Università degli Studi di Napoli "Federico II"

Ingegneria del Software

a.a. 2013/14

Lezione 11: Attività System Design

Panoramica sulle attività di System Design

Attività

- Mappare I sottosistemi su Processori e Componenti
- Identificare e gestire la persistenza dei dati
- Provvedere il controllo d'accesso
- Progettare il flusso del controllo globale
- Identificare Servizi
- Identificare le condizioni ai confini

Mappare I sottosistemi su Processori e Componenti

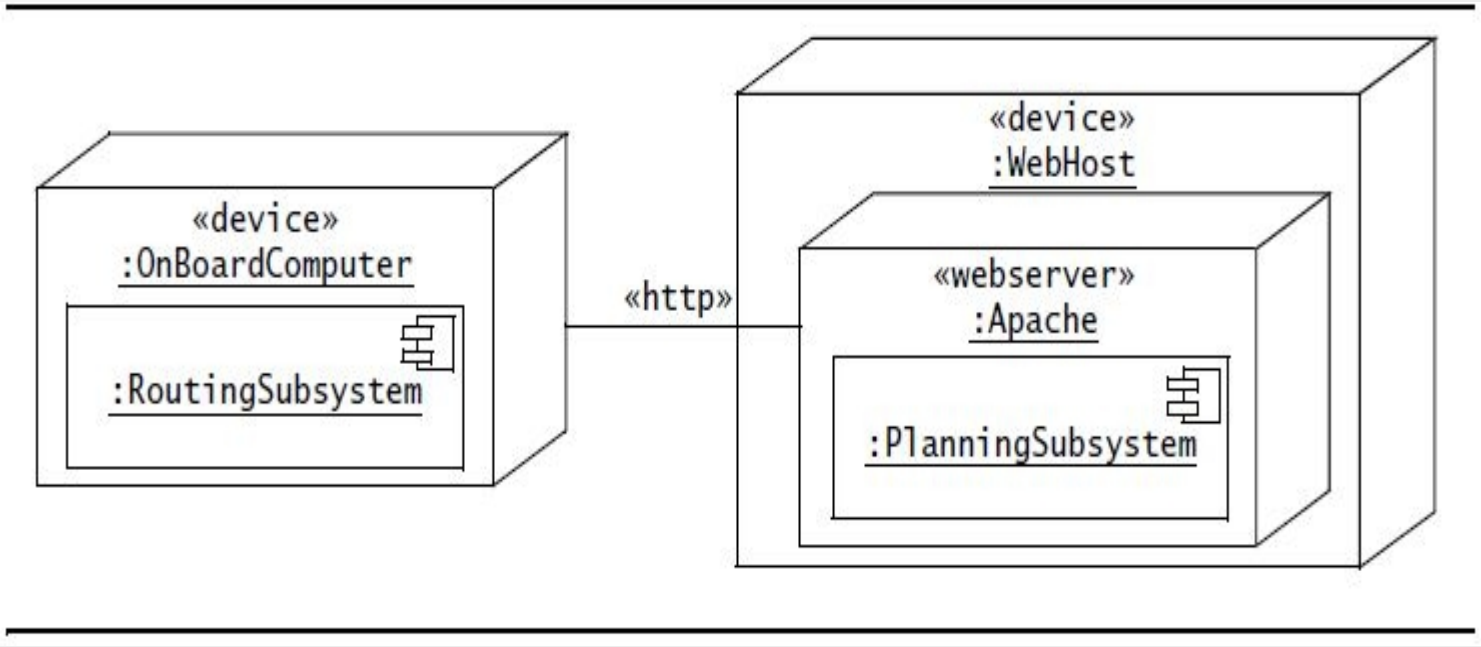
- L'uso di processori multipli e di infrastutture di comunicazione può essere reso necessario per vincoli di performance
  - In questo caso va stabilita la mappatura tra sottosistemi e processori
  - Va progettata l'infrastruttura per la comunicazione tra I sottosistemi.
- Selezione di una “virtual machine” su cui il sistema dovrebbe essere sviluppato
  - Sistema operativo
  - Ogni componente software necessario (DBMS, Package di comunicazione, moduli off the shelf)
- I processori sono indicati come 'nodi' (nodi di un Deployment Diagram in UML).

Mappare I sottosistemi su Processori e Componenti (2)

- Dopo la selezione della configurazione HW e della macchina virtuale I sottosistemi vengono mappati sui nodi
- Conseguenza alla mappatura può essere quella di dover integrare il sistema con nuovi sottosistemi dedicati alla comunicazione e al trasferimento dati tra I diversi nodi
- Mappare I sottosistemi sui nodi permette di distribuire funzionalità e capacità computazionale dove è maggiormente richiesta sollevando tuttavia problemi ovvi di
  - Memorizzazione distribuita
  - Trasferimento dati tra I sottosistemi
  - Dupplicazioni dati tra I sottosistemi
  - Sincronizzazione dati tra I sottosistemi

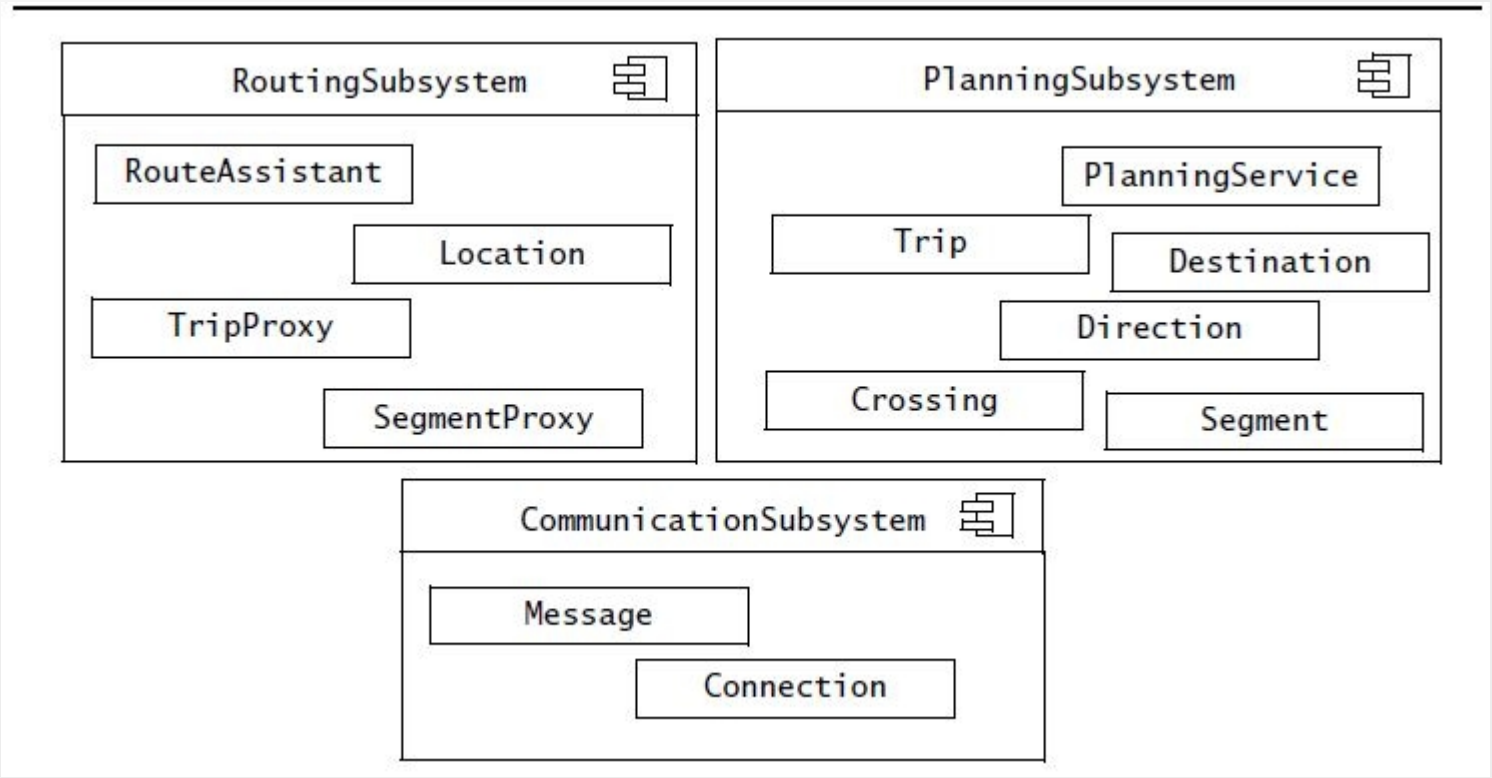
Esempio di mapping sottosistema nodi (deployment diagram)

- Mappa di un sistema di navigazione assistita (navigatore di bordo e WebHost)



Esempio di mapping sottosistema nodi (deployment diagram)

- Individuazione di un sottosistema di comunicazione
-



Diagrammi di Deployment

- Specificano I sistemi fisici su cui verrà eseguito il sistema software (nodi) e descrive l'assegnazione delle componenti software sull'hardware (nodi).
- Fa corrispondera all'architettura software progettata a una archiettura del sistema fisico.
- Nei sistemi distribuiti modella la distribuzione del software nei nodi fisici.

Diagrammi di Deployment

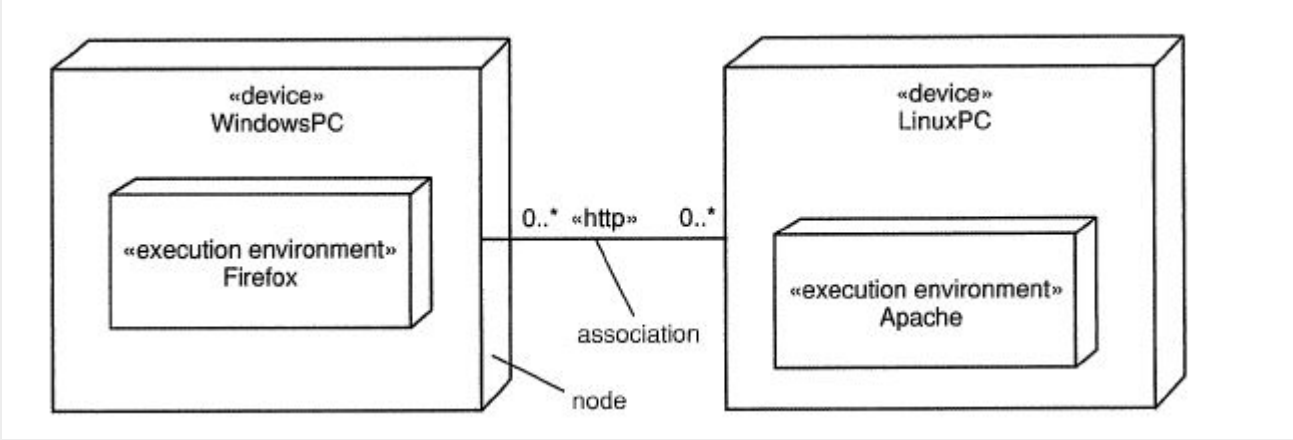
Diagramma di deployment in forma descrittore.

- **Contiene:**
  - Nodi (Tipologie di HW)
  - Relazioni tra nodi
  - Manufatti (Tipo di artefattto SW)
- **Diagramma di deployment in forma istanza.**
- **Contiene:**
  - Istanze di nodi (HW specifico e identificabile)
  - Relazioni tra istanze di nodi
  - Istanze di manufatti (specifica istanza di un artefatto SW)

Diagrammi di Deployment: Nodi

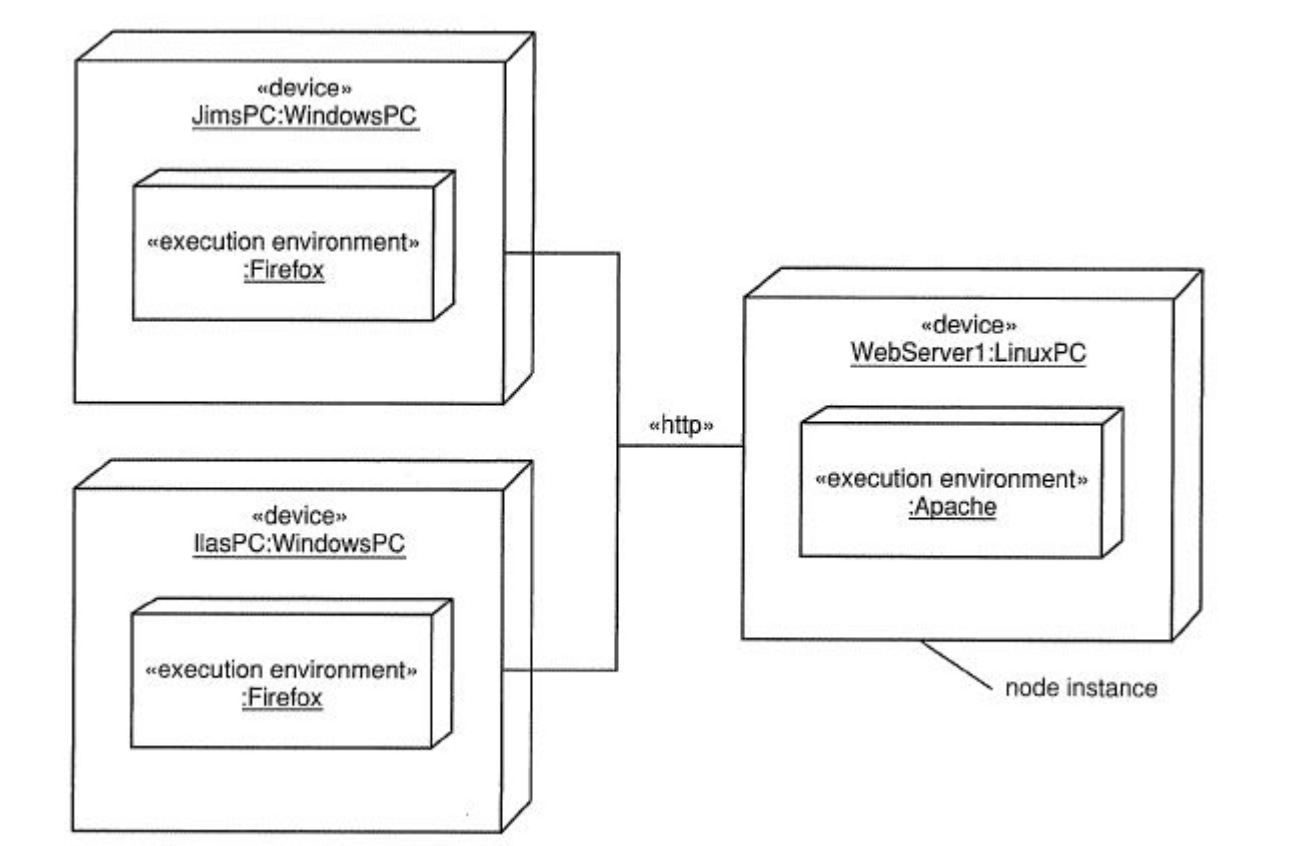
Nodo: un tipo di risorsa computazionale su cui gli artefatti possono essere collocati per la loro esecuzione.

- I nodi possono essere annidati in nodi
- Una associazione tra nodi rappresenta un canale di comunicazione
- **Stereotipi:**
  - **<<device>>** : device fisica (PC, server etc)
  - **<<execution environment>>** : ambiente software di esecuzione (es. Server web Apache, Enterprise JavaBeans, Jboss)
-



Diagrammi di Deployment: Diagramma in forma di istanza

- I diagrammi di deployment in forma di descrittore sono utili per modellare un tipo di architettura fisica
- I diagrammi di deployment in forma di istanza sono utili per modellare il deployment in un particolare sito

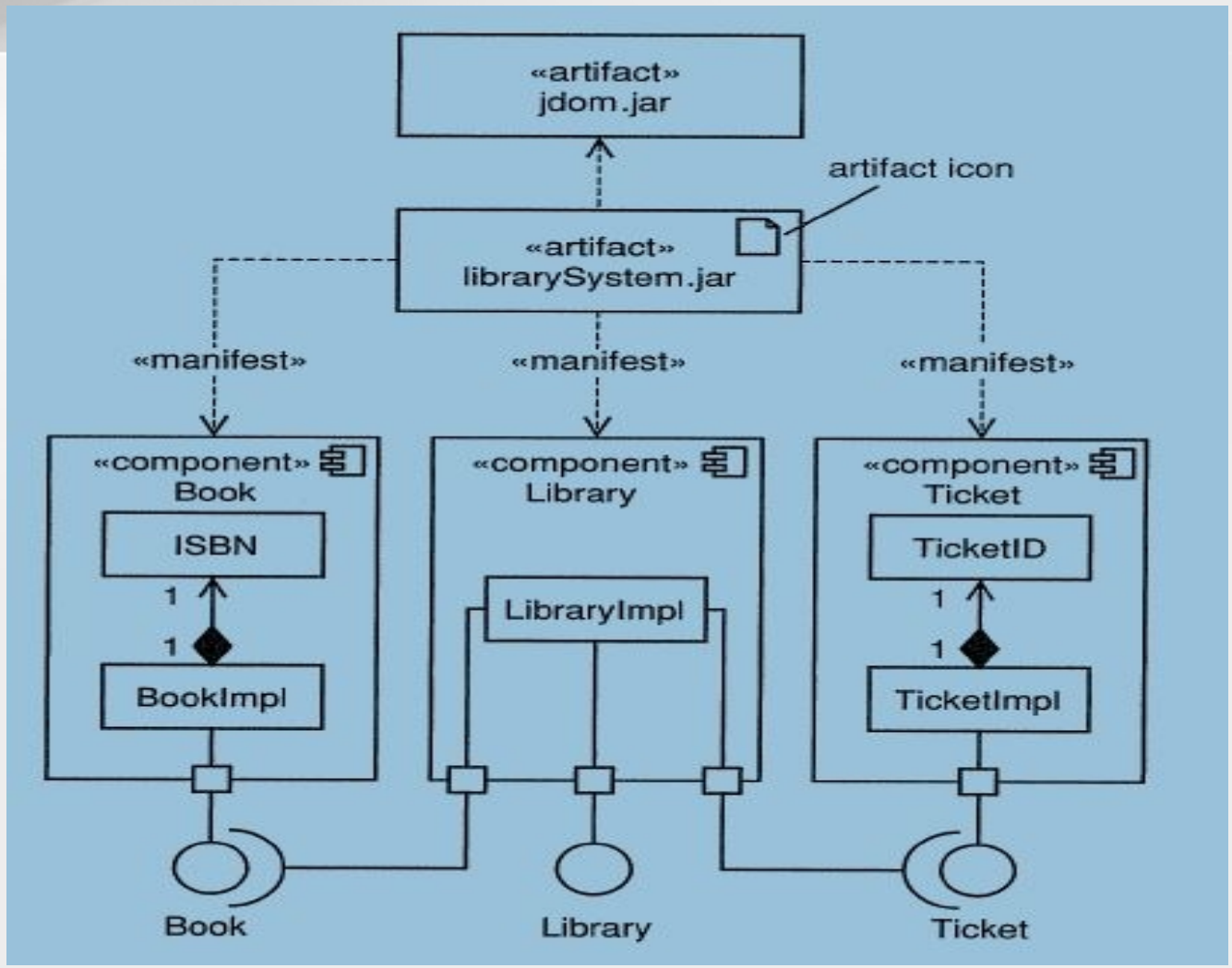


Diagrammi di Deployment: Artefatti

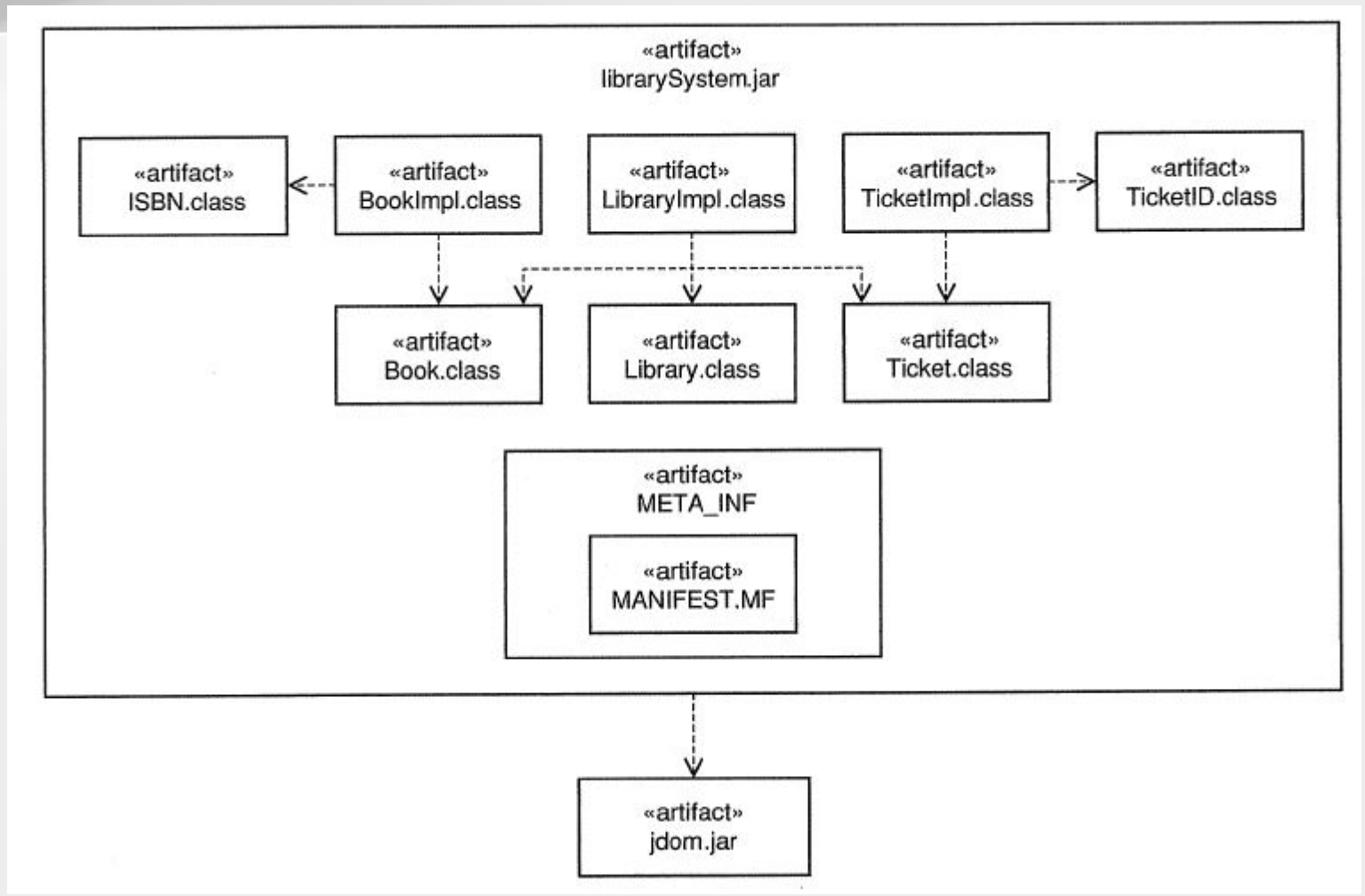
Artefatto: rappresenta una specifica entità oggetto di produzione.

- Esempi:
  - File di codice sorgente (**<<source>>**)
  - File di codice eseguibile (**<<executable>>**)
  - Script (**<<script>>**)
  - Un file fisico (**<<file>>**)
  - Un file con informazioni (**<<document>>**)
  - Una libreria (<<**library**>>)
- Gli artefatti sono rappresentati con box stereotipate <<artifact>>
- Possono essere espresse dipendenze tra gli artefatti.

Diagrammi di Deployment: Artefatti (1)



Diagrammi di Deployment: Artefatti (2)



Identificazione e memorizzazione dei dati persistenti

- **Identificare quali dati devono essere persistenti**
  - Le entity individuate in analisi sono buoni candidati
  - Non tutte le entity sono necessariamente persistenti
  - Non solo le entity possono essere coinvolte, **qualsiasi oggetto che debba sopravvivere alla conclusione di una sessione di uso del sistema.**
- **Stabilire una strategia di memorizzazione**
  - **Flat file** (astrazioni per la memorizzazione fornite direttamente dal sistema operativo).
  - **Database relazionali** (scalabili e ideali per ampie collezioni di dati strutturati; inefficienti per collezioni ridotte o per dati non strutturati; Complessa la memorizzazione di oggetti)
  - **Database ad oggetti** (dati memorizzati direttamente come oggetti ed associazioni; meno efficienti le interrogazioni rispetto ai relazionali)

Scelta della strategia di persistenza

- **Flat file**
  - Dati voluminosi poco strutturati (immagini, documenti full text etc)
  - Dati temporanei (ad es. Core file)
  - A basso contenuto informativo (file di archiviazione, history log …)
- **Database (relazionale o a oggetti)**
  - Accesso concorrente
  - Accesso ai campi strutturati
- **Relazionale**
  - Query complesse sugli attributi
  - Collezioni di dati ampie
- **Oggetti**
  - Uso estensivo di associazioni per recuperare I dati
  - Collezioni di media dimensione
  - Associazioni irregolari tra oggetti.

Controllo degli accessi

- Identificare quali oggetti possno essere condivisibili dagli attori e le autenticazioni al sistema.
- E' possibile modellare l'accesso alle classi e alle operazioni mediante **matrici di accesso**
- **Global access table.**
  - La matrice ha una struttura del tipo **(actor, entity, operation).**
  - La presenza/assenza di una tupla per un attore equivale a diritto/mancanza di diritto all'accesso.
  - La memorizzazione della tabella richiede molto spazio.
-

Controllo degli accessi (2)

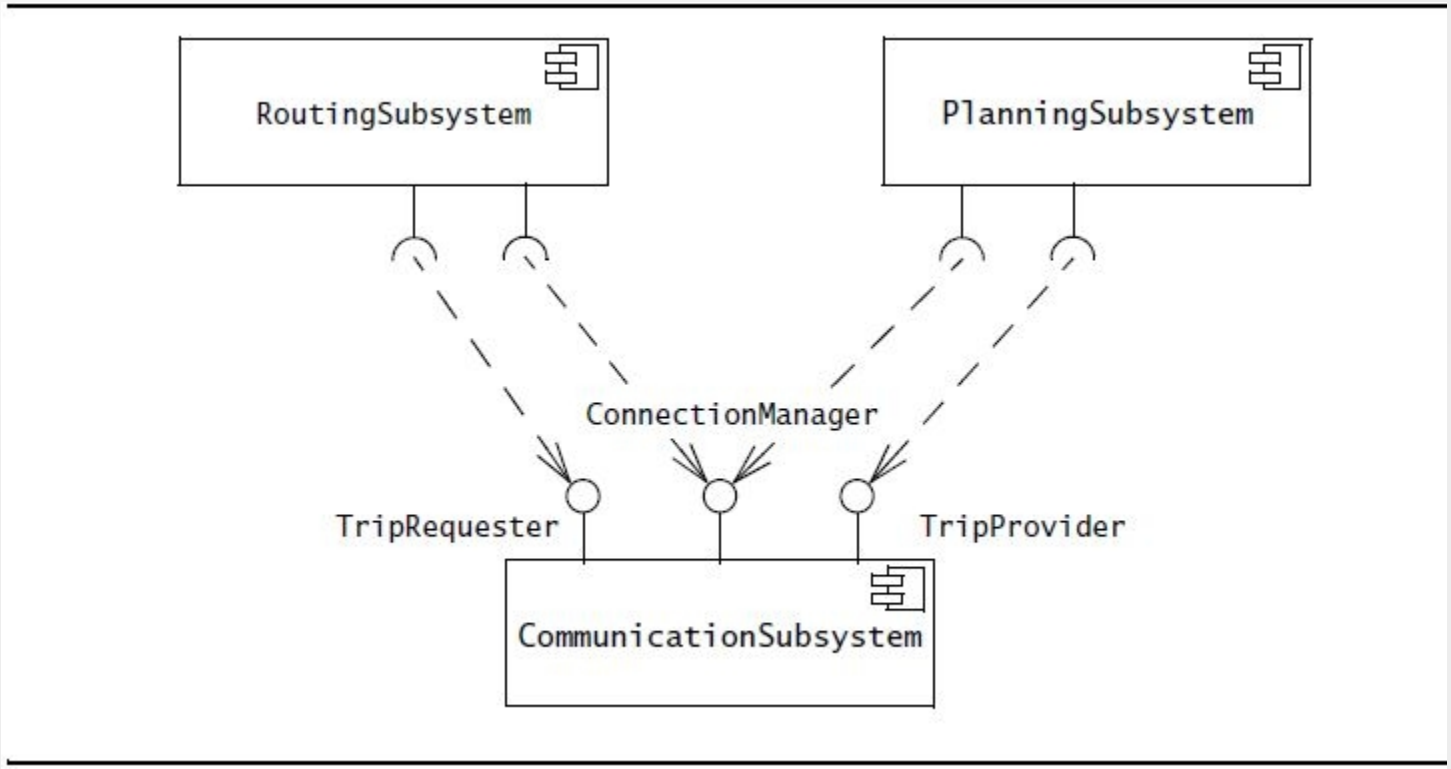
- **Access control list**
  - Si associa ad ogni classe a cui è possibile l'accesso una lista di coppie di autorizzazione **(actor,operation)**
  - L'attore che fa accesso all'istanza della classe può accedere all'operazione solo se presente nella lista associata.
  - Semplifica la risposta al quesito 'Chi può accedere questo oggetto?'
- **Capability.**
  - Una lista di coppie (class,operation) viene associata a ciascun attore a rappresentare le sue capacità di accesso.
  - Semplifica la risposta al quesito 'A quali oggetti può accedere l'attore?'
- Quando il numero di attori e di oggetti protetti diviene troppo ampio si possono usare sistemi alternativi basati su regole come rappresentazione compatta (intensionale) delle matrici di accesso globale.

Identificazione dei servizi

- Presuppone che la suddivisione in sottosistemi sia stata attuata e che vi sia una ragionevole associazione delle responsabilità ai singoli sottosistemi
- **Individuazione dei servizi per ogni sottosistema**
- **Definizione di una interfaccia per ogni servizio**
  - In questa fase si dà un nome all'interfaccia
  - Nella fase di object design per ogni interfaccia si definiscono più precisamente operazioni, parametri e vincoli per ciascuna interfaccia
  - Le interfacce sono rappresentate come lollipop nei component diagram.

Esempio di identificazione dei servizi

.



Identificazione delle condizioni ai limiti

- Usualmente gli use case ai limiti non sono trattati durante la fase di analisi dei requisti
- Decidere come il sistema è
  - acceso (start),
  - inizializzato,
  - spento (shut down)
- Decidere il comportamento in caso dei principali crash
  - In caso di corruzione dei dati
  - Di disfunzioni nella comunicazione di rete
- **Vanno a questo punto considerati use case per ogni sottosistema e e per ogni oggetto persistente.**

Use case per casi limite

- **Configuration**: per ogni oggetto persistente si esamina in che use case sono creati o distrutti. In assenza di creazione/distruzione si crea uno use-case invovato da un system administrator
- **Start up and shutdown.** Per ogni componente i aggiungono tre use case per lo start up, shut down e configurazione della componente.
- **Exception handling.** Per ogni fallimento di una componente si decide come il sistema debba reagire al fallimento. Si scrive dunque per ogni fallimento una estensione di uno use-case esistente.
- **Sorgenti di eccezioni:**
  - HW failure
  - Cambiamenti nell'ambiente di operatività
  - SW fault

Progettazione del flusso di controllo globale

- Durante la fase di analisi il flusso del controllo (sequenzializzazione delle azioni nel sistema) poiché (irrealisticamente) si assume che tutti gli oggetti eseguono simultaneamente eseguendo le operazioni quando necessitano.
- Possibili meccanismi di controllo
  - **Procedure driven**
    - Legato al tradizionale flusso del controllo dei linguaggi procedurali;
    - Più difficoltoso gestire il flusso dei controlli dei paradigmi ad oggetti

Progettazione del flusso di controllo globale

- **Event-driven**
  - Un loop principale aspetta per la ricevione di un evento che viene comunicato agli oggetti appropriati alla gestione
  - Permette di centralizzare la gestione degli eventi
  - Rende difficile la gestione di sequenze multistep
-

```
Iterator subscribers, eventStream;
Subscriber subscriber;
Event event;
EventStream eventStream;
/* ... */
while (eventStream.hasNext()) {
    event = eventStream.next();
    subscribers = dispatchInfo.getSubscribers(event);
    while (subscribers.hasNext()) {
        subscriber = subscribers.next()) {
        subscriber.process(event);
    }
}
/* ... */
```

Progettazione del flusso di controllo globale

- **Threads**
  - Variazione concorrente di un procedure driven
  - Può creare un numero arbitrario di tread ciascuno in corrispondenza con uno specifico evento

```
Thread thread;
Event event;
EventHandler eventHandler;
boolean done;
/* ...*/
while (!done) {
    event = eventStream.getNextEvent();
    eventHandler = new EventHandler(event)
    thread = new Thread(eventHandler);
    thread.start();
}
/* ...*/
```

Progettazione del flusso di controllo globale

- **Una volta scelta la politica di gestione del flusso di controllo**
  - Si creano oggetti di controllo addetti alla gestione del flusso di controllo
  - Gli oggetti di controllo memorizzano la gestione degli eventi
  - Gli oggetti di controllo producono la giusta sequenza di chiamate agli oggetti interessati agli eventi esterni
- La localizzazione della gestione del controllo in un modulo rende più chiara la comprensione de codice e più semplici I cambiamenti eventuali delle politiche di gestione del controllo

Politiche di Controllo: Approfondimento

Gestione di eventi asincroni

- Per sapere se un’entità esterna ha fatto un’azione o ha subito un cambiamento di stato, si hanno due opzioni:
  - Polling: si accerta periodicamente e ripetutamente se è occorso un evento
  - Event-based: si attende una notifica dall'entità interessata all'evento.

- Nella progettazione del sistema si hanno a disposizione le stesse due strategie per la definizione delle politiche di controllo.

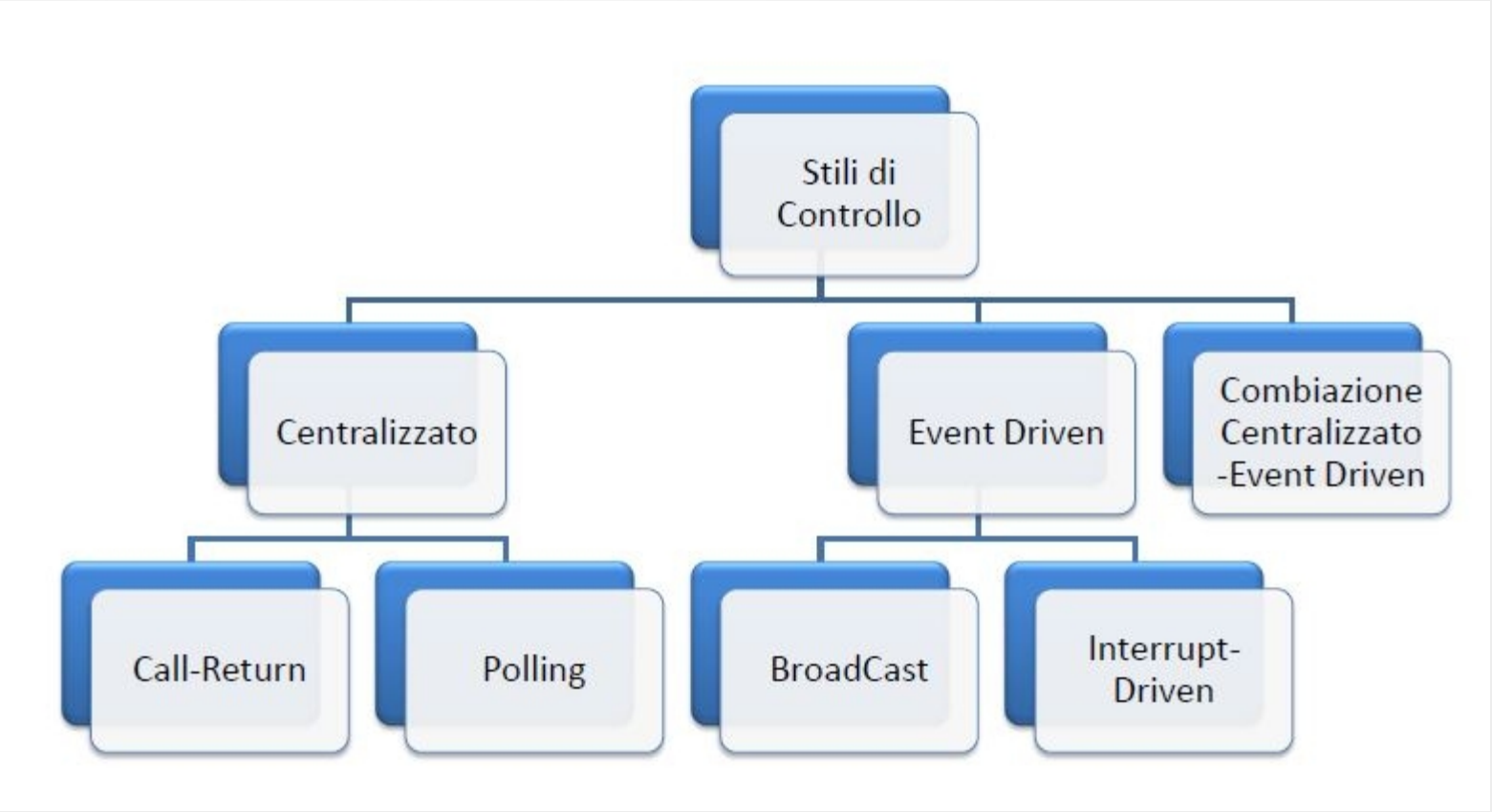
Stili di controllo

- La maggior parte dei sistemi software sono costituiti da Sottosistemi/ Classi / Metodi che reagiscono a richieste esterne di servizio:
  - Eventi da tastiera/mouse
  - Segnali da sensori
  - Invocazioni da altri moduli
  - …
- Obiettivo della progettazione di architetture riguardo gli stili di controllo è la definizione delle politiche per gestire (smistare) tali input verso i moduli

Stili di controllo

- Politiche di controllo:
  - Controllo centralizzato
    - Un unico sottosistema è responsabile di attivare e interrompere gli altri
  - Controllo basato su eventi
    - Ciascun sottosistema può rispondere a eventi esterni, generati da altri sottosistemi o dall'ambiente
  - Combinazione dei due precedenti
- In linea di principio, tipo di architettura e politica di controllo sono due aspetti complementari

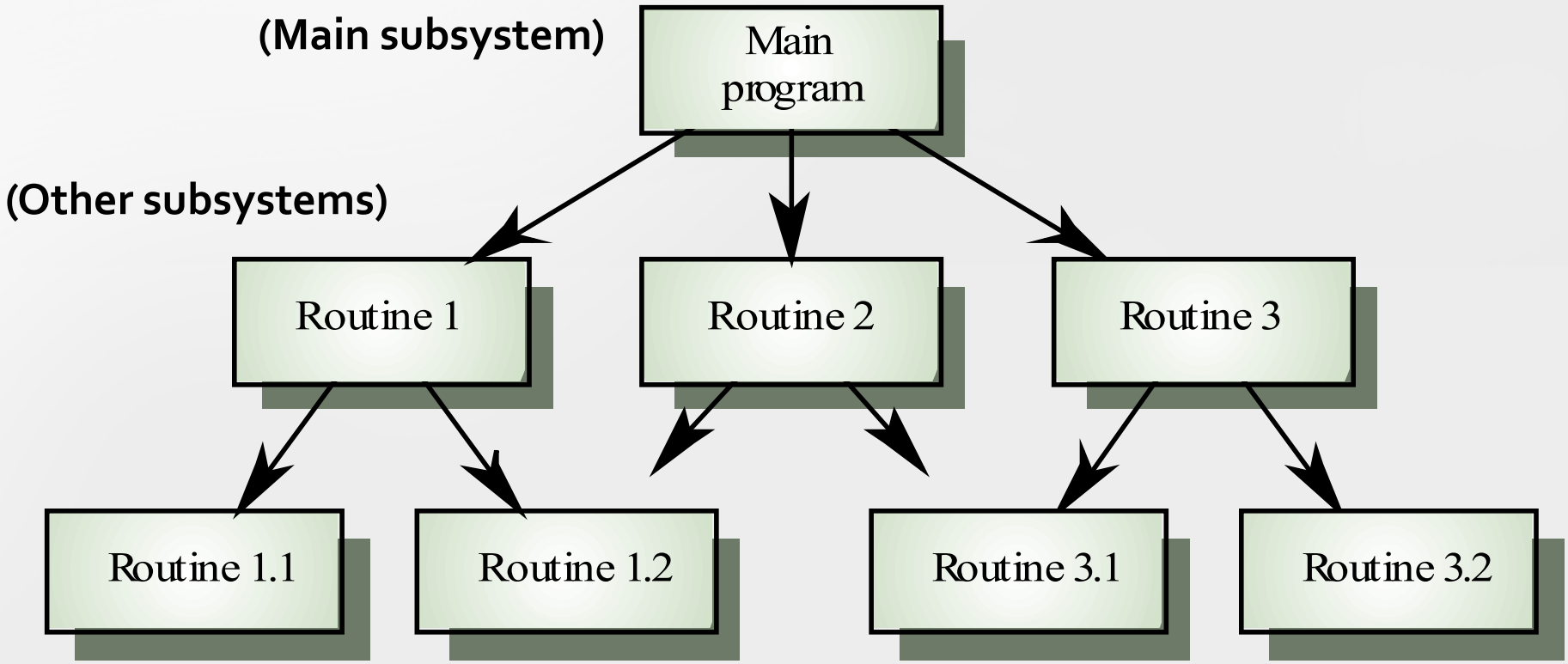
Stili di controllo



Controllo centralizzato

- Modello Call-return
  - Applicabile a sistemi sequenziali.
  - Il controllo si muove come nell'albero delle chiamate di routine nei programmi sequenziali.
    - E' applicabile solo a sistemi con eventi sincroni sequenziali
- Modello a Polling
  - Applicabile a sistemi concorrenti. Il sottosistema controller (o manager) sovrintende il funzionamento di tutto il sistema, prevedendo l'avvio, l'arresto, il coordinamento degli altri sottosistemi.
    - E' applicabile a qualunque tipo di sistema

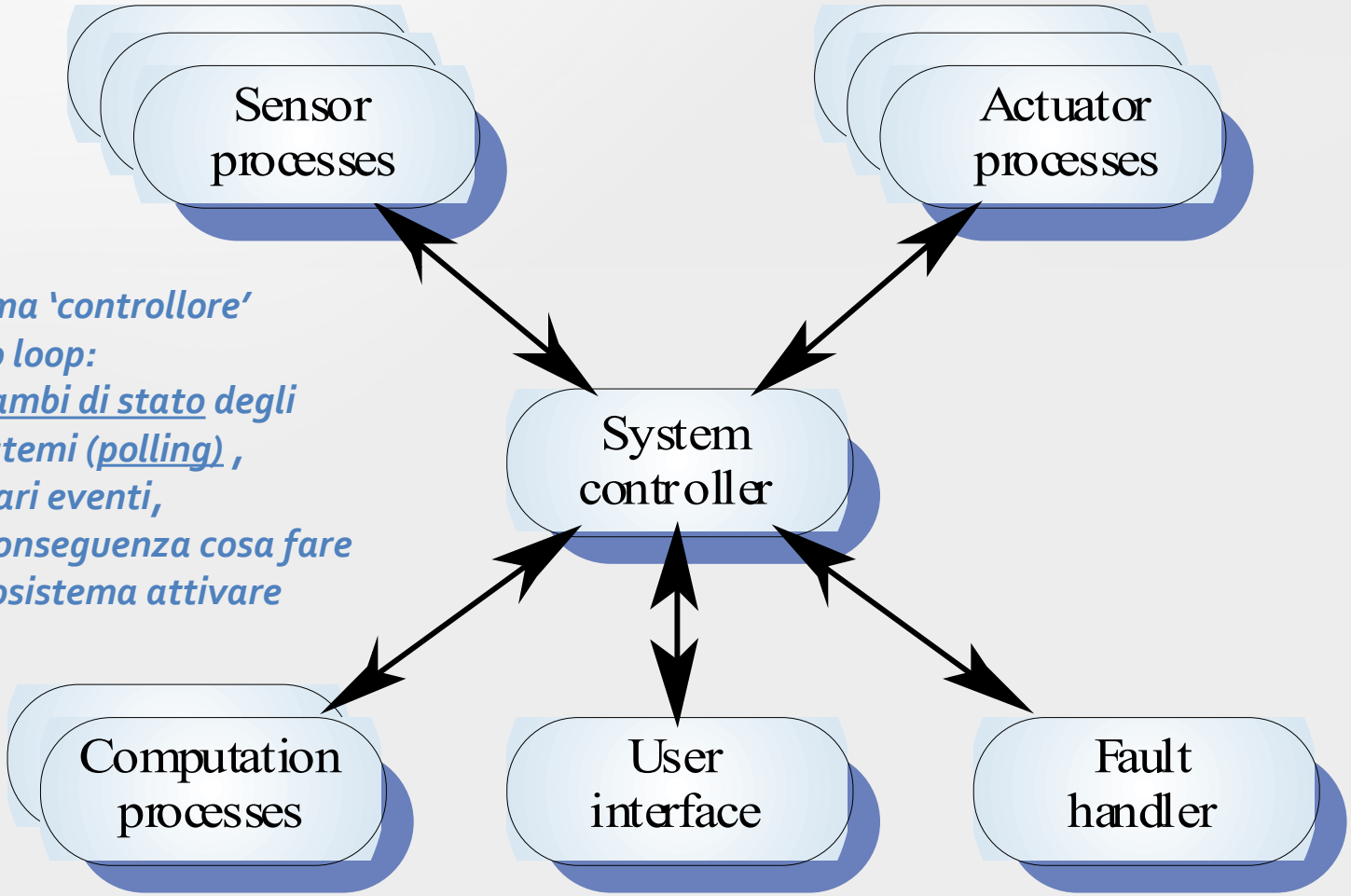
Controllo centralizzato / modello Call-return (sequenziale)



Vantaggio: è facile analizzare il flusso di controllo

Svantaggio: La sequenza di interazioni deve essere definita a priori

Controllo centralizzato / Modello a 'manager' (Polling)



Il sottosistema 'controllore' è in continuo loop: controlla i cambi di stato degli altri sottosistemi (polling) , causati da vari eventi, e decide di conseguenza cosa fare o quale sottosistema attivare

Controllo basato su eventi (sistemi event-driven)

- Nelle precedenti architetture a controllo centralizzato, le decisioni dipendono dalle variabili di stato, osservate su iniziativa del controllore.
- Al contrario, nei sistemi event driven il controllo è pilotato dagli eventi, generati dall'ambiente, o dagli stessi sottosistemi, a tempi imprevedibili (asincroni).
- Modelli a broadcast.
  - Ogni evento rilevato viene trasmesso a tutti i sottosistemi. Ogni sottosistema in grado di trattare l'evento lo fa.
- Modelli interrupt-driven.
  - Usati per sistemi (soft) real time: interrupt esterni sono rilevati da un interrupt handler e passati al sottosistema appropriato per il trattamento.

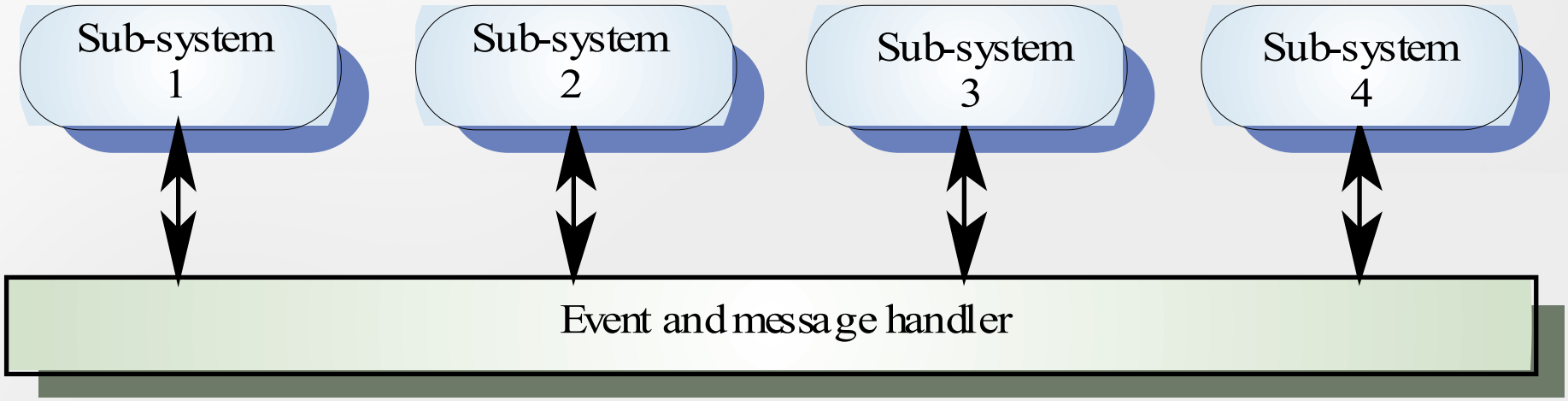
Controllo basato su eventi / modello a broadcast

- I sottosistemi registrano il proprio interesse per determinati eventi.
  - Quando un evento accade (p. es. un sottosistema segnala che ci sono dati pronti per essere elaborati), il gestore di eventi (Event and Message Handler - EMH) lo rileva, controlla il registro, e lo trasferisce, assieme al controllo, ai sottosistemi interessati.

- La politica di controllo non è racchiusa in EMH, come nel controllore centralizzato: i sottosistemi decidono autonomamente quali sono gli eventi di loro interesse.

- EMH può anche gestire la comunicazione fra sottosistemi.

Modello a broadcast



Modello a broadcast: vantaggi e svantaggi

- Vantaggi
  - L'evoluzione è facilitata: ogni nuovo sottosistema aggiunto deve solo informare EMH sui messaggi di proprio interesse
  - I sottosistemi si attivano a vicenda indirettamente, mandando messaggi a EMH, e non devono conoscere i propri indirizzi: la struttura distribuita è trasparente per i sottosistemi.

- Svantaggi
  - I sottosistemi non sanno se e quando i loro messaggi verranno raccolti e gestiti
  - Possibili conflitti se diversi sottosistemi sono interessati agli stessi eventi.

Controllo basato su eventi / modello interrupt-driven

- Politica usata per sistemi real-time in cui risposte rapide agli eventi sono essenziali
- Idea di base:
  - Esistono diversi tipi di interrupt, e un interrupt-handler definito per ciascuno di essi
  - Ciascuno dei tipi di interrupt è associato a una locazione di memoria, dove è memorizzato l'indirizzo del corrispondente handler.
  - Uno switch hardware rapido provoca l' immediato trasferimento del controllo allo handler opportuno, che a sua volta attiva specifici processi.
- Puo' essere combinato con il modello a manager centralizzato
  - Diversi eventi vengono gestiti con l'una o l'altra tecnica (interrupt e polling), a seconda della loro urgenza.