ComputareComunicare

Interagire

Computare e …

- Computare
- Comunicare
- Agire
- Inframmezzare azioni (server **stateless** e con **stateful**): si veda *Interlacciare* Il caso *ServiceMath24Asynch* è **stateless!!!**
- Interagire

1. Primo computare e poi interagire?
2. Primo interagire e poi (forse) computare?

La problematica è essenziale per ile tema dei *Micro e Nano servizi*.

La computazione può emergere dalla comunicazione tra componenti ed è un approccio teorico ben studiato in molte aree della computazione, inclusi i sistemi concorrenti, distribuiti, biologici e basati su agenti. La comunicazione tra entità semplici che seguono regole definite può portare a comportamenti complessi e alla risoluzione di problemi anche sofisticati.

1. **Macchine a stati finiti comunicanti** Questo modello descrive un insieme di macchine a stati finiti che possono comunicare tra loro per sincronizzarsi o coordinare le loro azioni. Ogni macchina individualmente può essere relativamente semplice, ma la combinazione della loro comunicazione consente di ottenere comportamenti complessi. Questi sistemi vengono studiati nell'ambito della teoria dei sistemi concorrenti e nei sistemi distribuiti.
2. **Modelli basati su agenti (Multi-Agent Systems)** In questo modello, una computazione emerge dalle interazioni tra diversi agenti che comunicano tra loro. Gli agenti possono essere software, hardware, o entità astratte in grado di eseguire azioni e scambiare messaggi. Ogni agente ha una capacità limitata di elaborazione, ma il sistema nel suo complesso può risolvere problemi complessi tramite la comunicazione e la coordinazione tra gli agenti.
3. **Calcolo concorrente e parallelo** Nella computazione parallela o concorrente, più componenti eseguono computazioni in parallelo e scambiano informazioni tra loro. In modelli come il calcolo a processi (ad esempio, il modello di CSP, Communicating Sequential Processes), le unità di calcolo (processi) comunicano tramite canali di comunicazione e il calcolo viene determinato dalle loro interazioni.

- *CSP* (*Communicating Sequential Processes*): Questo modello di calcolo descrive un insieme di processi che interagiscono attraverso la comunicazione sui canali, scambiando messaggi in modo sincrono. Il comportamento globale del sistema dipende dalle comunicazioni tra i processi, e quindi, le computazioni emergono dal coordinamento dei processi attraverso il passaggio di messaggi.

4. **Lambda calcolo e pi-calcolo (modelli di comunicazione)** Il pi-calcolo (o calcolo dei processi mobili) è un modello teorico di computazione che generalizza il lambda calcolo per includere la nozione di comunicazione dinamica tra processi. Nel pi-calcolo, i processi possono creare canali di comunicazione e scambiarsi messaggi. Il calcolo stesso è una forma di comunicazione, e quindi tutta la computazione si basa sulla trasmissione di informazioni tra processi.
   - *Pi-calcolo*: Modello di calcolo nel quale la comunicazione tra processi è la base stessa del calcolo. I processi comunicano tramite canali e possono anche creare nuovi canali in modo dinamico, rendendo possibile la descrizione di sistemi distribuiti e dinamici.
5. **Modelli di calcolo distribuito** Nella computazione distribuita, più nodi o entità eseguono calcoli indipendenti, ma la soluzione globale di un problema dipende dalla cooperazione e dalla comunicazione tra questi nodi. Questo modello è molto utilizzato in contesti reali come i sistemi distribuiti, dove l'elaborazione viene suddivisa tra vari nodi di un sistema.
   - *Modelli di consenso*: In molti sistemi distribuiti, la computazione avviene tramite protocolli di consenso, dove i componenti devono comunicare per raggiungere un accordo su uno stato globale o una decisione (ad esempio, il protocollo di consenso in blockchain o negli algoritmi distribuiti di database).
6. **Automi cellulari e modelli locali** Gli automi cellulari sono un modello di calcolo in cui una griglia di celle, ciascuna rappresentante una macchina a stati finiti, evolve nel tempo seguendo semplici regole locali. Ogni cella comunica solo con le celle adiacenti. Anche se la comunicazione è limitata alle vicinanze immediate, la dinamica globale del sistema può portare a comportamenti complessi. Un esempio celebre è il Gioco della vita di Conway.
7. **Reti neurali** Le reti neurali artificiali possono essere viste come una forma di computazione basata sulla comunicazione tra nodi, dove i nodi (neuroni) sono connessi tra loro da pesi (sinapsi). I neuroni trasmettono segnali l'uno all'altro, e l'apprendimento avviene adattando la forza delle connessioni. La computazione emerge dalla trasmissione e dall'elaborazione dei segnali tra i nodi della rete.

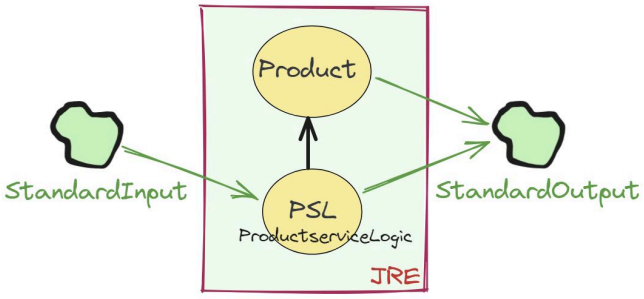
8. **Computazione quantistica con comunicazione** Anche nei computer quantistici, i qubit possono interagire tra loro tramite operazioni di entanglement e comunicazione quantistica, portando a forme di calcolo che sfruttano le proprietà della meccanica quantistica. Le reti quantistiche sono un campo emergente dove i qubit distribuiti su più nodi possono comunicare e cooperare per risolvere problemi complessi.

9. **Sistemi biologici (calcolo basato su comunicazione)** Anche in biologia, ci sono modelli teorici di calcolo che si basano sulla comunicazione tra componenti. Ad esempio, i sistemi immunitari o le reti di interazioni tra proteine possono essere visti come sistemi che “computano” risposte biologiche attraverso interazioni e segnali tra componenti.

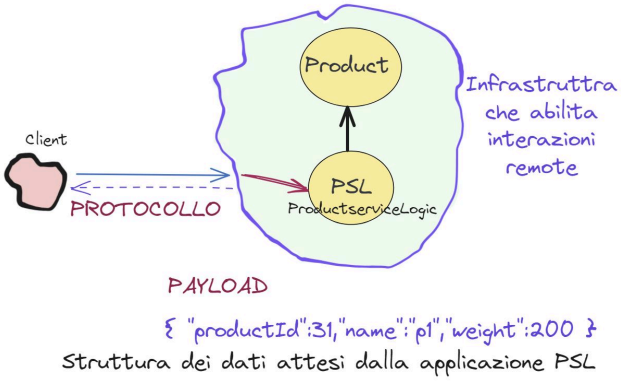
Framework abilitanti

Interagire = scambiare informazione

1. Un nucleo che computa viene reso capace di interagire con dispositivi di I/O



2. Un nucleo che computa viene reso capace di interagire con altri componenti inserendolo in un Framework



Nel caso il Framework sia **SpringBoot**:

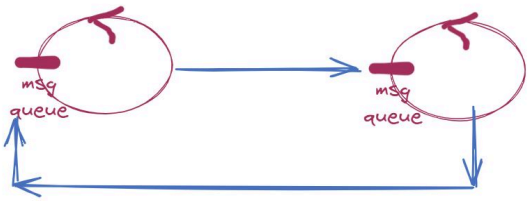
- La comunicazione avviene mediante protocollo **HTTP**
- Il Framework **SpringBoot** si occupa di gestire la comunicazione mediante un **RestController**
- Il **RestController** può operare in modo sincrono o asincrono
  - sincrono: *@PostMapping(“/createProductStr”)*
  - asincrono: *@PostMapping(“/createProductStrAsynch”)*

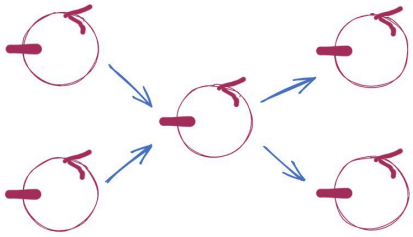
- Il **client** può operare in modo sincrono o asincrono
  - sincrono: *productservicelogicNat/src/main/java/callers/PSLCallerHTTPSynch.java*
  - sincrono: *productservicelogicNat/src/main/java/callers/PSLAnotherCallerHTTPSynch.java*
  - asincrono: *productservicelogicNat/src/main/java/callers/PSLCallerHTTPAsynch.java*

3. Un 'nuovo tipo' di componente

Un componente **intrisecamente capace** di interagire **in modo asincrono** con gli altri componenti dello stesso tipo

si relaziona con altri per 'computare'





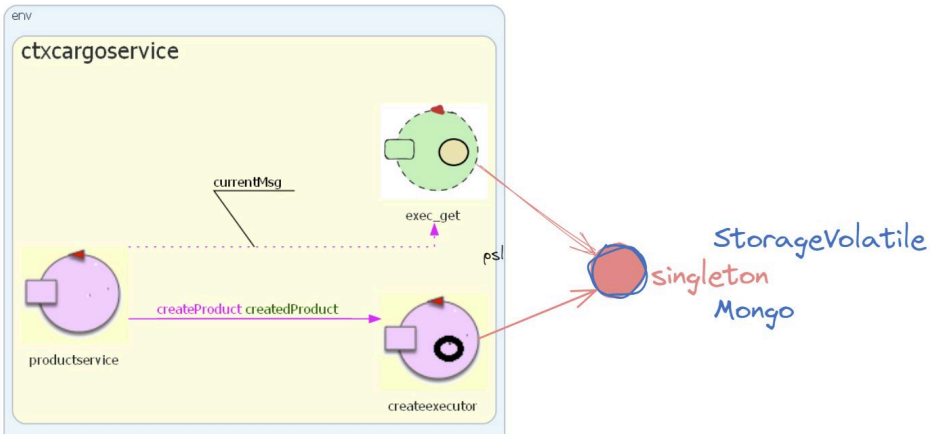
Interlacciare

Le interazioni asicnrone tra microservizi possono introdurre problemi di sincronizzazione e di condivisione di oggetti.

Considerando agli attori qak *QakActors24* come una forma di nanoservizi, osserviamo che (si veda *Problematiche usando attori*):

1. è possibile eseguire azioni locali in parallelo (ad esempio **getProduct** in **productservice**)
2. è possibile sequenzializzare azioni attivate in parallelo con altre (ad esempio **createProduct** in **productservice**) usando il meccanismo di delega di azioni a un attore
3. è talvola necessario che attori diversi condividano uno stesso oggetto
4. è possibile evitare l'uso di meccanismi quali syncrhronized, lock, ecc.

La condivsione di oggetti tra microserivzi talvolta è necessaria.



Esperimento di interazione 'naive'

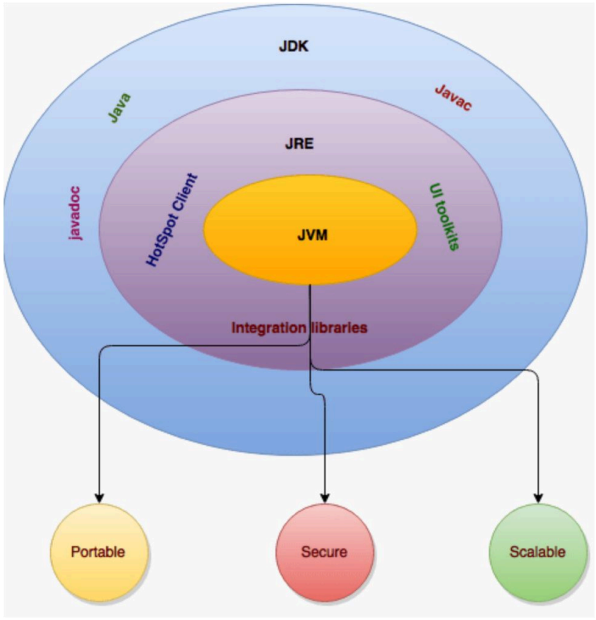
VIDEO: ESPERIMENTIHTTPsynch/asynch:
https://unibo.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=17f063c3-a30d-4112-8dc1-b1f90110f4ae

1. eseguo **gradlew bootRun** nel progetto **productservicelogicNat**
2. lancio **PSLCallerHTTPSynch.java** due volte e guardo i tempi nel file di log
3. lancio **PSLCallerHTTPAsynch.java** due volte e guardo i tempi nel file di log Uno dei due restituisce wrong (MA POTREI AVERE DUPLICATI)
4. lancio **PSLCallerHTTPAsynch.java** e poi lancio (entro 4 sec) **PSLAnotherCallerHTTPSynch.java** e guardo nel file di log che prima finisce il sincrono e poi l'asincrono

Esperimento con ELK

1. attivo **<appender name="LOGSTASH" ...** nella configurazione di **logback.xml**
2. lancio **docker-compose-ELK.yml**
3. rifaccio gli Esperimenti di interazione e guardo i log in **Kibana**

Questi appunti su Java nascono dal fatto che il *package java.lang* non contiene istruzioni specifiche per la comunicazione diretta con il sistema operativo nè tantomento istruzioni per la comunicazione remota.



- **JVM**: È la macchina virtuale che esegue il bytecode Java. È il cuore dell'ambiente di esecuzione di Java.
- **JRE**: Include la JVM e altre librerie necessarie per eseguire le applicazioni Java. Fornisce l'ambiente di esecuzione completo.
- **JDK**: È l'intero kit di sviluppo Java, che contiene la JRE (e quindi la **JVM**), oltre a strumenti per scrivere e compilare programmi.

Caratteristica	JVM (Java Virtual Machine)	JRE (Java Runtime Environment)
Funzione principale	Esegue il bytecode e gestisce l'esecuzione di programmi Java.	Fornisce tutto ciò che serve per eseguire un programma Java (inclusa la JVM).
Componenti principali	Include l'interprete del bytecode, il garbage collector, il ClassLoader, e l'execution engine.	Include la JVM, le librerie di classi Java e altre risorse necessarie.
Indipendenza dalla piattaforma	La JVM permette l'esecuzione del bytecode su diverse piattaforme.	La JRE include la JVM specifica per ogni piattaforma, quindi cambia da piattaforma a piattaforma.
Utilità per lo sviluppo	È specifica per l'esecuzione, non fornisce strumenti di sviluppo.	Anche la JRE non include strumenti di sviluppo (quelli sono inclusi nel JDK).
Integrazione con la JDK	Parte della JRE e del JDK.	Parte della JDK (ma la JRE è usata per eseguire il programma).

- In Java, il *package java.lang* è il nucleo della libreria standard e fornisce classi fondamentali per il linguaggio, ma non contiene istruzioni specifiche per la comunicazione diretta con il sistema operativo (**SO**). Le classi di java.lang sono più orientate a funzionalità di base del linguaggio, come la gestione dei tipi primitivi, le operazioni sulle stringhe, il threading, l'input/output (**I/O**) di base, e la gestione delle eccezioni.

- Per quanto riguarda la comunicazione con il sistema operativo, Java adotta un approccio astratto e indipendente dalla piattaforma. Le funzionalità specifiche del sistema operativo, come la gestione di file, la comunicazione di rete o le interazioni hardware, sono fornite da altre librerie o framework specializzati, non da **java.lang**.

- La *classe System* in Java è parte della libreria standard (**java.lang**) e non cambia direttamente a seconda del sistema operativo. La sua implementazione è universale e coerente su tutte le piattaforme, ma può interagire con il sistema operativo in modi diversi, grazie all'astrazione fornita dalla Java Runtime Environment (**JRE**) e dalla Java Virtual Machine (**JVM**). Tuttavia, la JRE deve implementare delle funzioni native per fare in modo che la classe **System** possa interagire con i dettagli del sistema operativo sottostante.

- La **JRE** di Java, da sola, non ha accesso diretto all'hardware del *Raspberry Pi*, ma è possibile accedere ai **PIN** **GPIO** del *Raspberry Pi* grazie a librerie come **Pi4J**.

Java NIO

Java NIO (**New IO**) è un'API IO alternativa alternativa alle API Java IO e Java Networking standard.

- Java NIO consente di eseguire IO non bloccanti. Ad esempio, un thread può chiedere a un canale di leggere i dati in un buffer. Mentre il canale legge i dati nel buffer, il thread può fare qualcos'altro. Una volta che i dati sono stati letti nel buffer, il thread può continuare a elaborarli. Lo stesso vale per la scrittura dei dati nei canali.
- A volte si afferma che NIO significhi *Non-blocking IO* . Tuttavia, questo non è ciò che NIO intendeva originariamente. Infatti, parti delle API NIO sono effettivamente bloccanti, ad esempio le API dei file, quindi l'etichetta "Non-blocking" sarebbe fuorviante.

Java HttpURLConnection

- La classe HttpURLConnection è una classe astratta che si estende direttamente dalla classe URLConnection . Include tutte le funzionalità della sua classe padre con caratteristiche aggiuntive specifiche di HTTP. HttpsURLConnection è un'altra classe che viene utilizzata per il protocollo HTTPS più sicuro.
- È una delle scelte più diffuse tra gli sviluppatori Java per interagire con i server Web e il team di sviluppo Android ha ufficialmente suggerito di utilizzarlo ovunque possibile.