Gestione delle risorse: priority inheritance e priority ceiling con RT-POSIX

Corso di
Progetto e Sviluppo di
Sistemi in Tempo Reale

Marcello Cinque

RT-POSIX: gestione delle risorse

- Sommario della lezione:
  - Interprocess communication: introduzione
  - Utilizzo di mutex
  - Uso dei protocolli priority inheritance e priority ceiling
- Riferimenti
  - https://pubs.opengroup.org/onlinepubs/009695399/idx/realtime.html

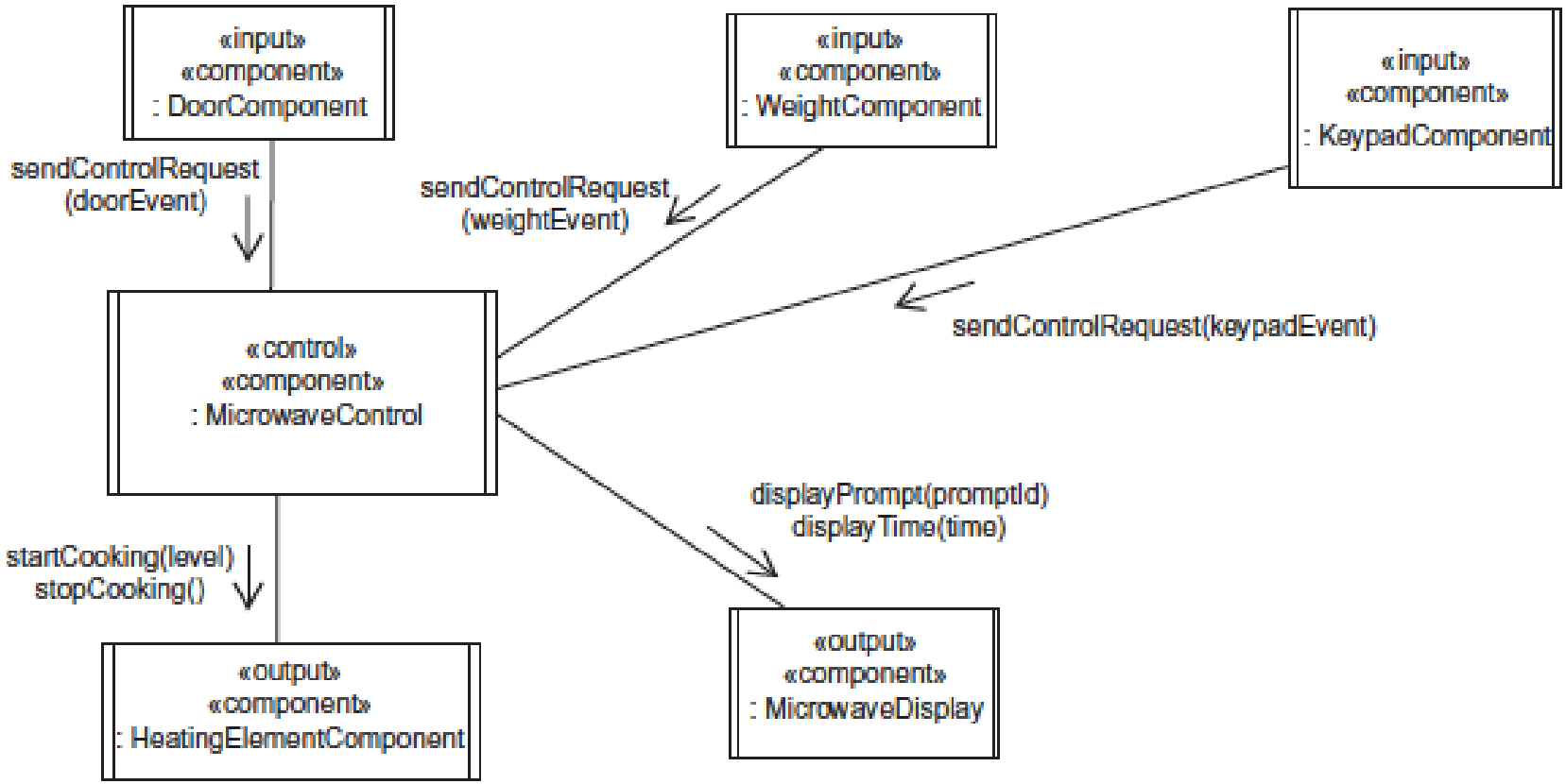
Interprocess communication

- L'interprocess communication (IPC) è l'insieme dei meccanismi offerti dal sistema operativo per consentire la comunicazione tra processi **concorrenti**, in termini di:
  - **Competizione**: per l'accesso a risorse condivise (mutua esclusione)
  - **Cooperazione**: per la realizzazione di un fine comune (comunicazione)

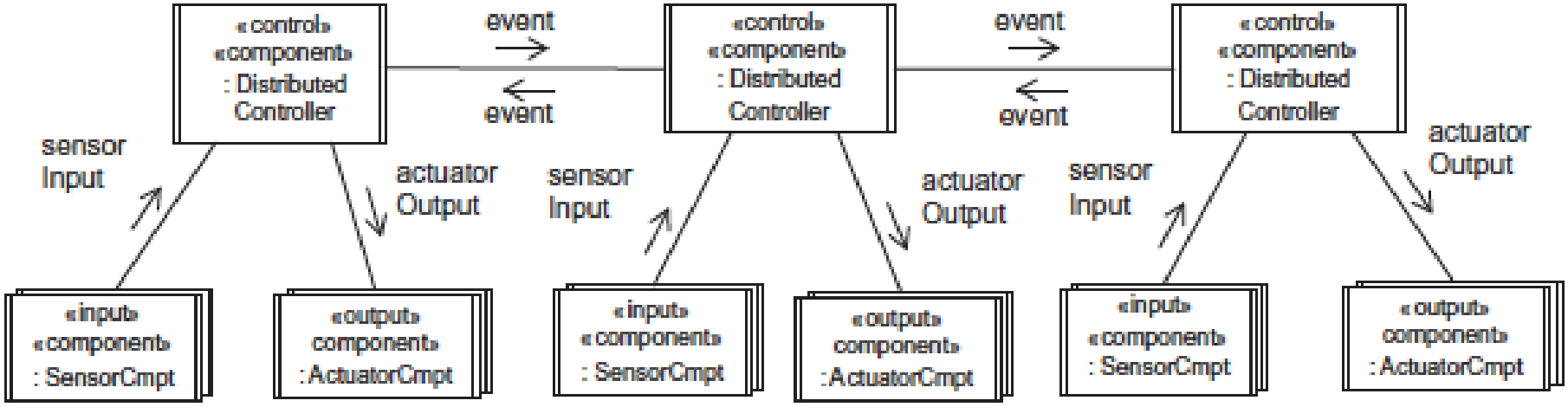
Pattern architetturali di controllo

- La concorrenza è insita nella realizzazione di applicazioni real-time di controllo, quale che sia il pattern architetturale utilizzato
  - Controllo centralizzato
  - Controllo distribuito
  - Controllo gerarchico

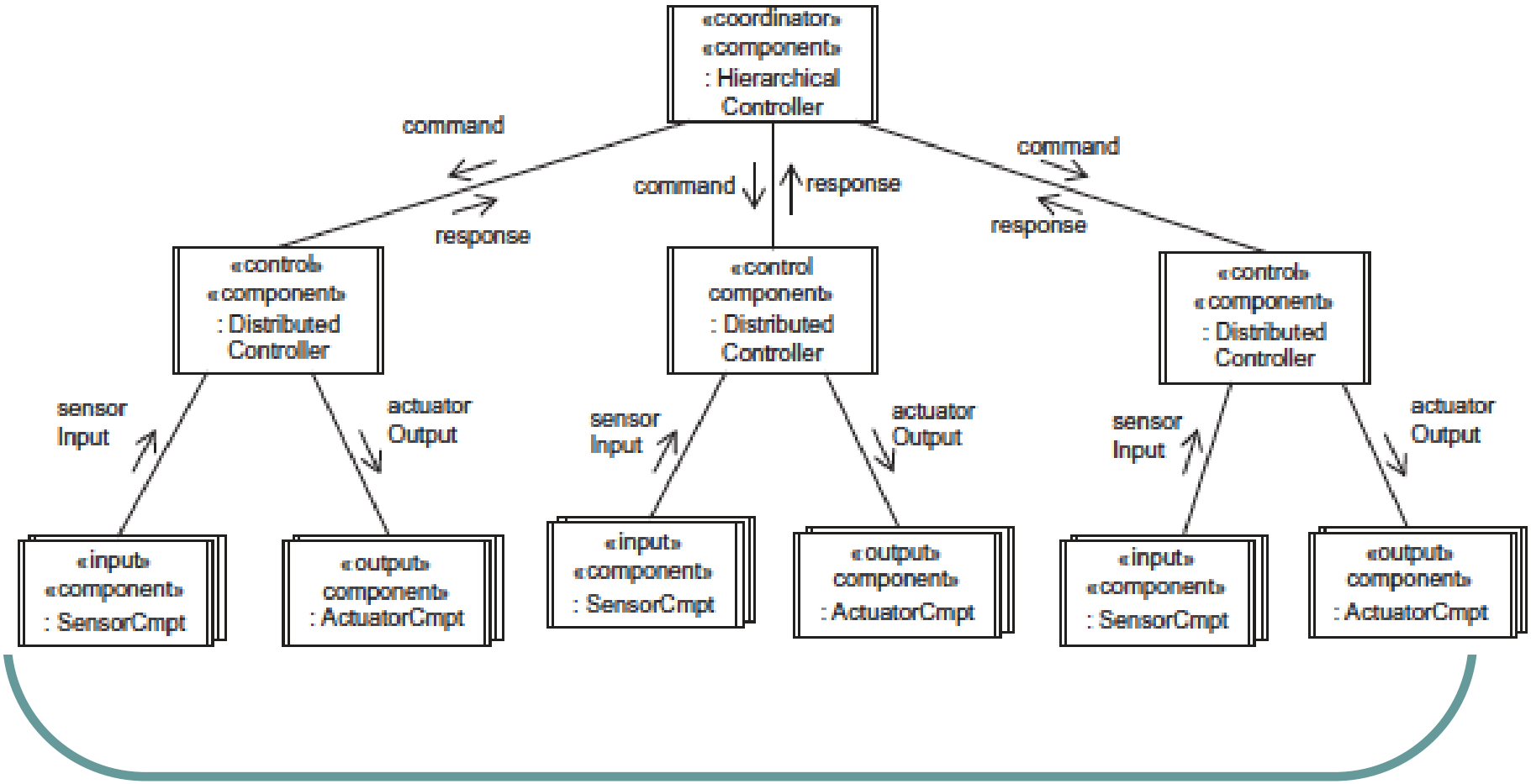
Controllo Centralizzato



Controllo Distribuito



Controllo Gerarchico



IPC in RT-POSIX

- RT-POSIX offre numerosi strumenti per l'IPC tra thread (ITC: Inter-Thread-Communication)
- Focalizzeremo la nostra attenzione su
  - Uso di aree di memoria condivise protette da mutex
  - Scambio di messaggi (lezione successiva)

Memoria condivisa tra thread

- I thread condividono l'area di memoria del processo in cui eseguono
- Di conseguenza ogni oggetto creato in area heap on in area dati globali è accessibile da ogni thread
- Questo comporta potenziali situazioni di race condition che vanno risolte attraverso l'uso di semafori per la mutua esclusione (mutex)

Creazione e distruzione di Mutex

- **`pthread_mutex_init (mutex,attr)`**
  - Crea un nuovo mutex e lo inizializza come “sbloccato” (unlocked).
  - **`mutex`** (output): di tipo `pthread_mutex_t` è un identificatore del mutex creato;
  - **`attr`** (input): per impostare gli attributi del mutex;

- **`pthread_mutex_destroy (mutex)`**
  - Dealloca un mutex.

lock e unlock

- **`pthread_mutex_lock (mutex)`**
  - Un thread invoca la lock su un mutex per acquisire l'accesso in mutua esclusione alla sezione critica relativa al mutex. Se il mutex è già acquisito da un altro thread, il chiamante si blocca in attesa di un unlock.
- **`pthread_mutex_trylock (mutex)`**
  - Analoga alla lock, ma non bloccante. Se il mutex è già acquisito, ritorna immediatamente con un codice di errore EBUSY.
- **`pthread_mutex_unlock (mutex)`**
  - Un thread invoca la unlock su un mutex per rilasciare la sezione critica, e per consentire quindi l'accesso ad un altro thread precedentemente bloccato.

Impostare i protocolli sui mutex

```
int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
                                  int protocol);
```

- Il `protocol` può essere:
  - `PTHREAD_PRIO_NONE` nessun protocollo
  - `PTHREAD_PRIO_INHERIT` per priority inheritance
  - `PTHREAD_PRIO_PROTECT` per priority ceiling
- In caso di priority ceiling, il `ceiling` del mutex deve essere specificato con:

```
int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
                                     int ceiling);
```

Esempio

- Impostare priority inheritance a un mutex
  - La variabile `mutexattr` può essere usata per inizializzare più di un mutex, se necessario

```
pthread_mutex_t mylock;
pthread_mutexattr_t mymutexattr;

pthread_mutexattr_init(&mymutexattr);
pthread_mutexattr_setprotocol(&mymutexattr,
                              PTHREAD_PRIO_INHERIT);
pthread_mutex_init(&mylock, &mymutexattr);
pthread_mutexattr_destroy(&mymutexattr);
```