento.

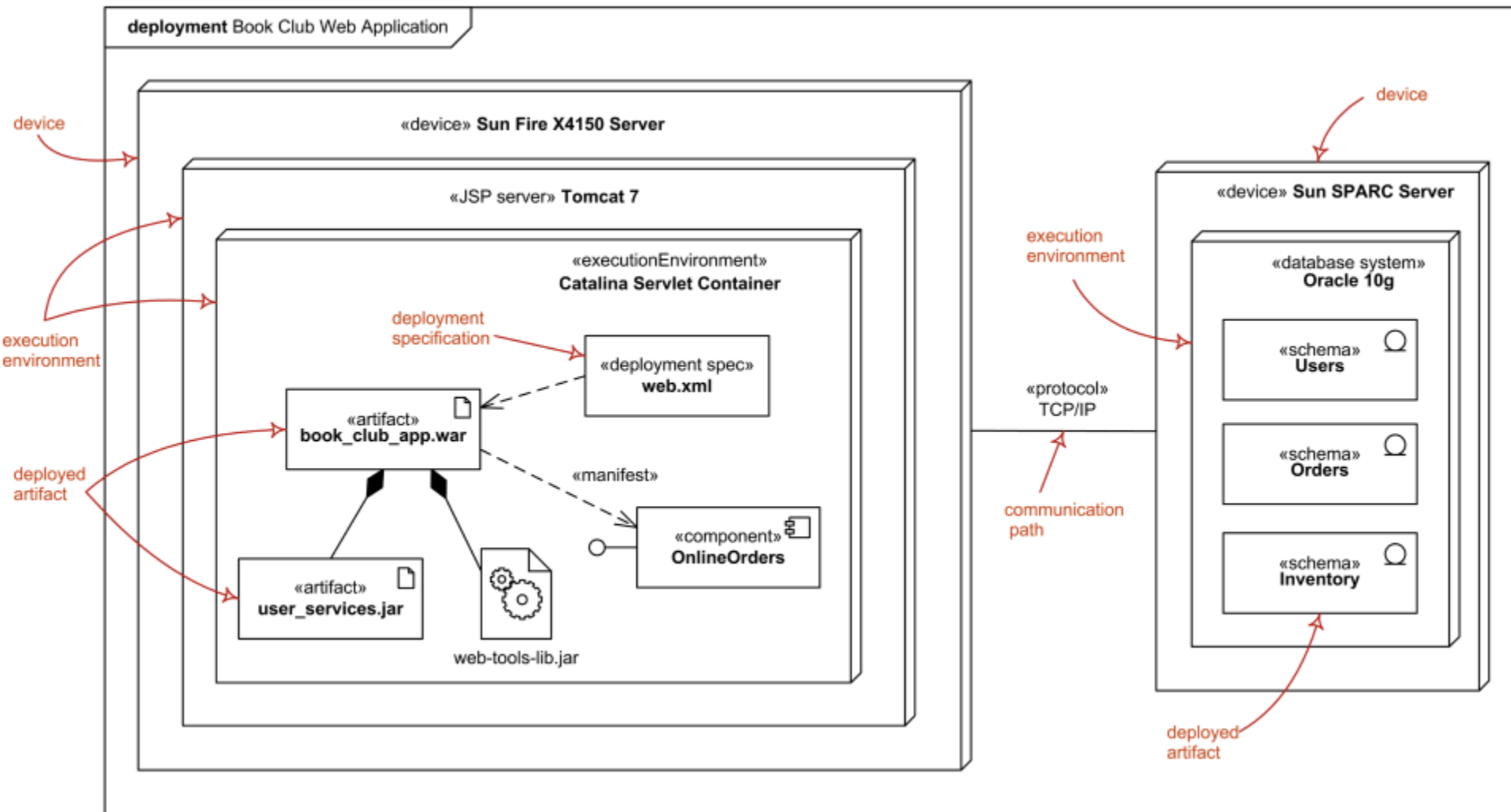


The portfolio.ear, stocks.ear, weather.ear artifacts deployed in J2EE 1.4 container.

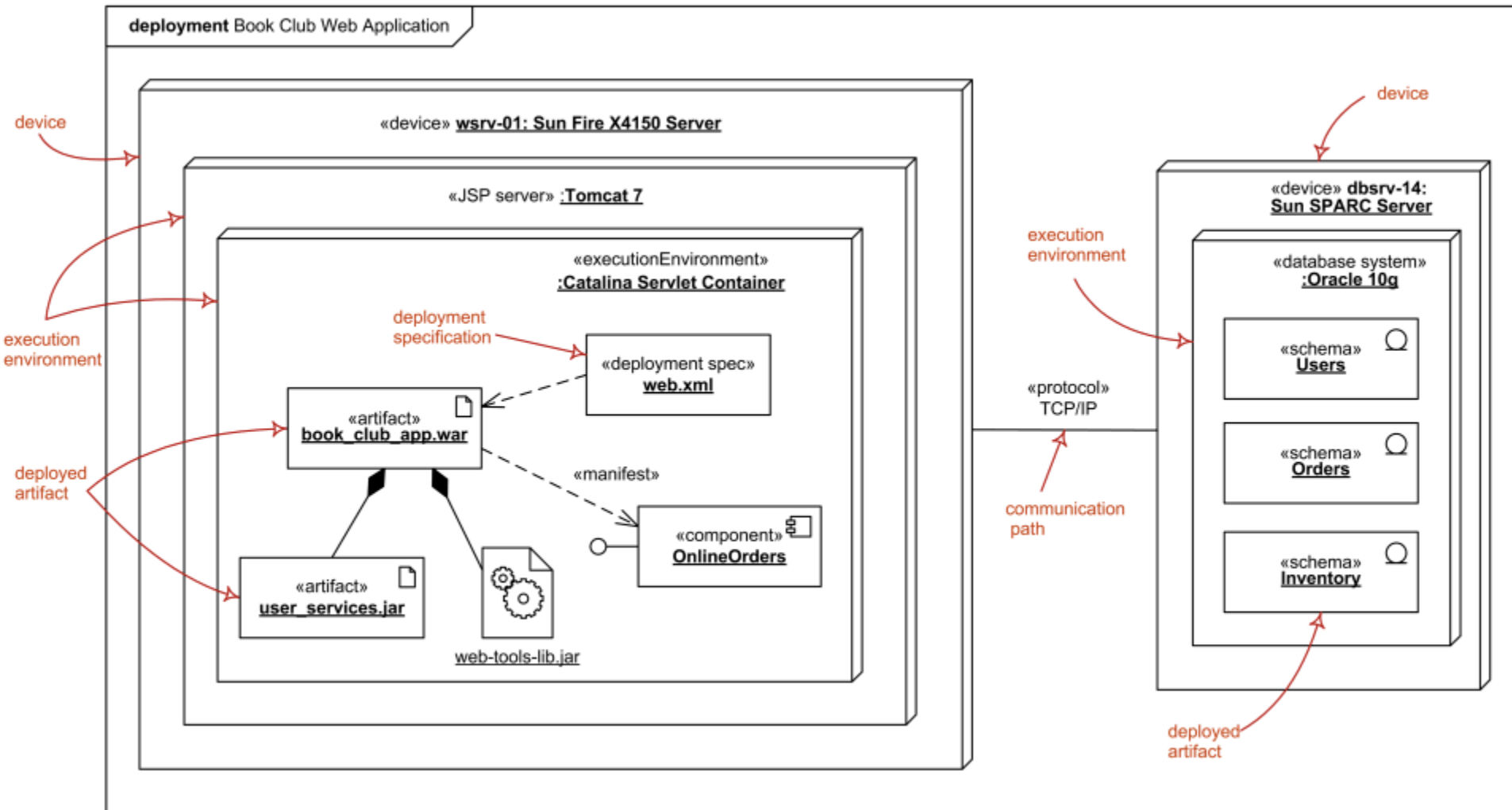
Manifestation of Components by Artifacts



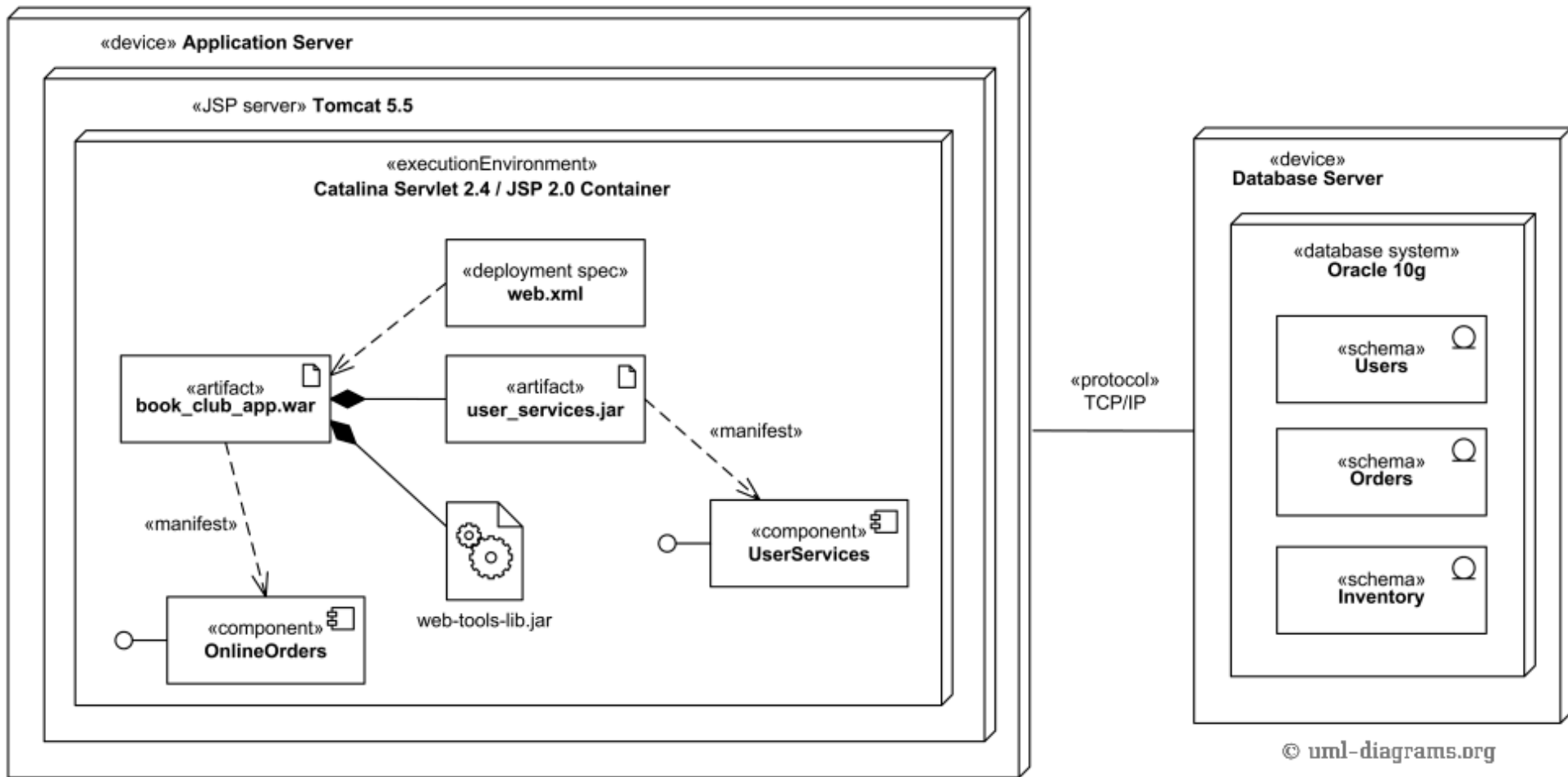
Specification Level Deployment Diagram



Instance Level Deployment Diagram

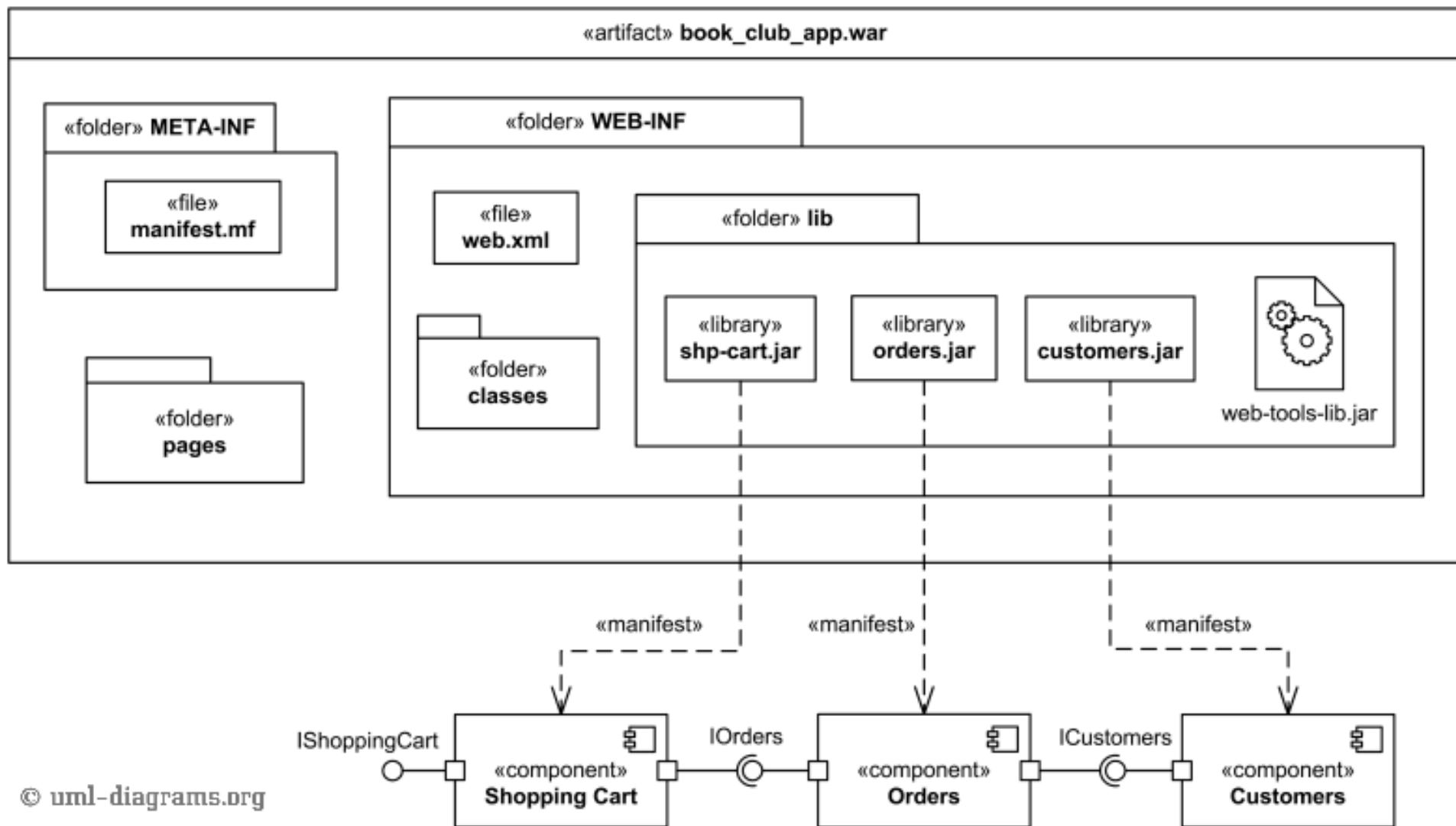


Esempio Web Application



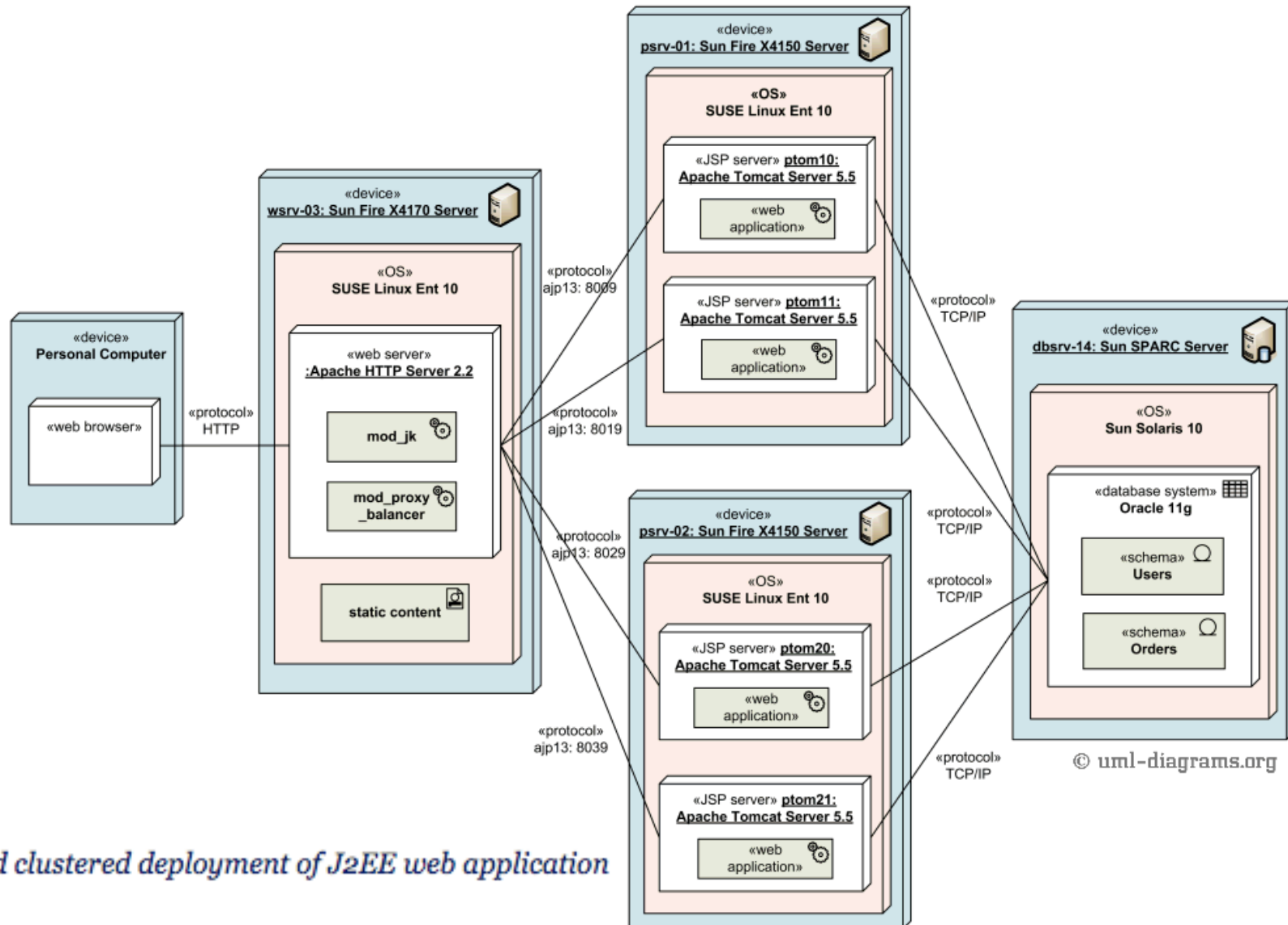
An example of deployment diagram for J2EE web application.

Esempio Web Application



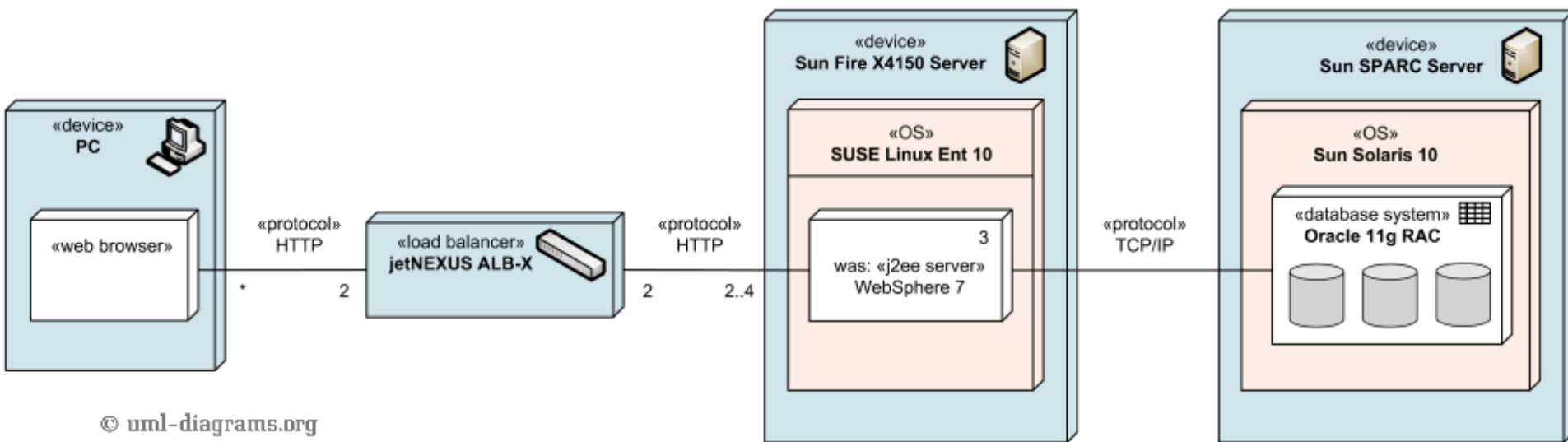
Example of manifestation diagram for web application.

Esempio Web Application Clusters



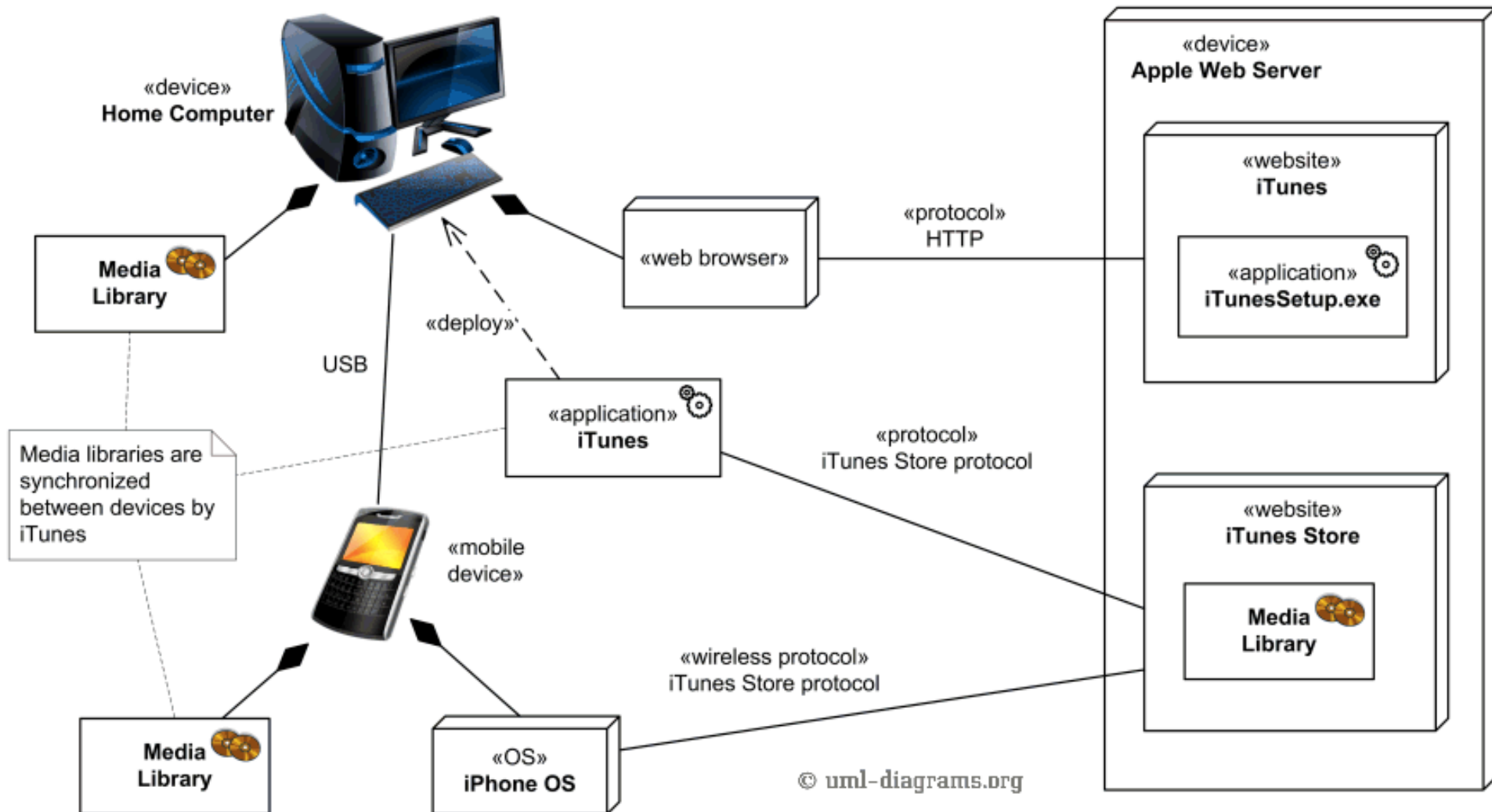
Load balanced and clustered deployment of J2EE web application

Esempio J2EE Web Servers Load Balancing



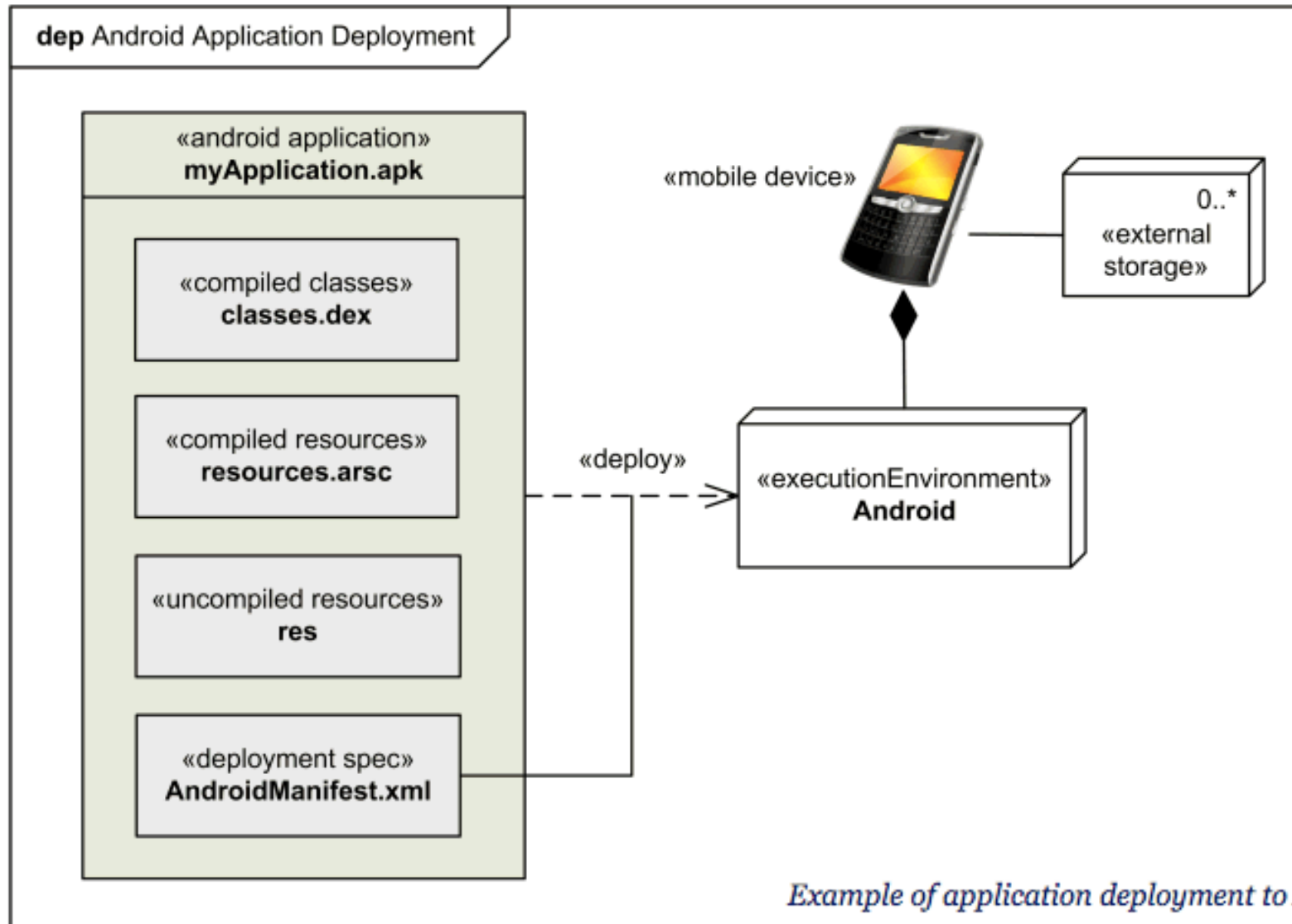
An example of UML deployment diagram with hardware load balancing of J2EE servers.

Esempio Apple iTunes

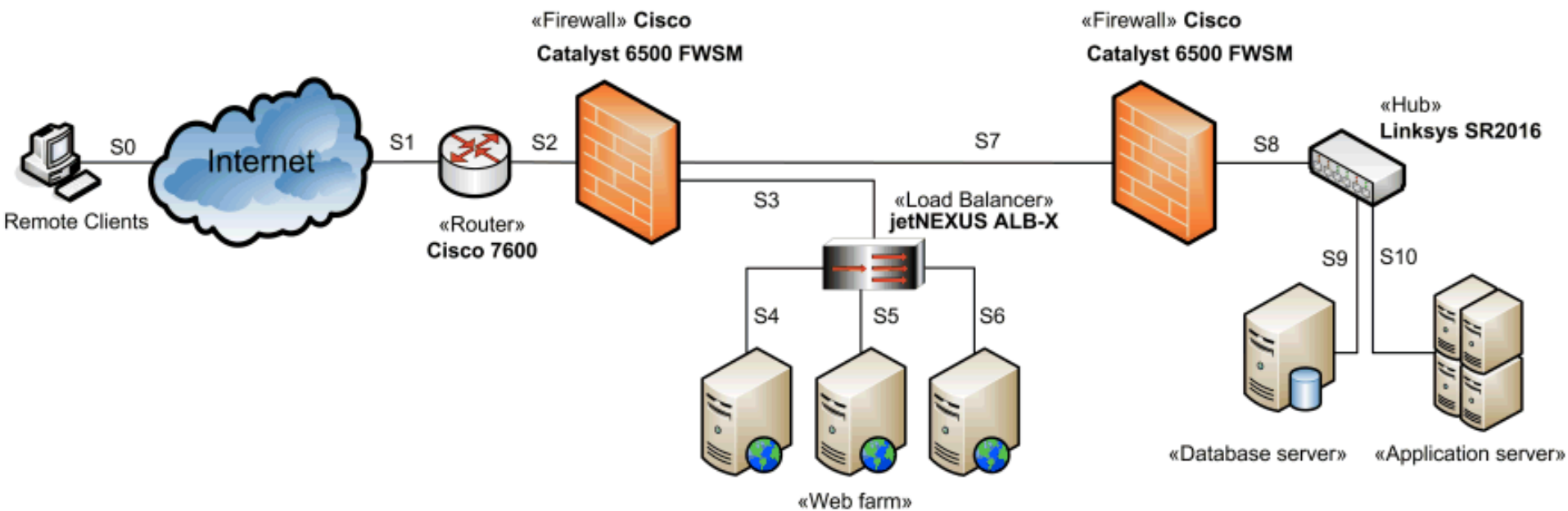


An example of UML deployment diagram for Apple iTunes

Esempio Android Application

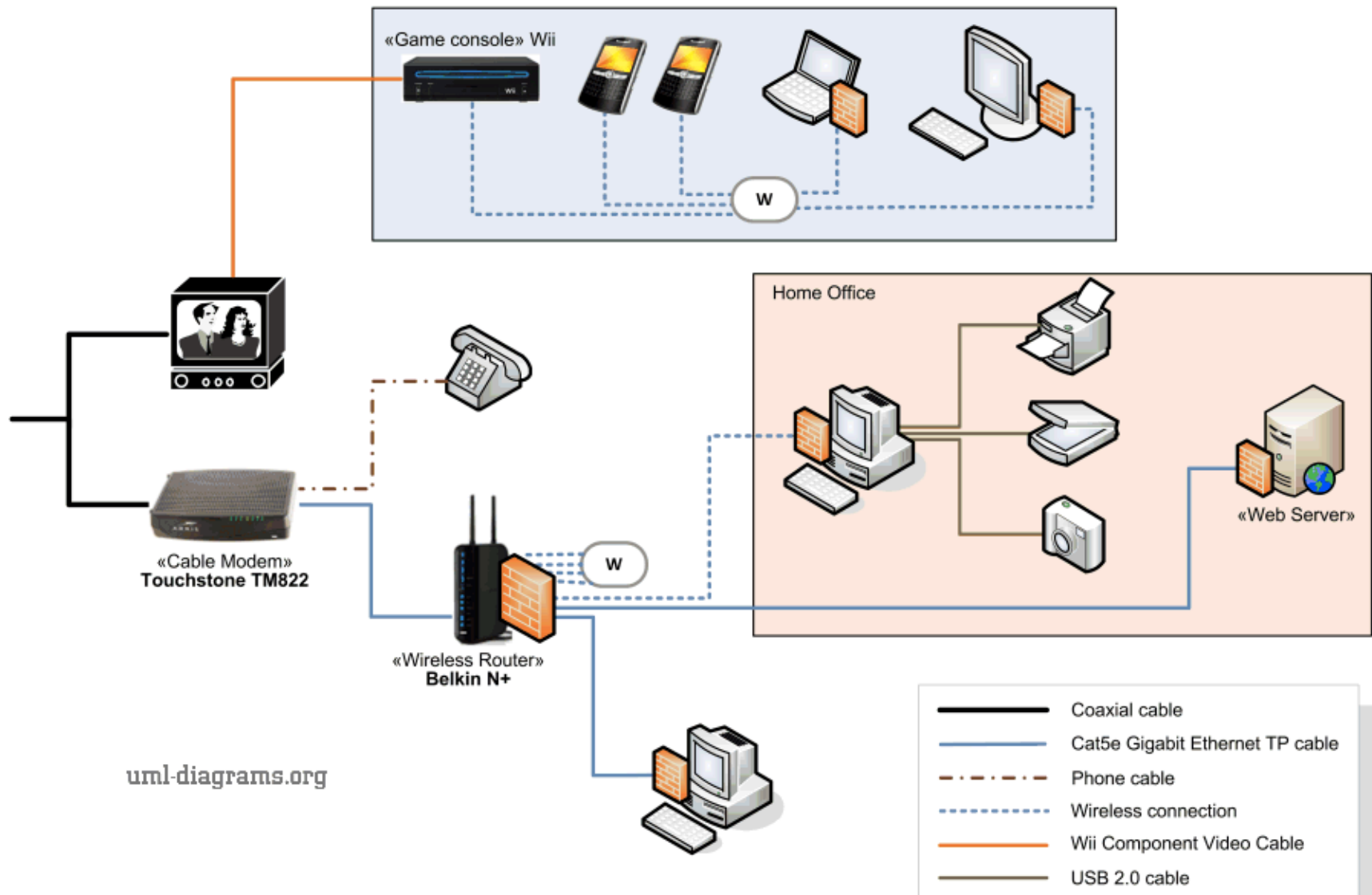


Esempio Web Application



An example of networking diagram for web application with two firewall DMZ configuration.

Esempio Home Network



uml-diagrams.org

Example of home networking diagram - cable modem, wireless router, various computers and devices.