**INTRO 2024**

**0) SISTEMA?**

Spazio concettuale = tutto ciò che trovi nella documentazione

Olismo o riduzionismo?

* Un sistema è tale perchè possiamo **distinguere** in esso più componenti, che occorre introdurre e definire in modo opportuno (come vedremo per i [*Componenti software*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\Intro2024.html#componenti-software)).
* Si dibatte ancora molto se le proprietà di un sistema possono essere sempre spiegate in termini delle proprietà dei componenti ([*Riduzionismo*](https://it.wikipedia.org/wiki/Riduzionismo) ) o emergano come aspetti nuovi rispetto ai componenti stessi ([*Olismo*](https://it.wikipedia.org/wiki/Olismo) ).

**1) SISTEMA SOFTWARE**

In questa sede limiteremo la nostra attenzione ai sistemi software, intendendo per software *l’insieme dei programmi che possono essere impiegati su un sistema fisico di elaborazione*.

Il software

Storicamente, il software nasce come l’insieme di **istruzioni-macchina** che deve essere eseguito da un computer su un insieme di **dati** per realizzare un algoritmo.

L’evoluzione di linguaggi ad alto livello ha quasi subito visto l’introduzione di costrutti capaci di esprimere semplici forme di aggregazioni di istruzioni e dati (si veda [*Componenti software di base*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\Intro2024.html#componenti-software-di-base)) che sono poi progressivamente evolute in veri e propri [*Componenti software*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\Intro2024.html#componenti-software).

L’uso del termine *sistema software* enfatizza, in genere, prodotti complessi, costruiti o studiati in termini della organizzazione e della interazione tra i componenti che formano il prodotto.

Vi è quindi una stretta relazione con il concetto di [*Architettura software*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\Intro2024.html#archittettura-software) .

Immagine che contiene cono, triangolo

Descrizione generata automaticamenteViste di un sistema software

Un sistema software può essere descritto da diversi punti di vista.

**Vista esterna**

Il sistema viene visto come una entità accessibile attraverso una [*API*](https://it.wikipedia.org/wiki/Application_programming_interface) (**Application Programming Interface**) intesa come il punto di contatto che consente l’interazione tra codici in esecuzione

**Vista** **interna**

Il sistema viene visto come un insieme di enti computazionali (funzioni, oggetti, processi, etc.) che operano interagendo tra loro e con il mondo esterno (clienti, dispositivi, etc.) usando adeguati supporti

**Vista sommersa**

Il sistema è l’ultimo livello (*layer*) di uno stack i cui livelli sottostanti (librerie, infrastruttura, etc. ) forniscono il supporto alla esecuzione.

Nel corso degli anni sono stati proposti e usati diversi tipi di componenti software, che hanno contribuito a formare lo spazio concettuale dei linguaggi di programmazione di alto livello.

**2) COMPONENTI SOFTWARE**

In generale, il concetto di componente software può essere definito come:

**un’unità autonoma e riusabile di codice che svolge una funzione specifica all’interno di un sistema software, promuovendo la riusabilità, la modularità e la manutenibilità del codice**

Questa idea di base ha subito una profonda evoluzione nel corso degli anni, tanto che occorrerebbe un corso ad hoc per descriverla. Noi cercheremo di delinearne alcuni punti importanti sia sul piano concettuale, sia sul piano pragmatico delle moderne pratiche di costruzione di sistemi software.

Componenti software di base

I linguaggi di programmazione più diffusi permettono di costruire sistemi software composti da due specie di componenti-base:

|  |  |
| --- | --- |
| *[Immagine che contiene cerchio, Carattere, Elementi grafici, design  Descrizione generata automaticamente](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\_images\funz.PNG)* | Una funzione rappresenta un componente che permette la definizione parametrica di istruzioni e la loro esecuzione mediante trasferimento di controllo.  [è un corpo morto che prende vita se viene chiamata, quindi necessita di un’azione esterna]  *Linguaggio di riferimento*: JavaScript (Node.js) [o anche Python] |
| *[Immagine che contiene cerchio, clipart, cartone animato, design  Descrizione generata automaticamente](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\_images\obj.PNG)* | Un oggetto incapsula uno stato e un insieme di funzioni (metodi), la cui esecuzione avviene mediante chiamata di procedura (trasferimento di controllo).  *Linguaggio di riferimento*: Java.  Un oggetto può essere dotato di comportamento autonomo incapsulando un **Thread**. |

Oltre gli oggetti

* Lo sviluppo delle reti informatiche e di Internet ha promosso la costruzione di **sistemi software distribuiti** su più nodi di elaborazione, su ciascuno dei quali possono essere eseguiti programmi espressi medianti linguaggi diversi.
* La costruzione di sistemi software distribuiti eterogenei è resa possibile dall’uso (entro funzioni ed oggetti) di protocolli di comunicazione (come UDP, TCP, HTTP, MQTT, CoAP, etc.) e richiede logicamente componenti capaci di interagire (spesso in modo asincrono) mediante scambio di messaggi e non più mediante trasferimento di controllo con chiamate di procedura. [procedure call non va più bene a causa dei sistemi eterogenei --> comunicazione asincrona]

Gli attori

* I linguaggi di programmazione più diffusi non hanno ancora introdotto componenti intrinsecamente capaci di interagire a messaggi, evidenziando una lacuna che può essere colmata attraverso l’introduzione di **librerie**, come ad esempio la libreria *[Akka](https://akka.io/)* per Java/Scala, che trae ispirazione dal [*Modello computazionale ad attori*](https://en.wikipedia.org/wiki/Actor_model) .

Le motivazioni introdotte in *[Akka Actors](https://doc.akka.io/docs/akka/current/typed/guide/actors-motivation.html)* sono condivise da molti e verranno prese anche qui come riferimento.

Un modo alternativo, rispetto all’uso di una libreria, per introdurre componenti che operano come attori, è quello di definire un linguaggio di programmazione custom, con un adeguato livello di astrazione, avvalendosi di tecniche sviluppate nel campo dei [*Domain Specific Languages*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\Principi.html#domain-specific-languages).

Questo approccio è stato adottato per definire il linguaggio denominato [*qak*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\QakActors24.html#qakactors24) [Quasi Attori Kotlin], con lo scopo di esprimere concetti di alto livello (technology-independent) per definire componenti software capaci di interagire attraverso scambio di messaggi.

[Scala è un esempio di contaminazione dei linguaggi con java]

Il linguaggio [*qak*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\QakActors24.html#qakactors24)

* intende promuovere la definizione di [*modelli eseguibili*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\Intro2024.html#modelli-e-metamodelli) di un sistema software, da utilizzare durante l’analisi di un problema e il progetto e lo sviluppo di prototipi di sistemi distribuiti. In molti casi, può anche essere usato per costruire il prodotto finale;
* non è un linguaggio general-purpose, ma piuttosto un meta-modello (si veda [*Metamodellazione*](https://it.wikipedia.org/wiki/Metamodellazione)) per la definizione e costruzione di architetture software distribuite;
* introduce componenti software che portano in primo piano l’idea di [*attore*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\QakActors24.html#qakactors24) e di [*interazione*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\Interaction.html#interaction) tra attori, lasciando sullo sfondo (nella [*parte sommersa*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\Intro2024.html#vista-sommersa)) i dettagli implementativi, grazie alla [*Qak factory*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\QakActors24.html#qak-factory).

Attori qak

|  |  |
| --- | --- |
| *[Immagine che contiene cerchio, Elementi grafici, clipart, compact disk  Descrizione generata automaticamente](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\_images\actor.PNG)* | Un [*attore*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\QakActors24.html#actors24-l-attore) è un componente software dotato di comportamento autonomo, capace di ricevere e trasmettere messaggi e di gestire i messaggi in modo da porre in esecuzione funzioni e/o oggetti. |

Un sistema basato su attori esclude la presenza di memoria comune tra componenti, prefigurando la possibilità che i componenti di uno stesso sistema possano essere distribuiti su nodi di elaborazione diversi.

Per rendere concreta questa possibilità, occorre l’introduzione di server e l’uso di protocolli di comunicazione con cui rendere possibile l’invio di messaggi avvalendosi delle infrastrutture di rete.

Per rendere questo dettaglio tecnologico trasparente a livello applicativo, il linguaggio [*qak*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\QakActors24.html#qakactors24) introduce il concetto di contesto come elemento capace di abilitare il passaggio da comunicazioni locali a comunicazione remote.

Contesti qak

|  |  |
| --- | --- |
|  | Un [*contesto*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\QakActors24.html#qakactors24) è un componente software che gestisce (in modo sommerso) N>0 Attori, **abilitandoli** alla ricezione e trasmissione di messaggi via rete attraverso l’introduzione di un server TCP e un server CoAP.  La parte sommersa è fornita da una libreria che crea la [*Qak infrastructure*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\QakActors24.html#qak-infrastructure). |

**3) ARCHITETTURA SOFTWARE**

Una volta accettato che un sistema software sia composto da un insieme di componenti di una qualche natura, il concetto di [*architettura*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\Architetture.html#architetture) assume, dagli anni 90 del secolo scorso in avanti, un ruolo progressivamente più importante.

Evoluzione delle architetture

Si sviluppano e si affermano nuovi modi di concepire le architetture software, partendo non più da componenti come oggetti, ma da componenti orientati al [*Modello computazionale ad attori*](https://en.wikipedia.org/wiki/Actor_model) e che assumono la forma di **servizi**.

|  |  |
| --- | --- |
| L’evoluzione delle architetture software può essere riassunta in una progressiva trasformazioni di un sistema monolitico in un sistema composto da micro servizi e [*nano servizi*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\Intro2024.html#nano-servizi). | Un attore [*qak*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\QakActors24.html#qakactors24) può essere visto come un micro-servizio, con precise responsabilità, incluso in un contesto allocato su un nodo (fisico o virtuale) di elaborazione. |

Architetture di integrazione

|  |  |
| --- | --- |
| Nascono anche nuovi approcci alle  architetture di integrazione  che consentano il flusso di dati tra sistemi e applicazioni software. | *[Immagine che contiene testo, Carattere, schermata, linea  Descrizione generata automaticamente](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\_images\EvolutionProgModels.PNG)*  Nelle architetture di integrazione a microservizi, un ruolo-chiave è relativo ai diversi modi con cui è possibile far interagire gli attori |

nano servizi

Il concetto di **nano-servizio** viene introdotto da [*AWS Lambda*](https://aws.amazon.com/it/getting-started/hands-on/run-serverless-code/?pg=gs&sec=lyfa), come forma di *[FaaS](https://it.wikipedia.org/wiki/Function_as_a_service)* (Function as a service) in cui le Funzioni offrono servizi accessibili in rete, liberando il loro progettista dalla necessità di occuparsi delle infrastrutture e della gestione dei server.

Le funzioni [*AWS Lambda*](https://aws.amazon.com/it/getting-started/hands-on/run-serverless-code/?pg=gs&sec=lyfa) -> possono anche agire come **collante** tra [*AWS Services*](https://it.wikipedia.org/wiki/Amazon_Web_Services) -> concepiti come sorgenti di eventi a cui le funzioni stesse possono reagire.

* Il linguaggio [*qak*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\QakActors24.html#qakactors24) permette di **modellare la struttura logica** di architetture di questo tipo e di ragionare sulla loro possibile organizzazione.
* Anche in questi caso il progettista non dovrà occuparsi delle infrastrutture e della gestione dei server, che viene realizzata in modo sommerso nella [*Qak infrastructure*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\QakActors24.html#qak-infrastructure). Ovviamente saranno assenti le features legate alla scalabilità etc. ma l’intendo è di usare [*qak*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\QakActors24.html#qakactors24) anche nella progettazione e nello sviluppo di applicazioni IOT e non solo cloud.

**4) LA PRODUZIONE DEL SOFTWARE**

La necessità/opportunità di definire un’appropriata architettura del software è strettamente correlata al processo di *definire, documentare e mantenere* i requisiti del sistema.

Il cosa e il come

Mentre l’architettura del software si rivolge allo “spazio della soluzione” o al **“come”**, la [*Ingegneria dei requisiti*](https://it.wikipedia.org/wiki/Ingegneria_dei_requisiti#:~:text=L'ingegneria%20dei%20requisiti%20%C3%A8,e%20nell'ingegneria%20del%20software) -> si rivolge allo “spazio del problema” o al **“cosa”**.

Per costruire (capendo e motivando quello che si fa) occorre avere:

* un [*metodo di sviluppo*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\Intro2024.html#metodologie-per-lo-sviluppo-del-software)
* un insieme (evolutivo) di [*Principi ispiratori*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\Intro2024.html#principi-ispiratori) di rifermento [principi ispiratori perché non dobbiamo avere bias fissi che ci impediscono di apprendere nuove cose]
* un modo compatto ed efficace di descrivere problematiche e scelte progettuali.

Auspicabilmente, questa descrizione dovrebbe essere **comprensibile sia agli esseri umani sia alla macchina**, al fine di evitare ambiguità e errori di ‘trascrizione’. L’uso di modelli [*qak*](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\QakActors24.html#qakactors24) è motivata da questa necessità.

|  |  |
| --- | --- |
| Processi e conoscenza   * Il lavoro di diversi attori (*committente, product-owner, analista, progettista, programmatore, etc.*) si basa du un insieme di **assunzioni, conoscenze e decisioni** che rimangono spesso implicite all’iterno di un processo di produzione * Occorre esprimere in modo esplicito e non ambiguo tali assunzioni, conoscenze e decisioni | Modelli   * Un modo tradizionalmente usato nell’ingegneria è la introduzione di **modelli del sistema** espressi con un linguaggio ‘formale’, cioè dotato di sintassi e semantica comprensibile sia all’uomo sia alla macchina. |

Ma c’è ancora bisogno di questo con i nuovi strumenti di AI? Perchè non *[UsiamoChatGPT](file:///C:\\Users\\valen\\OneDrive\\Desktop\\issLab24\\iss24Material\\html\\UsiamoChatGPT.html" \l "usiamochatgpt)* ? [Perché non sa il COSA]

[In SCRUM si lavora per SPRINT, ovvero per obiettivi. I vari sprint0 dovrebbero essere identici perchè il COSA della commissione deve essere disambiguato e quindi vanno fatti dei modelli dei requisiti che devono essere identici. Gli analisti di un progetto devono essere d’accordo per poter proseguire nel progetto. Ogni sprint produce un prodotto deployable, che può essere venduto, solo che non rispetta tutti i requisiti fissati nella fase di analisi]

**QAK SERVICE 24 USAGE**

**Il servizio Il servizio servicemath**

La nostra ditta (disiUnibo) ha realizzato un servizio stateless, accessibile via rete, che esegue (al momento) richieste di calcolo dell’N-mo (N>=0) numero di Fibonacci.

AVVERTENZA: questo caso di studio si basa su uno schema di calcolo dei numeri di Fibonacci basato su doppia ricorsione, in modo da poter apprezzare differenze temporali, anche sensibili, nella esecuzione del servizio.

In questa fase ci occuperemo solo della [Vista esterna](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\QakServiceSynch24.html#vista-esterna) del servizio, con l’obiettivo di capire come accedere al servizio e come utilizzarlo sia mediante browser sia mediante programmi.

Dalla sezione [QakService24Init](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\QakService24Init.html#qakservice24init) inizieremo invece ad affrontare il tema della [Vista interna](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\QakServiceSynch24.html#vista-interna) del servizio adottando il punto di vista del costruttore del servizio, introducendo anche elementi importanti della sua [Vista sommersa](file:///C:\Users\valen\OneDrive\Desktop\issLab24\iss24Material\html\QakServiceSynch24.html#vista-sommersa).

**servicemath: attivazione**

Il servizio (implementato in Java) può essere attivato in due modi:

* 1. eseguendo il file batch unibo.servicemath.bat (su windows)
  2. unibo.servicemath (su Linux) etsratto dal file di distribuzione del prodotto (unibo.servicemathsynchbase-1.0.zip)

AVVERTENZA Su Windows fare unzip del file in una directory dal nome breve, ad es. C:/issRun

eseguendo il comando docker: docker-compose servicemath.yaml

[Docker ps -a per vedere i container attivi]

[Il browser interpreta html, il curl ti spara direttamente il codice html]

[http (human to machine), websocket (machine to machine). http serve per mostrare le cose agli utenti]

[il localhost si connette alla pagina 8088, ma c’è anche la porta 8011. la porta 8011 serve per girare la richiesta ricevuta dal server all’attore. Il web server mi aiuta solo a gestire le connessioni, la logica applicativa la fa qualcun altro. Di conseguenza io posso parlare direttamente sulla porta 8011 evitando l’intermediario. Quindi posso creare un programma con websoket sulla porta 8088 oppure una connessione tcp sulla porta 8011 (server tcp/coAP creato dal contesto). Da ingegnere devo però evitare che chiunque possa parlare direttamente con l’attore.]

[Nei qak è il contesto che realizza un server tcp e un server coAP]

Una volta attivato, il servizio fornisce una pagina HTML, accessibile mediante browser alla porta 8088, in cui compare:

* una zona di input, per la immissione del valore N
* una zona di output, per visualizzare la risposta e altre informazioni.

**Uso del servizio servicemath**

Come utenti, ci aspettiamo che il servizio sia accessibile in rete attraverso una qualche interfaccia di programmazione delle applicazioni (API) intesa come intermediario software tra il servizio stesso e i client che vogliono interagire con lui.

Le API semplificano la programmazione dei client perché di solito nascondono i dettagli interni del servizio, esponendo parti che sono mantenute coerenti anche nel caso di modifiche interne al servizio

* Il servizio è remoto? Se risponde allora c’è una connessione http (un web server)
* Una pagina cambia perché il server ne genera un’altra oppure si usano le pagine dinamiche, ma le pagine dinamiche non usano http, usano le websocket e per rispondere il server invia qualche stringa di risposta, ma chi la gestisce? Una funzione javascript che è stata scritta dentro il browser.
* Lui lavora solo via http o solo tramite websocket? Nel 2024 obv usa http per aprire la prima pagina e poi websocket per rispondere alle richieste send. Come faccio a sapere che la mia ipotesi è corretta? Se veramente lavora a websocket, se apro una seconda istanza del browser (un’altra pagina) una volta che invio la richiesta su una pagina la risposta comparirà su entrambe

Per poter usare il nostro servizio, occorre conoscere:

* il protocollo da usare per l’invio dei messaggi di richiesta via rete
* la struttura dei messaggi che il servizio è capace di interpretare.

Il servizio è anche una entità osservabile, nel senso che emette informazioni percepibili da client connessi come osservatori e non come emettitori di comandi. Si veda:

* Osservabilità via WS
* Osservabilità via CoAP
* Osservabilità via MQTT

**Descrizione della API in linguaggio naturale**

Di solito, le informazioni su come usare un servizio vengono date attraverso qualche documento scritto in linguaggio naturale (ambiguo). Ad esempio quali protocolli sono utilizzabili:

* TCP
* CoAP
* MQTT
* HTTP
* WebSocket

Indipendentemente dal protocollo e dal servizio, la struttura dei messaggi che ogni client deve inviare è fissata come qui di seguito indicato.

**Struttura generale dei messaggi**

I messaggi devono avere la seguente struttura:

msg( MSGID, MSGTYPE, SENDER, RECEIVER, CONTENT, SEQNUM )

ove

* MSGID: identificativo del messaggio
* MSGTYPE: tipo del msg
* SENDER: nome del componente che invia il messaggio
* CONTENT: contenuto applicativo (payload) del messaggio
* RECEIVER: nome del componente chi riceve il messaggio
* SEQNUM: numero di sequenza del messaggio

L’argomento MSGTYPE può assumere uno dei seguenti valori: dispatch, request, reply, event

Al variare del servizio o del tipo di messaggio, cambiano gli altri argomenti del messaggio.

Essendo questa la struttura dei messaggi qak, il servizio stesso è molto probabilmente realizzato internamente da uno o più Attori qak.

**Contenuto specifico dei messaggi**

Nel caso del nostro servizio, il messaggio per richiedere il calcolo di un numero di Fibonacci assume la forma che segue:

msg( MSGID, request, callername, CONTENT, servicemath, 1)

le parti MSGID e CONTENT devono avere la forma

MSGID : dofibo

CONTENT: dofibo( N ) |N>=0, numero di Fibonacci da valutare

il servizio fornisce la risposta con un messaggio in cui la parte CONTENT assume la forma che segue:

fibodone( CALLER,N,RESULT,TIME )

ove

* CALLER: nome del client che ha invocato il servizio
* N: numero di Fibonacci che il client ha chiesto di valutare
* RESULT: valore del numero di Fibonacci di posizione N
* TIME: tempo richiesto per il calcolo

**Uso naive del servizio**

L’uso del servizio consiste nella scelta di un protocollo e nella scrittura di un programma che utilizza tale protocollo per inviare una richiesta e ricevere la risposta.

**BasicMsgUtil.kt**

Per facilitare la costruzione di messaggi secondo la Struttura generale dei messaggi indicata, viene fornita la classe *unibo.basicomm23.utils.BasicMsgUtil.kt* come parte della libreria unibo.basicomm23-1.0.jar descritta in *unibo.basicomm23*.

[Tutti i messaggi sono entità che possono essere manipolate attraverso un oggetto fornito da libreria]

Questa utility fornisce metodi per costruire i diversi tipi di messaggio dispatch, request, reply, event come oggetti che implementano la interfaccia IApplMessage.

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente**IApplMessage**

Per comodità di lettura, riportiamo qui *unibo.basicomm23.interfaces.IApplMessage* della libreria unibo.basicomm23-1.0.jar

L’aspetto naive consiste nella scrittura di un diverso programma per ogni diverso protocollo, il che conduce a una prolificazione di codice ricco di dettagli tecnologici, quando invece la logica (inviare un messaggio di richiesta e gestire il previsto messaggio di risposta) è la stessa in ogni caso.

[aka sono uguali nel COSA (inviare una richiesta e ricevere una risposta), diverse nel COME (tecnologies dependecies). Spoiler: grazie a una libreria basterà cambiare un parametro e lasciare il codice essenzialmente uguale]

Riprenderemo questo importante punto più avanti, nella sezione Il COSA e il COME.

**Il servizio usato via TCP**

Un client di nome clientjava che vuole ottenere il 33° numero di Fibonacci può inviare via TCP alla porta 8011 il seguente messaggio di richiesta:

msg(dofibo,request,clientjava,seevicemath,dofibo(33),K) |con Int K>0

Il client riceverà (sulla sua connessione TCP) un messaggio di risposta quale:

msg(fibodone,reply,servicemath,servicemath,fibodone(servicemath,33,3524578,14 ),J)|con Int J>0

[c’è un errore negli argomenti, correggi in seguito]

Esempio di uso con TCP (issLab24\servicemath24Synch\demo\ServiceCallerTCPNaive.java):

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Immagine che contiene testo, schermata

Descrizione generata automaticamente

Attenzione all’uso della utility BasicMsgUtil.kt al punto 8 e alla interazione asincrona tra i punti 9 e 10: il programma potrebbe eseguire altro codice

**Il servizio come risorsa CoAP**

Il servizio è utilizzabile anche inviando messaggi di richiesta alla porta 8011 mediante il protocollo CoAP. L’uso di CoAP:

* 1. promuove una Vista esterna del servizio come risorsa accessibile in stile REST su supporto UDP anzichè TCP. Nel nostro caso l’URI del servizio-risorsa è IP:8011/ctxservice/servicemath
  2. rende il servizio osservabile da parte di clienti CoAP che fanno una subscribe alla topic: ctxservice/servicemath

[put o post se voglio ottenere risposte, ma meglio put]

[CoAP permette interazioni restful usando dispositivi non molto potenti come i pc. Se voglio comunicare col mio servizio, questo viene visto come una risorsa URI]

[SendUsingCoap è sincrona bloccante (a differenza di TCP) || client.put è bloccante]

**Il COSA e il COME**

Avere introdotto un diverso programma per ogni specifico protocollo di interazione con il servizio è un chiaro indizio che alla macchina bisogna dire in modo non ambiguo COME agire , mentre nella nostra mente di progettisti è chiaro un denominatore comune che può essere sintetizzato nel COSA fare:

inviare un messaggio di richiesta al servizio e gestire il previsto messaggio di risposta

**Verso una logica della interazione**

Nella prossima sezione, QakService24WithInteraction, cercheremo un modo per esprimere a livello di codice solo la logica della interazione, lasciando a una parte sommersa il compito di realizzarla in modo dipendente da una specifica tecnologia di comunicazione. Sarà il nostro primo passo nel cammino che porta dagli oggetti agli attori e ai modelli.

**QAK SERVICE 24 WITH INTERACTION**

**Oltre i singoli protocolli**

Costruiamo un client che interagisce con il servizio introdotto in QakService24Usage in modo diverso da quanto visto in quella sede, lasciando sullo sfondo i dettagli del protocollo usato.

L’obiettivo è rendere il progetto del client e il suo codice focalizzato sulla logica applicativa: e sul tipo sincrono/asincrono di comunicazione che si vuole realizzare. In altre parole, facciamo un primo passo verso una direzione che pone in primo piano il COSA fare, piuttosto che sul COME fare.

**Svantaggi nell’uso di astrazioni**

Il passaggio dal come al cosa significa che molti aspetti, anche rilevanti, sull’uso dei protocolli verranno inseriti nella parte sommersa del client.

Con questo modo di procedere si perderà il controllo di molti particolari, che potrebbero rivelarsi importanti.

**Vantaggi con l’uso di astrazioni**

Avremo però il vantaggio di maggiore astrazione ed economia concettuale, utile nelle prime fasi del processo di sviluppo software: l’analisi dei requisiti e del problema e l’impostazione di una prima progettazione dell’architettura del sistema.

Su questo punto focalizzeremo l’attenzione di buona parte delle nostre attività future.

**Dai protocolli a Interaction**

La libreria unibo.basicomm23 fornisce supporti che:

* introducono un concetto (astrazione) di alto livello: quello di Interconnessione punto-punto tra due componenti software
* definiscono l’interfaccia Interaction come contratto che gli oggetti che dovranno realizzare l’astrazione dovranno rispettare
* realizzano utility per l’uso di vari protocolli, implementando l’interfaccia Interaction per alcuni di essi (tcp, udp, coap, mqtt, ws, http).
* forniscono la classe unibo.basicomm23.utils.ConnectionFactory come factory degli oggetti-supporto che realizzano l’astrazione Interconnessione.

Lo schema del codice di un client può ora essere schematizzato come segue:

* Immagine che contiene testo, schermata, Carattere, linea

  Descrizione generata automaticamente*Creazione oggetto di supporto*
  1. Definizione del protocollo da usare
  2. Definizione del messaggio di richiesta
  3. Specifica (protocol-related) dell’host del servizio
  4. Specifica (protocol-related) dell’accesso al servizio
  5. Creazione mediante ConnectionFactory di un oggetto che implementa Interaction
* *Richiesta sincrona*
  1. Immagine che contiene testo, schermata, Carattere, linea

     Descrizione generata automaticamenteIl metodo request blocca il client fino alla ricezione della risposta.
* *Richiesta asincrona*
  1. Immagine che contiene testo, Carattere, schermata, linea

     Descrizione generata automaticamenteIl metodo forward è di tipo fire-and-forget
  2. Il metodo receiveMsg blocca il client fino alla ricezione della risposta.

Tra le due chiamate, il client può eseguire altre azioni.

**ServiceCallerInteraction**

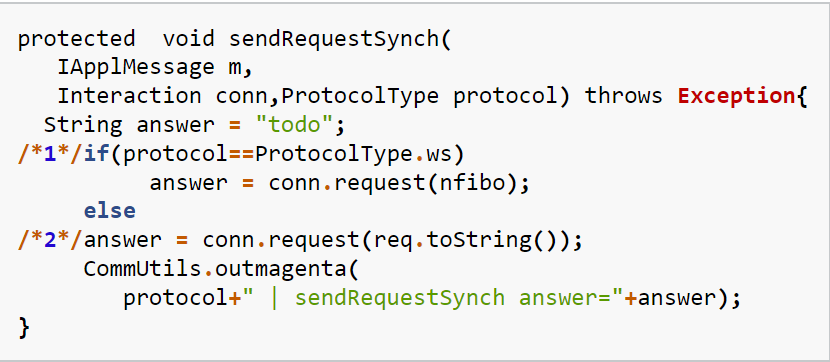
La impostazione del client è, come al solito, relativa alla definizione di un metodo doJob, cui affidiamo, in questo caso, il compito di usare tutti i protocolli.

**ServiceCallerInteraction: impostazione**

1. Dichiarazione del supporto che realizza Interaction
2. Definizione del messaggio di richiesta
3. Definizione della topic MQTT usata per la risposta
4. Invio della richiesta usando TCP
5. Invio della richiesta usando MQTT
6. Invio della richiesta usando CoAP
7. Invio della richiesta usando WebSocket
8. Attesa, per non perdere possibili ricezioni asincrone delle risposte
9. Terminazione del client
10. Specifica del metodo di invio richiesta

**ServiceCallerInteraction: invio richiesta**

1. Impostazione parametri per TCP
2. Impostazione parametri per CoAP
3. Impostazione parametri per MQTT
4. Impostazione parametri per WS
5. Creazione supporto
6. Possibile settaggio per tracing
7. Invio richiesta in modo sincrono
8. Invio richiesta in modo asincrono
9. Chiusura della connessione

**ServiceCallerInteraction: richiesta sincrona**

1. La richiesta in caso di WS è il numero
2. In ogni altro caso è un IApplMessage

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente**ServiceCallerInteraction: richiesta asincrona**

1. Nel caso di MQTT, preparo un oggetto che riceve la risposta
2. La richiesta in caso di WS è il numero
3. In ogni altro caso, la richiesta è un IApplMessage
4. Attesa della risposta

Tra le due chiamate, il client può eseguire altre azioni.

NOTA: Va notato (e approfondito) come, nel caso MQTT e WS, l’oggetto che realizza la connessione sia capace anche di predisporre un oggetto per gestire le risposte inviate dal servizio.

**UNIBO.BASICOMM23**

Il progetto intende realizzare il concetto di Interconnessione, fornendo supporti di tipo generale per le comunicazioni via rete mediante un insieme di protocolli. Protocolli punto-a-punto come TCP, HTTP, CoAP, etc. sono in grado di stabilire una connessione stabile sulla quale inviare e ricevere messaggi. Il protocollo UDP non stabilisce una connessione, ma può comunque essere usato per inviare messaggi e ricevere risposte.

**Interconnessione**

Nel seguito, useremo il termine Interconnessione nel senso astratto di canale di comunicazione bidirezionale tra due end-points. A partire da questo concetto, introdurremo un insieme di interfacce e classi

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente**Interfaccia Interaction**

Un oggetto che implementa l’interfaccia Interaction realizza in modo concreto il concetto astratto di Interconnessione usando uno specifico protocollo.

Ogni metodo rappresenta una modalità di invio (forward/reply/request) o di ricezione (receiveMsg) di informazione dal punto di vista applicativo. La creazione di oggetti di questo tipo è affiadata alla ConnectionFactory.

**L’informazione scambiata**

L’informazione scambiata è rappresenta in due modi:

1. con una String, che è un tipo di dato presente in tutti i linguaggi di programmazione. I metodi che usano String non impongono particolari vincoli sulla struttura sintattica e semantica della stringa-messaggio, ma proprio per questo possono causare problemi di interpretazione;
2. con un oggetto di tipo IApplMessage, che rappresenta uno standard interno sulla struttura delle informazioni scambiate via rete. I metodi che usano IApplMessage verranno implementati da supporti che invieranno e riceveranno String della forma descritta in Struttura dei messaggi.

**Metodi per la comunicazione**

Il metodo forward è un metodo di trasmissione ‘fire-and-forget’, mentre il metodo request denota l’invio di informazione cui deve corrispondere una ack o una response da parte del ricevente.

Concettualmente, un componente che invia una response/ack sulla comunicazione connessa con un client, effettua una operazione di reply, assimilabile alla forward di un messaggio con appropriato contenuto informativo.

La String restituita dal metodo receiveMsg rappresenta un messaggio qualsiasi, tra cui la risposta a un messaggio inviato in precedenza con forward.

Il metodo receive restituisce un messaggio organizzato secondo la struttura dei messaggi vista precedentemente che costituisce uno standard interno sulla struttura delle stringhe-messaggio scambiate via rete.

[Per inviare un messaggio bisogna conoscere: indirizzo, porta, protocollo, formato\_messaggio.]

**Natura e struttura dei messaggi applicativi**

I messaggi sono logicamente suddivisi in diverse categorie:

enum ApplMessageType{event, dispatch, request, reply, invitation}

* Immagine che contiene Carattere, testo, calligrafia, diagramma

  Descrizione generata automaticamentedispatch: un messaggio inviato a un preciso destinatario senza attesa di una risposta (in modo detto anche fire-and-forget);
* invitation: un messaggio inviato a un preciso destinatario aspettandosi un ‘ack’ da parte di questi;
* request: un messaggio inviato a un preciso destinatario aspettandosi da parte di questi una reply logicamente correlata alla richiesta;
* event: un messaggio inviato a chiunque sia in grado di elaborarlo.

[poi ha interrotto il ppt]

**ProdConsSystem**

Ottimo template iniziale per un’analisi dei requisiti e del problema. Non lo ricopio perché è scritto bene ed è una linea guida, non ci sono nozioni generali da segnarsi. Riprendo solamente la fine della pagina, dove ci sono nozioni un po’ più generali:

**In sintesi:**

* Il concetto Connessione permette di nascondere a livello applicativo i dettagli del protocollo, ma si muove nello spazio concettuale dei sistemi client-server (il Produce è un client, il Consumer è il server)
* il client può eseguire in modo esplicito un metodo receive / receiveMsg (bloccante) per acquisire informazione dal server
* il Consumer può interagire con un Producer solo dopo che questi lo ha chiamato. Il Consumer non conosce quanti Producer ci sono o ci possono essere nel sistema.

[Passando da naive a interaction deve sparire la parola socket, ma dal lato server deve comparire un concetto più generale di connessione e quindi devo usare una libreria personalizzata per esprimere quello che mi serve , invece dal lato client deve comparire un qualcosa (definito nella libreria) che mi apra un server su tcp (in questo caso) e come gestire il messaggio (handler, oggetto attivo perché posso ricevere più cose (se fosse passivo, appena riceve una cosa muore)) (quindi definire un oggetto in libreria)]

**TestPlans per ProdCons**

Il Testing è una delle attività principali nello sviluppo del software.

Ricordiamo che, nella fase di analisi, l’obiettivo non è quello di eseguire dei test, ma quello di definire i test in modo che possano individuare in modo automatico le falle del software rispetto ai requisiti.

[Nel test plan: Come producer farò questo e mi aspetto questo <- da fare sempre perché deriva dai requisiti]

L’obiettivo di definire un Piano di Testing (TestPlan) automatizzabile, introduce di fatto un nuovo requisito: **rendere l’applicazione osservabile**, o meglio, renderne verificabili, da parte di una macchina, gli effetti dell’applicazione.

[Per fare il test plan scriviamo una user story in linguaggio naturale e poi creiamo i test adatti. I test li posso scrivere anche senza aver scritto i componenti.]

[Se invio a qualcuno la richiesta è osservabile perché qualcosa la fa e mi da una risposta, se gli invio un dispatch non è osservabile perché dovrei vedere cosa fa il consumer. Per il test plan della request è facile perché è osservabile, mentre la dispatch mi causa dei problemi perché non è osservabile. Come faccio a sapere se l’ha ricevuto e se ha fatto quello che doveva fare? Uso i logfile. Cerco di fare scrivere al consumer sul file di log così io producer lo posso vedere]

user story: an informal, natural language description of features of a software system

Enablers: classi che aiutano la comunicazione

Alieno: programma che non sa niente del nostro mondo (il nostro codice custom)

Enunciare i TestPlans a parole è un primo passo, utile ma non sufficiente. Ciò che occorre è esprimere un TestPlan in modo che sia interpretabile ed eseguibile in modo automatizzato da un computer. Per raggiungere questo obiettivo sono stati sviluppati strumenti appositi.

Uno strumento molto diffuso è JUnit, ideato da Beck e Gamma come supporto al concetto di collaudo continuo nell’ambito di processi di sviluppo agili e incrementali.

[JUnit è usato per eseguire in maniera automatica tutti i test scritti]

Osservazione: il codice che scrive in un Log-file all’intero di un componente. Per evitarlo, si potrebbe introdurre un opportuno supporto, oppure, ancora meglio, fare ricorso al Observer pattern.

**Un primo passo verso gli attori**

È il momento di introdurre un nuovo spazio concettuale, in cui Consumer e Producer sono componenti di una stessa categoria (gli Actor), in cui non occorre più parlare di client e server, ma solo di informazioni trasmesse, informazioni ricevute e capacità di elaborare le informazioni ricevute.

* Automessage per far partire l’attore (autostartmsg)
* Thread importante a livello di tracing
* Dispatch: fire and forget

**QAK ACTORS 24**

Secondo Carl Hewitt (uno dei padri fondatori) il modello computazionale ad attori è stato ispirato, a differenza dei precedenti modelli di calcolo, dalla fisica, inclusa la relatività generale e la meccanica quantistica. Vi è oggi una ampia gamma di proposte di linguaggi / librerie ad attori, tra cui:

* Akka: ispirato a Modello computazionale ad attori di Hewitt. Per le motivazioni si veda Akka actors
* GO: ispirato a CSP, propone goroutine e CanaliGO. Per la documentazione si veda GO doc
* Kotlin Actors: propone croutines e channels (si veda Kotlin channel)

Un motto di riferimento alquanto significativo per questo modello è il seguente:

Do not communicate by sharing memory … instead, share memory by communicating.

**QakActors24: Introduzione**

La Q/q nella parola QActor, significa “quasi” poiché il linguaggio non è inteso come un linguaggio di programmazione generico, ma piuttosto un linguaggio di modellazione eseguibile, da utilizzare durante l’analisi del problema e il progetto di prototipi di sistemi distribuiti, i cui componenti sono attori che si comportano come un Automa a stati finiti, in stretta relazione con l’idea di sistemi basati su Microservizi.

L’aggiunta di k al prefisso (es qak, Qak) significa che stiamo facendo riferimento alla versione implementata in Kotlin, senza utilizzare i supporti Akka (come fatto nella prima versione del linguaggio).

Per una Introduzione all’uso di Kotlin si veda: si veda: KotlinNotes.

**Quadro generale**

Un attore qak specializza la classe astratta it.unibo.kactor.ActorBasicFsm.kt che a sua volta specializza la classe astratta it.unibo.kactor.ActorBasic.kt, entrambe definite nella Qak infrastructure.

È possibile costruire un sistema software basato su attori qak semplicemente usando queste librerie. Tuttavia, l’uso della Qak software factory e del connesso Linguaggio qak rende lo sviluppo dei sistemi molto più rapido, comprensibile e gestibile.

**Qak software factory**

Il Linguaggio qak è definito utilizzando il framework Xtext, che permette di costruire un insieme di plugin per l’ecosistema Eclipse che, una volta installati, permettono ad un application designer de realizzare in tempi rapidi un modello eseguibile del sistema.

L’application designer usa l’editor guidato dalla sintassi per scrivere un modello del sistema che definisce struttura, interazione e comportamento di un sistema distribuito. Il modello è una istanza de Il metamodello Qak, sulla base del quale è costruita la Factory. Una volta salvato il modello, la factory produce codice e risorse.

**Qak codice e risorse generate**

La Qak software factory costruisce vari prodotti indispensabili o utili, tra cui:

* un file che contiene la descrizione del sistema, in sintassi Prolog
* il file build2024.gradle e altre risorse
* il codice di raccordo con la Qak infrastructure (la parte sommersa di ogni sistema Qak)
* il codice Python per la produzione di una rappresentazione grafica del sistema

**Qak infrastructure**

La libreria unibo.qakactor23-5.0.jar è prodotta nel progetto unibo.qakactor23 e costituisce la qak-infrastructure, che si appoggia al supporto unibo.basicomm23-1.0.jar introdotto nel progetto unibo.basicomm23, che implementa il concetto astratto di Interaction per diversi protocolli (TCP, UDP, CoAP, etc.).

**Il metamodello Qak**

Il Linguaggio qak reso disponibile dalla Qak software factory intende fornire un linguaggio per la definizione di modelli eseguibili di un sistema, basati su un insieme di concetti volti a catturare l’idea che un sistema software (distribuito):

è formato da un insieme di attori che si comportano come Automi a stati finiti 🡪 che interagiscono scambiandosi messaggi 🡪 raggruppati in contesti che li abilitano a interazioni via rete 🡪 contesti che possono essere allocati (deployed) su uno o più nodi computazionali

**QakActors24: il sistema**

Immagine che contiene Arte bambini, disegno, viola, rosa

Descrizione generata automaticamenteUn sistema ad attori qak è composto da una collezione di attori, attivati in uno o più contesti, allocati in uno o piò nodi di elaborazione.

Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamenteUn sistema ad attori qak è configurato in modo automatico a partire da una descrizione espressa in forma di base di conoscenza, in sintassi Prolog

**QakActors24: l’attore**

Un attore qak è un componente **attivo** che:

* **nasce, vive e muore in un contesto** che può essere comune a (molti) altri attori;
* ha un **nome univoco** nell’ambito di tutto il sistema;
* è logicamente attivo, cioè dotato di flusso di controllo autonomo;
* **è capace di inviare messaggi ad un altro attore**, di cui conosce il nome, incluso sè stesso;
* **è capace di eseguire elaborazioni autonome e/o elaborazioni di messaggi**;
* **è dotato di una sua coda locale (msgQueue)** in cui sono depositati i messaggi a lui inviati
* Elabora i messaggi secondo quanto riportato in La gestione dei messaggi.

**QakActors24: il contesto**

Immagine che contiene testo, calligrafia, Carattere, clipart

Descrizione generata automaticamenteUn contesto è un componente software che gestisce N>0 actor qak, **abilitandoli alla ricezione e trasmissione** di messaggi via rete. **Un contesto rappresenta un nodo logico** di elaborazione dotato di un server e di una porta di ingresso, su cui altri contesti possono stabilire una Interconnessione, di solito basata su TCP, CoAP e MQTT. Un contesto deve essere allocato su un computer fisico o su un virtual macìhine / container.

Immagine che contiene testo, diagramma, calligrafia

Descrizione generata automaticamenteUn contesto **mantiene una tabella** (actorMap) **con i riferimenti agli attori locali** e una tabella (proxyMap) con i **riferimenti ai Proxy** che mantengono una Interconnessione con gli altri contesti del sistema.

Il Server di contesto depone i messaggi IApplMessage ricevuti su una Interconnessione sulla msgQueue dell’attore destinatario. Per questo scopo, il Sever si avvale di un unico gestore di messaggi di sistema: il ContextMsgHandler.

La figura mostra il caso di attori locali ad un nodo di elaborazione che possono inviare/ricevere messaggi tra loro oppure elaborare messaggi inviati da componenti remoti.

**Linguaggio qak**

Il linguaggio si pone nel solco dei Domain Specific Languages e permette di esprimere un insieme dei concetti che forma Il metamodello Qak. Il ruolo ‘strategico’ dei linguaggi in informatica si comprende subito considerando che ogni computazione (ogni sistema software) può essere espressa usando un insieme molto limitato di ‘mosse’ (istruzioni) studiate dalla teoria come macchine astratte elementari.

In sintesi, possiamo dire che l’uso di un linguaggio comporta descrizioni di un sistema software più compatte, più esplicite e semanticamente più ricche

**Il linguaggio qak intende promuovere la definizione in tempi brevi di prototipi di sistemi distribuiti, utilizzabili nelle fasi preliminari di un progetto di sviluppo software, al fine di interagire con il committente, per chiarire e stabilizzare i requisiti.**

In molti casi, la formalizzazione dei requisiti e della analisi del problema in termini di modelli eseguibili qak costituisce anche un passo pragmaticamente utile per la costruzione effettiva del prodotto finale.

**Qak syntax**

La sintassi del linguaggio è riportata in Qak syntax.

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteLo ‘scopo’ della grammatica è la produzione relativa alla specifica del sistema.

Specifica del sistema:

1. nome del sistema
2. indirizzo di un broker MQTT
3. dichiarazione di una o più librerie applicative
4. Dichiarazione dei messaggi
5. Dichiarazione dei contesti
6. Dichiarazione degli attori
7. Dichiarazione di un Display di sistema
8. Dichiarazione di una Facade di sistema

Le regole sintattiche del linguaggio impongono che un modello Qak venga definito organizzando la sua descrizione in **una sequenza di dichiarazioni**.

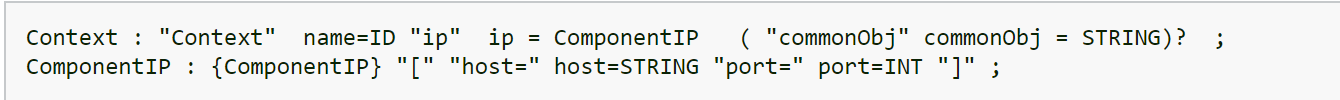
**Dichiarazione dei messaggi**

I diversi Tipi di messaggi sono dichiarati usando una sintassi Prolog-like (si veda tuProlog):

Immagine che contiene testo, schermata, Carattere, algebra

Descrizione generata automaticamente

**Dichiarazione dei contesti**



Un contesto può introdurre un **oggetto accessibile a tutti gli attori**.

**Dichiarazione degli attori**

**Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente**La sintassi indica che vi sono tre tipi di attori.

* Attori normali
* Attori coded
* Attori external

**Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteAttori normali**

Gli attori ‘normali’ sono descritti come Automi a stati finiti.

1. Nome dell’attore
2. Riferimento al Contesto
3. Oggetto locale usato dall’attore
4. Attore creato solo dinamicamente
5. Librerie importate
6. Azioni iniziali dell’attore
7. Stati dell’attore
8. Dichiarazione dell’oggetto locale
9. Dichiarazione delle librerie importate

Stati di un attore normale

1. Nome dello stato
2. Stato iniziale. Il tag initial deve essere presente in un unico stato.
3. Azioni locali allo stato
4. Transizioni verso lo stato futuro

**Transizioni di stato**

La transizione da uno stato a uno stato successivo può avvenire senza attesa di alcun messaggio (EmptyTransition) oppure (NonEmptyTransition) in relazione alla disponibilità di un messaggio tra quelli definiti in Dichiarazione dei messaggi.

**EmptyTransition - Guardia**

Una transizione associata a una guardia viene attivata solo se la valutazione della condizione espressa dalla guardia produce il valore true.

**NonEmptyTransition**

Una NonEmptyTransition associata alla disponibilità di un messaggio distingue tra i diversi tipi di messaggio (Dispatch, Request, Reply, Event)

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

**whenInterrupt**

Esegue la transizione da uno stato SA a uno stato SB, con ritorno allo stato SA, quando SB esegue l’istruzione qak

**Timeout per transizioni**

Per evitare una attesa indefinita di messaggi in uno stato, è possibile associare alla transizione un timeout (come un numero naturale in msec) scaduto il quale l’automa transita nello stato specificato.

Lo scadere del tempo indicato in whenTime (regola TimeoutInt) provoca l’emissione di un evento, con identificatore local\_tout\_actorname\_state ove actorname è il nome dell’attore e state è il nome dello stato corrente

Le forme che si aggiungono a TimeoutInt sono utili in situazioni in cui il tempo non sia noto a priori, ma derivi da elaborazioni.

**Comportamento di un attore**

Prima di illustrare cosa un attore qak può fare, è importante sottolineare **che un attore qak non dispone di una operazione receive bloccante.** La ragione è dovuta al comportamento message-driven dell’attore.

Infatti, il comportamento di base di un attore qak è definito dalla classe it.unibo.kactor.ActorBasic.kt che gestisce i messaggi disponibili sulla msgQueue dell’attore in modo FIFO; l’attore qak di base opera quindi in modo message-driven utilizzando un canale Kotlin.

L’attore che specializza it.unibo.kactor.ActorBasicFsm.kt opera invece come un automa di Moore a stati finiti, gestendo i messaggi ricevuti sul canale Kotlin ereditato da ActorBasic non in modo FIFO, ma **in funzione dello suo stato corrente**.

[FSM= final state machine]

Un attore qak ha un comportamento autonomo e quindi, una volta attivato dalla Qak infrastructure con un messaggio iniziale di ‘start’, può eseguire azioni anche ignorando eventuali altri messaggi sulla sua coda di input.

Normalmente però, un attore qak può entrare in specifici stati elaborativi se sulla sua coda di input sono presenti messaggi di un certo tipo.

[Un attore è vivo e per questo fa quello che gli pare (grazie alla possibilità di usare codice kotlin al suo interno), oltre a fare quello che gli pare vede se qualcuno gli ha messo un messaggio in coda. Può fare le due cose contemporaneamente? No

Se ha messaggi accede alla coda e legge il PRIMO messaggio della coda (FIFO). Io però nella realtà opero per priorità, quindi l’attore leggerà determinati messaggi in determinati **stati** per leggere il messaggio che gli interessa. Una volta trovato e letto, **ENTRO IN UN NUOVO STATO** e avrò una nuova coda di messaggi

Attore normale per forza FIFO, attore fsm decide LUI il messaggio da leggere in base allo stato]

[I nostri attori sono intrinsecamente osservabili (perché li abbiamo definiti così)]

**La gestione dei messaggi**

1. ogni attore possiede, oltre alla coda dei messaggi principale msgQueue , una seconda coda msgQueueStore in cui memorizza messaggi (di tipo request e dispatch) non elaborati;
2. uno stato è di norma associato a un insieme di transizioni TSET, ciascuna delle quali specifica lo stato futuro, in corrispondenza a un messaggio con uno specifico identificatore msgId;
3. al termine delle sue azioni, lo stato corrente dell’attore qak consulta, nell’ordine, le sue code msgQueueStore e msgQueue, ciascuna in modo FIFO;
4. se l’identificatore del messaggio prelevato da una coda è uguale al msgId di una qualche transizione in TSET, quella transizione è attivabile; in caso contrario, il messaggio ‘esaminato’ viene lasciato dove è se era nella coda msgQueueStore, oppure, se è un messaggio di tipo request o dispatch prelevato dalla coda principale msgQueue, viene depositato in fondo alla coda msgQueueStore.

Un messaggio di tipo event il cui identificatore non compare in TSET, viene scartato (e quindi ignorato e dimenticato);

1. appena lo stato corrente trova una transizione attivabile, passa il controllo allo stato futuro specificato da questa transizione;
2. se nessuna transizione è attivabile, l’attore qak rimane nello stato corrente; all’arrivo di un nuovo messaggio, si riprende ad eseguire il punto 3.

**Variabili e riferimenti**

simboli $ # @ sono Notazioni Shortcut per l’accesso al valore di variabili i cui nomi iniziano con una lettera MAIUSCOLA

* Notazione $ Kotlin-like per accesso al valore di una variabile entro una String
* Notazione # per accesso al valore in forma di String al valore di una variabile della soluzione di una dimostrazione logica
* Notazione @ per accesso al valore di una variabile della soluzione di una dimostrazione logica

**Azioni di un attore**

**[**Per sapere cosa può fare l’attore qak senza ricorrere a kotlin devo leggere la documentazione della sintassi (le frasi ammesse nel linguaggio, che sono infinite numerabili)]

Una volta entrato in un particolare stato computazionale, un attore qak può eseguire una sequenza di azioni di ‘alto livello’ espresse in linguaggio qak oppure di ‘basso livello’ espresse direttamente in Kotlin.

1. Immagine che contiene testo, schermata, Carattere, numero

   Descrizione generata automaticamenteAzioni kotlin
2. Operazioni di messaggista punto a punto
3. Azioni condizionali
4. Operazioni di messaggista publisch-subscribe
5. Operazioni relative agli eventi
6. Operazioni relative alla osservabilità
7. Operazioni di delegazione
8. Operazioni per le basi di conoscenza
9. Operazioni di creazione di attori
10. Operazioni di ritorno da interruzione
11. Operazioni per esecuzione di codice
12. Operazioni di utilità
13. Operazioni con il tempo
14. Operazioni di terminazione

**AnyAction**

Le azioni esprimibili nel linguaggio qak non danno un linguaggio computazionalmente completo.

Pertanto, volendo rendere eseguibile un modello qak, si introduce la possibilità che un attore qak possa esprimere una qualunque sequenza di azioni scritte in Kotlin.

[# … #]: Specifica di codice Kotlin.

**Operazioni di messaggistica punto a punto**

Le operazioni di invio messaggio sono le seguenti:

* forward: Invio di Dispatch.
* request. Invio di Request
* replyTo. Invio di Reply a una Request.
* ask. Invio di Request a un attore che ha fatto una Request.
* autodispatch. Invio di Dispatch di un attore a sè stesso.
* autorequest. Invio di Request di un attore a sè stesso.

*Le altre possibili azioni di un attore non saranno riportate in questo documento, guardare perciò il file html per ulteriori informazioni sulle altre azioni*

[Operazioni di delegazione -> possibilità di delegare i messaggi a un altro attore . Problema se io faccio una richiesta a un determinato attore devo capire se il delegato risponde a me o al delegante

Per inviare i messaggi devo conoscere il nome dell’attore destinatario]