QakActors25Intro

QakActors25Index

QakActors25 note storiche

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

QakActors25: 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 protitpi 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*.

E' possibile costruire un sistena 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.

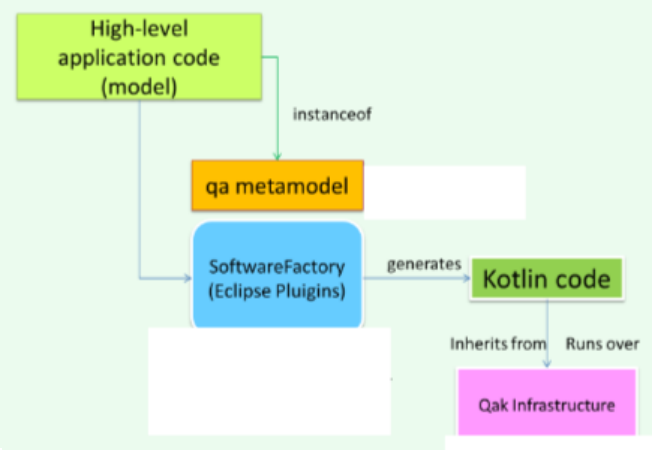
I plugin della Qak factory

I plugin che, installati in Eclipse, realizzano la *Qak software factory* sono:

- **it.unibo.Qactork.ide_1.6.0.jar**
- **it.unibo.Qactork.ui_1.6.0.jar**

- it.unibo.Qactork_1.6.0.jar

Essi sono disponibili in (issLab25/iss25Material/plugins).



L'application designer usa l'editor guidato dalla sintassi per scrivere un *modello* del sistema che definisce

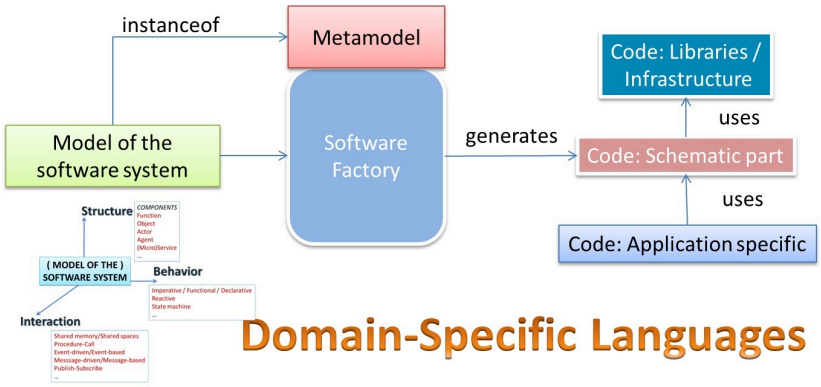
(struttura, interazione e comportamento)

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*.

Questa impostazione è tipica delle software factories relative ai *Domain Specific Languages* che si pongono nel solco della *Model driven engineering*>.



La factory genera il codice necessario per 'mappare' i concetti di alto livello (espressi nel modello) in strutture di più basso livello, che possono essere eseguite avvelendosi di appositi supporti e librerie di utilità application-independent.

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 **build2025.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.).

Per maggiori dettagli, s veda: *QakActors25Implementazione*

Il metamodello Qak

Il *Linguaggio qak* reso disponibile dalla *Qak software factory* intende fornire un linguaggio per la definizione di

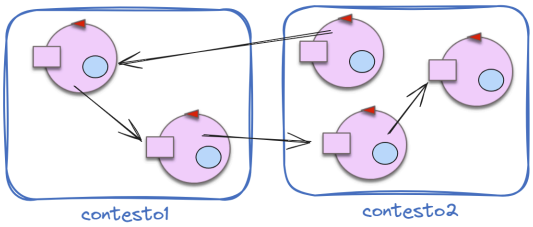
(modelli eseguibili) di un sistema,

basati su un insieme di :reamrk: `concetti volti a cattuare l'idea ` che un sistema software (distribuito) possiede le seguenti caratteristiche:

- il *sistema* è formato da una insieme di *attori*
- gli attori interagiscono scambiandosi messaggi (di tipo *IApplMessage*, nel nostro caso)
- un attore è un ente autonomo, capace di elaborare messaggi. Pertanto la struttura del codice di un attore si presta ad essere modellata come un *Automi a stati finiti*
- gli attori sono raggruppati in *contesti* che li abilitano a interazioni via rete
- i contesti possono essere allocati (deployed) su uno o più nodi computazionali, fisici o virtuali

QakActors25 - il sistema

Un sistema ad attori qak è composto da una collezioni di attori, attivati in uno o più contesti, allocati in uno o piò nodi di elaborazione.



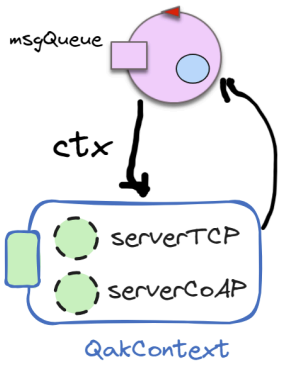
Un sistema ad attori qak è configurato in modo automatico a partire da una descrizione espressa in forma di *base di conoscenza*, in sintassi Prolog.

```
context(ctx1, "localhost",  "TCP", "8923").
context(ctx2, "localhost",  "TCP", "8925").
qactor( producer1, ctx1, <className>).
qactor( consumer,  ctx2, <className>).
qactor( producer2, ctx3, <className>).
...
```

QakActors25 - 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



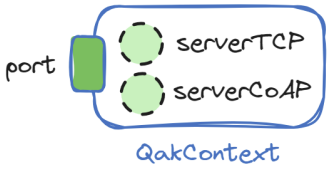
- Elabora i messaggi secondo quanto riportato in *La gestione dei messaggi*.

QakActors25 - il contesto

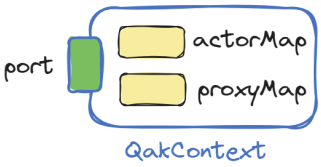
Un contesto è un componente software che gestisce **N>0** actor qak, **abilitandoli** alla ricezione e trasmissione di messaggi via rete.

Un contesto rappresenta un nodo logico di elaborazione dotato di un server e di un 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.

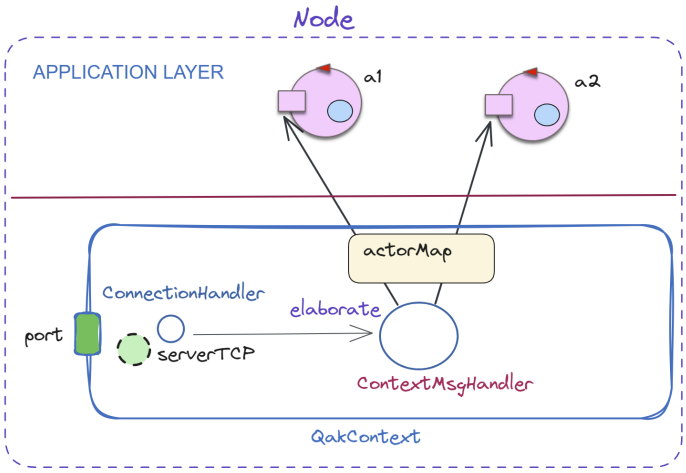


Un contesto mantiene una tabella (*actorMap*) con i riferimento agli attori locali e una tabella (*proxyMap*) con i riferimenti ai Proxy che mantengono una *Interconnessione* con gli altri contesti del sistema.



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.

Indice: *QakActors25Index*