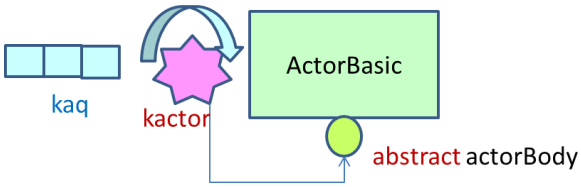
QakActors25Implementazione

Note sulla implementazione

it.unibo.kactor.ActorBasic.kt

Realizza il concetto di un ente computazionale dotato di flusso di controllo autonomo, capace di ricevere e gestire messaggi in modo FIFO, sfruttando un *Kotlin actor* incapsulato:

```
/*1*/ abstract class  ActorBasic(
/*2*/   name: String,
/*3*/   val scope:CoroutineScope=GlobalScope,
/*4*/   var discardMessages Boolean=false,
/*5*/   val confined :    Boolean = false,
/*6*/   val ioBound :     Boolean = false,
/*7*/   val channelSize : Int = 50
        ) :
/*8*/     CoapResource(name),
/*9*/      MqttCallback {
      ...
    //To be overridden by the application
/*10*/ abstract suspend fun actorBody(
          msg:IApplMessage)
  }
```



Si veda: *actor channel*

1. class ActorBasic Si veda *Oggetti e classi* in *KotlinNotes*.
2. name Nome (**univoco** nel sistema) dell'attore
3. scope Si veda *Le coroutines* in *KotlinNotes* e *kotlinUniboCoroutinesIntro* in *kotlinUnibo*.
4. discardMessages scarta o meno i messaggi non attesi. Usato principalmente in *ActorBasicFsm*
5. confined Si veda *Confinamento* in *KotlinNotes*.
6. ioBound Si veda *Confinamento* in *KotlinNotes*.
7. channelSize Si veda *I canali* in *KotlinNotes*.
8. CoapResource Si veda *Estende CoapResource*
9. MqttCallback Si veda *Implementa MqttCallback*
10. actorBody codice per la gestione dei messaggi *IApplMessage* ricevuti dall'attore.

La notazione:

```
class  ActorBasic( ... ) : CoapResource(name), MqttCallback
```

esprime in forma compatta che *ActorBasic* eredita dalla classe *CoapResource* e implementa l'interfaccia *MqttCallback* (si veda *kotlinInheritance*).

Estende CoapResource

Ogni attore è anche una risorsa CoAP, specializzazione della classe definita nella libreria *https://www.eclipse.org/californium/*.

Implementa MqttCallback

Ogni attore implementa anche l'interfaccia **org.eclipse.paho.client.mqttv3.MqttCallback**. Pertanto ogni attore può gestire notifiche emesse da un MQTT client, attraverso il metodo **messageArrived**. (TODO REF)

actor channel

```
val actor = scope.actor<IApplMessage>( dispatcher, capacity=channelSize ) {
  for( msg in channel ) {
                    if( msg.msgContent() == "stopTheActor") {  channel.close() }
    else actorBody( msg )
  }
}
```

Si veda: *Kotlin actor* in *KotlinNotes*.

sendMessageToActor

Il metodo **sendMessageToActor** realizza l'invio di un messaggio ad un attore di cui è noto il nome o la connessione.

```
suspend fun sendMessageToActor(msg : IApplMessage,
       destName: String, conn : Interaction? = null ) {
  //realizza l'invio di msg all'attore di nome destName
  //usando conn se conn!=null (destname è un 'alieno')
  val destactor = context!!.hasActor(destName)
  /*
    se destactor  è locale: destactor.kactor.send( msg )
    altrimenti usa il proxy verso il contesto di destactor
}
```

it.unibo.kactor.ActorBasicFsm.kt

```
abstract class ActorBasicFsm(  qafsmname:  String,
                    fsmscope: CoroutineScope = GlobalScope,
            discardMessages : Boolean = false,
                    confined :    Boolean = false,
                    ioBound :     Boolean = false,
                    channelSize : Int = 50
): ActorBasic(qafsmname,fsmscope,discardMessages,confined,ioBound,channelSize) { ... }
```

- Un attore che specializza questa classe opera come un automa a stati finiti.
- Il codice Kotlin viene generato dalla *Qak software factory*
- I messaggi ricevuti sul canale Kotlin (ereditato da *ActorBasic*) sono gestiti in relazione alle specifiche sulle transizioni associate allo stato corrrente dell'automa.

NEXT:

- *RiassuntoChatGpt*
- *QakActors25Demo*

Qak sysUtil

La classe **sysUtil** della infrastruttura offre un insieme di metodi di utilità:

metodi di supporto

suspend autoMsg(msg:IApplMessage)	realizza le operazioni **AutoMsg** e **AutoDispatch** desctitte in *Operazioni di messaggista punto a punto*
open public fun fromRawDataToApplMessage(m: String)	deve essere realizzato dall'application designer per trasformare una stringa in un messggio di tipo *IApplMessage* e per eseguire un autoMsg di questo messaggio

metodi di utilità

1. **curThread() : String**	1. thread corrente in cui si svolge la esecuzione
2. **aboutThreads(info: String)**	2. informazioni sui thread in esecuzione
3. **strRepToList(liststrRep: String) : List<String>**	3. da stringa a lista
4. **strCleaned(s : String) : String**	4. pulizia di scringa
5. **showOutput(proc: Process)**	5. visualizzazione
6. **waitUser(prompt: String,tout: Long=2000)**	6. attesa di input da utente
7. **createFile(fname:String,dir:String ="logs")**	7. creazione di un file
8. **deleteFile(fname : String, dir : String)**	8. eliminazione di un file
9. **updateLogfile(fname:String,msg:String,dir:String="logs")**	9. aggiornamento file
10. **getMqttEventTopic() : String**	10. topic corrente di contesto

metodi per la base di conoscenza

1. **getPrologEngine() : Prolog**	1. riferimento all'interprete Prolog locale
2. **solve(goal:String, resVar:String):String?**	2. solve di un goal
3. **loadTheory(path: String)**	3. caricamento di una teoria nella kb corrente dell'actor
4. **loadTheoryFromDistribution(path: String)**	4. caricamento di una teoria

---

metodi di sistama

1. **getActorNames(ctxName:String):List<String>**	1. lista dei nomi degli attori del contesto
2. **getAllActorNames(ctxName: String) : List<String>**	2. lista dei nomi di tutti gli attori del contesto
3. **getAllActorNames( )**	3. lista dei nomi di tutti gli attori del sistema
4. **getNonlocalActorNames(ctx:String):List<String>**	4. lista dei nomi non locali al contesto dell'actor
5. **getActor(actorName : String) : ActorBasic?**	5. riferimento ad un actor, dato il suo nome
6. **getContext(ctxName : String) : QakContext?**	6. riferimento ad un contesto, dato il suo nome
7. **getContextNames(): MutableSet<String>**	7. nome dei contesti
8. **getActorContextName(actorName:String):String?**	8. nome del contesto di un actor di un dato nome
9. **getActorContext(actorName : String):QakContext?**	9. riferimento al contesto di un actor di un dato nome
10. **getCtxCommonobjClass(ctxName:String): String**	10. nome della classe di un contesto

---

Indice: *QakActors25Index*