22/06/2020

PIAZZA_OVERVIEW INIZIALE

Introduzione

Lo scopo di questa overview è quello di iniziare ad affrontare il progetto finale del corso acquisendo sin da subito una visione d'insieme del problema. A partire da questa potrò impostare un workplan basato su un approccio di sviluppo incrementale ispirato alla metodologia SCRUM.

Dopo aver letto i requisiti del problema riporto in questa overview una primissima fase di Analisi dei Requisiti (con TestPlan), e di Analisi del Problema tentando di affrontare il problema nella sua interezza ma con un alto livello di astrazione.

NOTA: Questa overview iniziale verrà fatta in linguaggio naturale e con l'ausilio di alcune tabelle e/o immagini. Sono consapevole però di quanto sia ambiguo il linguaggio naturale e di quanto sia difficile, se non addirittura impossibile in certe situazioni, tentare di formalizzare un requisito con il linguaggio naturale.

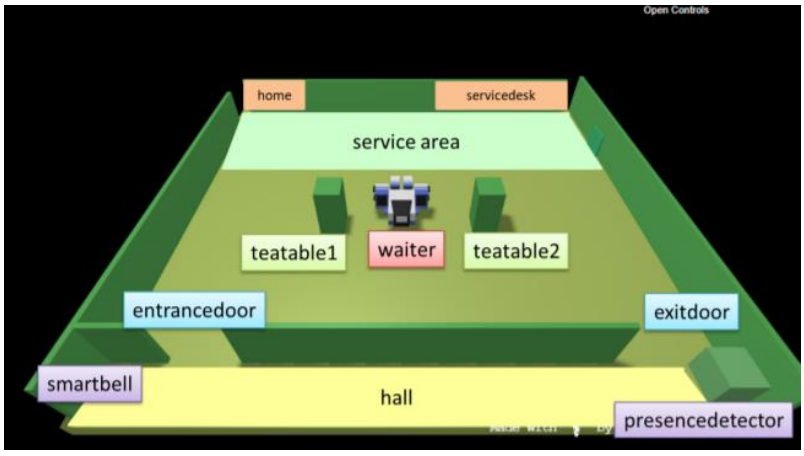
È per questo motivo che, già al termine di questa overview, mi impegnerò a produrre come output un primo modello formale (eseguibile) del sistema.

Requisiti

Si veda la descrizione completa dei requisiti.

Analisi dei Requisiti

L'obiettivo del progetto è quello di realizzare un sistema software per gestire una TeaRoom ai tempi del Covid-19. Tra i componenti di questo sistema vi è un ddr (Differential Drive Robot) cameriere, detto **waiter**, che si troverà a dover gestire i clienti e la sala.



[Figura n.1]

Gli elementi appartenenti alla tea-room sono:
-teatable1 -teatable2

Le posizioni significative della tea-room sono:

-home
-servicedesk
-entrancedoor
-exitdoor

Le entità individuabili dai requisiti sono:
-waiter
-barman
-smartbell
-presencedetector
-client
-manager

VOCABOLARIO DEI TERMINI

Termine	Significato inteso dal committente	Altre note
waiter	È un differential drive robot che ha il ruolo di cameriere della tea-room.	
tearoom	È una stanza dal perimetro rettangolare e le cui pareti sono libere da ostacoli. La tea-room deve essere *safe*, ossia popolata solo da clienti con T°<37.5, contenente dei tavoli opportunamente distanziati e i quali vengono sanificati dopo ogni consumazione.	Vedi Figura n.1 per conoscere la pianta della tea-room.
home	Posizione della tea-room da cui il robot inizia a lavorare e in cui ritorna quando non ha nulla da fare.	
entrancedoor	Posizione della tea-room da cui entrano i clienti.	
exitdoor	Posizione della tea-room da cui escono i clienti.	
teatable1	Tavolo n.1 nella tea-room in cui un cliente può sedersi e consumare il proprio tea.	È un ostacolo per il waiter.
teatable2	Tavolo n.2 nella tea-room in cui un cliente può sedersi e consumare il proprio tea.	È un ostacolo per il waiter.
barman	È l'entità addetta a ricevere le ordinazioni di tea provenienti dal waiter e a prepararle.	
servicedesk	Posizione della tea-room in cui si trova il barman.	
service area	Zona della tearoom in cui si trovano la home e la servicedesk.	
smartbell	È un campanello situato in prossimità della entrancedoor tramite il quale i clienti possono notificare al waiter il loro interesse ad entrare nella tea-room. È in grado di misurare la temperatura corporea del cliente che desidera entrare e assegna al cliente un clientidentifier.	
clientidentifier	Si tratta di un identificatore univoco.	
hall	Zona adiacente alla tea-room nella quale i clienti possono transitare per chiedere di entrare nella tea-room o per defluire dopo esserne usciti.	
presencedetector	È un dispositivo (un sonar) che rileva la presenza di una persona (o di qualche entità) nella hall.	
maxstaytime	Tempo massimo in cui un cliente può consumare il tea se non c'è nessun altro tavolo tableclean.	
maxwaitingtime	Tempo massimo che un cliente deve aspettare prima di poter entrare nella tea-room se tutti i teatable sono occupati.	
current situation of the TeaRoom	Dopo aver chiesto esplicitamente al committente cosa intendesse è emerso che il manager è interessato a: -stanza piena/vuota -stato dei tavoli -stato del waiter -n° di clienti serviti, respinti e informati	

In futuro potrebbero interessare anche altre informazioni significative riguardo allo stato corrente della stanza, da cui magari prevedere lo stato futuro (tramite tecniche di Data Mining). Sarebbe interessante per un manager poter prevedere la situazione dei giorni successivi per poter ad esempio valutare se assumere un ulteriore waiter.

TABELLA CON I TASK DEL WAITER

Task	Descrizione	Entità coinvolte	Interrompibile*
accept	Se c'è almeno un teatable nello stato tableclean il waiter risponde in modo affermativo alla richiesta di un cliente di entrare.	waiter → smartbell	No
inform	Se non c'è nessun teatable nello stato tableclean il waiter informa il cliente del maximum waiting time.	waiter → cliente	No
reach	Il waiter raggiunge la entrance door.	waiter-cliente	No
convoyToTable	Il waiter accompagna il cliente al tavolo.	waiter-cliente	No
take	Il waiter prende l'ordine del cliente e lo trasmette al barman.	cliente → waiter → barman	No
serve	Quando l'ordine del clienteID è pronto il barman lo comunica allo waiter, che lo prende e lo porta al tavolo opportunuo.	barman → waiter → cliente	No
collect	Quando il cliente ha finito di consumare, o è passato maxstaytime il waiter si reca presso il tavolo cliente per farlo pagare.	waiter- cliente	No
convoyToExit	Il waiter accompagna il cliente alla exitdoor.	waiter-cliente	No
clean	Il waiter pulisce il tavolo.	waiter	Si
rest	Il waiter si reca alla home e vi resta fintanto che non ha nulla da fare.	waiter	Si

*colonna aggiunta in seguito all'analisi della problematica "ottimizzazione del lavoro del robot", sotto l'ipotesi che, per massimizzare la soddisfazione dei clienti, gli unici task interrompibili siano quelli non dedicati ai clienti.

FORMALIZZAZIONE DEI REQUISITI

Si veda una prima formalizzazione dei requisiti al seguente link: RequirementAnalysisModel.qak.

ALCUNI TESTPLAN FUNZIONALI SIGNIFICATIVI

NOTA: Al momento li esprimo in linguaggio naturale. Nei successivi sprint verranno formalizzati.

1. *testAccept*
   Scenario 1.1: TeaRoom con almeno un tavolo *N* nello stato tableclean. Arriva una richiesta da parte di un cliente con ClientID=CID.
   Al termine del task *accept* mi aspetto che il tavolo *N* si trovi nello stato busy(CID).
2. *testInform*
   TeaRoom con nessun tavolo nello stato tableclean e n_clienti_informati=n. Arriva una richiesta da parte di un cliente con ClientID=CID.
   Al termine del task Inform mi aspetto di avere n_clienti_informati=n+1.
3. *testReach*
   In seguito al task accept inizia il task reach.

Al termine di quest'ultimo mi aspetto che pos(waiter) = pos(entrancedoor).

4. Anche per testare i task *take, serve, collect, convoyToTable e convoyToExit* si può pensare di testare la posizione finale del waiter per assicurarci che si rechi al tavolo giusto, al service desk o alla exit door a seconda del task che stiamo testando.
5. *testClean*

   TeaRoom con almeno un tavolo *N* nello stato dirty. Al termine del task clean mi aspetto che il tavolo N sia nello stato tableclean.
6. *testRest*

   Se passa un tempo TimeToRest nel quale il waiter non ha nulla da fare mi aspetto che il waiter inizi a dirigersi verso la home. Gli scenari possibili sono due: il waiter raggiunge la home, oppure mentre tenta di raggiungerla viene interrotto da una richiesta.
7. *testRefuse*

   n_clienti_respinti=n e alla smartbell arriva la richiesta di un cliente con T°>=37.5. Mi aspetto che n_clienti_respinti=n+1 e che il waiter non venga disturbato.

Analisi del Problema

Di seguito riporto alcune problematiche che sono emerse, divise a seconda delle entità che sono coinvolte.

In un'ottica di sviluppo incrementale, nei successivi Sprint dovrò sicuramente riprendere alcune di queste problematiche per farne un'analisi più approfondita. In questo caso le citerò riferendomi al titolo con cui le ho descritte qui sotto.

Nota: Al momento non è stato preso in considerazione il requisito opzionale "one client in the hall" per mantenere il sistema più semplice. Verrà considerato in Sprint più avanzati.

Waiter

Countdown del maxStayTime : Il timer maxStayTime deve essere fatto partire nel momento in cui, al termine del task reach, il cliente ha raggiunto il tavolo e si siede per consultare il menù. Se venisse fatto partire dopo non avremmo il controllo della situazione perché un cliente potrebbe ad esempio prendersi tutto il tempo che vuole per ordinare.

Inoltre, dopo una discussione con il committente è emerso che vogliamo decurtare dal maxstaytime solo il tempo di attività imputabili al cliente. Quindi il timer verrà fatto scorrere mentre il cliente deve scegliere cosa ordinare e mentre il cliente sta consumando l'ordine. Verrà invece stoppato mentre il barman sta preparando l'ordine.
Allo scadere di maxStayTime il waiter dovrà recarsi dal cliente per eseguire il task *collect.*
Attenzione: Può succedere che un cliente indeciso passi tutto il maxstaytime a sfogliare il menù. In quel caso il waiter dovrà direttamente eseguire il task *convoy* saltando il task *collect* (il pagamento).

Stima del maxWaitingTime : Secondo una stima pessimistica e al momento grossolana, il waiter può valutare il maxwaitingtime pari a maxstaytime+*LongestPreparationTime*, dove *LongestPreparationTime* indica il tempo di preparazione dell'ordine più lungo da preparare. Dopo tal tempo, infatti, il tavolo che dei due era stato occupato per primo si sarà molto probabilmente liberato. Al momento trascuriamo il tempo impiegato dal waiter per i task *reach* e *convoy* che comunque, essendo la room piccola, non sono significativi.

Stato dei teatable : Il problema richiede di tenere traccia dello stato dei teatable in modo che il waiter possa rispondere opportunamente alle richieste di ingresso. Ho realizzato questa figura per visualizzare graficamente i possibili stati in cui può trovarsi un teatable e le transizioni tra questi:

Momento in cui occupare un tavolo : Per evitare fraintendimenti con i clienti che fanno richiesta di entrare è opportuno che il tavolo passi allo stato 'busy' sin dal task accept del waiter. Infatti, non appena il waiter accetta una richiesta di ingresso di un cliente, anche se fisicamente il tavolo non è ancora occupato, logicamente lo è già.

Il waiter, o qualcuno per lui, deve conoscere la mappa della tea-room : Il problema lo richiede in quanto il waiter deve essere muoversi autonomamente al suo interno.

Il waiter, o qualcuno per lui, deve tenere traccia della situazione corrente della TeaRoom : Il waiter, infatti, deve essere in grado di rispondere opportunamente alle richieste di ingresso, di servire gli ordini al tavolo giusto ecc..

Current situation of the tearoom : La situazione corrente della stanza interessa anche al Manager, che desidera poterla visualizzare. Ma come raccogliere le informazioni? Soluzione centralizzata dove tutto lo stato della stanza viene mantenuto da un'unica entità, ad esempio da waiter, o decentralizzata dove ogni entità tiene una propria rappresentazione dello stato e quando voglio sapere quella totale devo interrogare ciascuna entità? Fare un'analisi con i Pro e i Contro delle due soluzioni.

Stato logico del waiter : Il problema richiede di tenere traccia dello stato logico in cui si trova il waiter. Al momento possiamo pensare a 4 stati logici principali. Se servirà in futuro potremo modificarli o aggiungerne altri:
-*rest*: il waiter sta riposando e si trova nella home.

-*doing_nothing*: il waiter non ha nulla da fare, ma non si trova alla home.

-*serving client(ClientID)*: il waiter si sta dedicando al cliente clientID.

-*cleaning (table(N))*: il waiter sta pulendo il tavolo N.

Il waiter non deve perdere nessuna richiesta : Per non suscitare malcontento nei clienti, il waiter dovrà essere in grado di recepire tutte le richieste che gli vengono fatte e, se non può soddisfarle subito, deve tenerne conto e soddisfarle in futuro quando riterrà più opportuno. → Discard Message Off.

Ottimizzazione dei task del waiter : Il problema richiede che il waiter sia in grado di ridurre al minimo il tempo di attesa di una richiesta proveniente da un cliente. Questo significa che il waiter dovrà essere in grado di fare **interleaving** dei suoi task, ossia dovrà essere in grado di valutare autonomamente se, sotto determinate condizioni, sia il caso di interrompere l'attività che sta eseguendo per dedicarsi eventualmente ad un'altra. Dovrà poi essere in grado di riprendere l'attività interrotta dal punto in cui era rimasto e di portarla a termine. Si valuti nei successivi Sprint anche l'opportunità di far eseguire al waiter alcuni task in **parallelo**.

Rest : Se il waiter tentasse di tornare alla home non appena non riceve più richieste, potrebbe essere frequente che venga subito e nuovamente interpellato generando un 'avanti e indietro' da e verso la home inutile. Per efficientare energeticamente il suo comportamento e ridurre di conseguenza i costi del suo utilizzo si è pensato di farlo tornare alla home SOLO DOPO un determinato tempo *TimeToRest* (da concordare con il committente) nel quale non ha ricevuto alcuna richiesta. Si ipotizza infatti che, se per un tempo *TimeToRest* non arrivano richieste, la situazione sia abbastanza tranquilla il waiter possa quindi riposarsi.

Barman-waiter

Interazione tra barman-waiter: necessario comunicarsi i clientID . Sotto l'ipotesi che il barman sia in grado di ricevere e preparare più ordinazioni in parallelo è fondamentale che, quando il waiter comunica al barman l'ordine da preparare e, successivamente, quando il barman comunica al waiter che l'ordine è pronto si specifichi sempre un riferimento al clientID a cui appartiene l'ordine, in modo da poterlo servire al tavolo con il cliente giusto.

NOTA: Essendo il n° di clienti in sala <= 2 potranno sovrapporsi al massimo 2 ordinazioni.

Smartbell

Licenza di respingere i clienti con febbre : In un'ottica di ottimizzazione e alleggerimento del lavoro del waiter si può pensare di dare la possibilità allo smartbell di poter respingere i clienti con T°>=37.5 senza assegnare loro un clientID e senza inoltrare una richiesta al waiter.

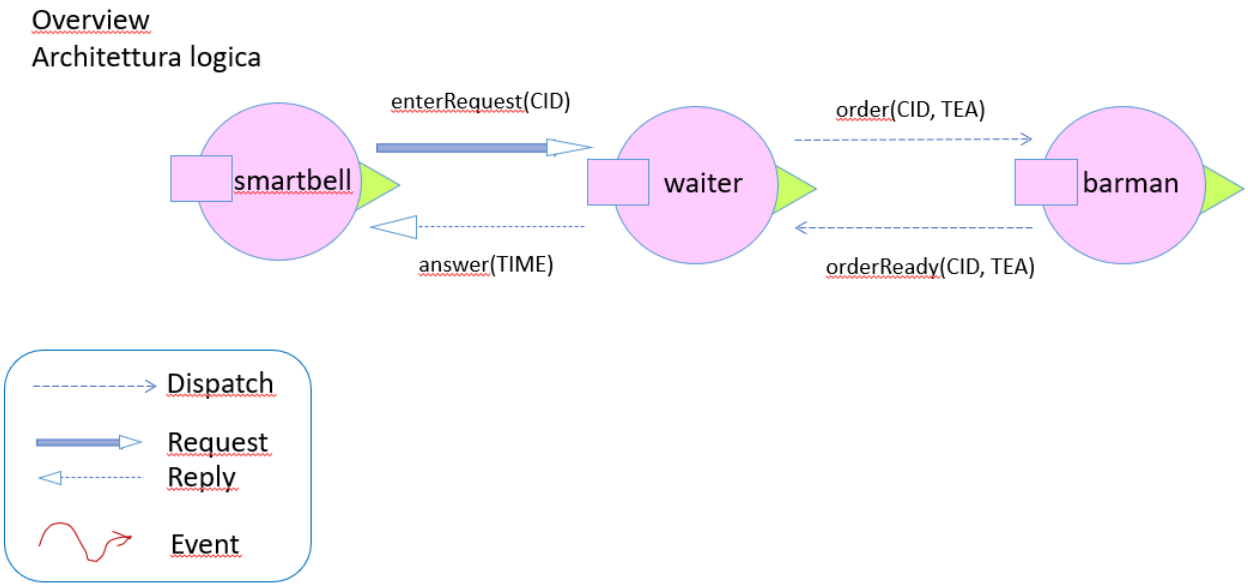
Interazioni Wi-Fi : Dopo averlo chiesto al committente si segnala la possibilità che lo scambio di informazioni tra smartbell<->waiter e waiter<->barman possano avvenire via rete senza la necessità che il waiter si rechi fisicamente presso l'entrancedoor o il service desk. I task *take* e *collect* invece richiedono che il waiter si rechi fisicamente al tavolo del cliente. Anche il task *serve* richiede lo spostamento del waiter tra tavolo e servicedesk.

Requisito opzionale: one client in the hall . La hall andrà modellata come risorsa mutuamente esclusiva.

Cliente dispettoso : Come comportarsi se un cliente suona tante volte oppure suona una volta e poi va via? Questa problematica può essere margine di raffinamento di uno Sprint avanzato.

ARCHITETTURA LOGICA

Si delinea quindi la seguente architettura logica del sistema:



ABSTRACTION GAP

Leggendo i requisiti è abbastanza evidente come il problema che stiamo affrontando descriva un sistema distribuito composto da diverse entità. Ciascuna di queste deve essere in grado di agire autonomamente e deve essere in grado di interagire con le altre entità del sistema. Queste caratteristiche sono molto più vicine alla definizione di Attore che alla definizione di oggetto. Gli attori sono entità dotate di flusso di controllo autonomo e che comunicano tra loro non tramite procedure-call, bensì in modo asincrono, a scambio di messaggi. Si presenta dunque l'opportunità di ridurre l'abstraction gap scegliendo di modellare le entità del nostro sistema come attori.

Per ridurre ulteriormente l'abstraction gap e velocizzare il processo di sviluppo si segnala l'opportunità di utilizzare i QActor sfruttando il Domain Specific Language (DSL) QaKActor. Si tratta di un DSL interno che è stato sviluppato nel corso delle lezioni come infrastruttura sopra al linguaggio Kotlin.

Grazie a questo DSL è possibile scrivere dei modelli di un sistema distribuito composto da attori focalizzandoci solo sulla logica applicativa del problema, sul comportamento dei suoi componenti e sulla loro interazione, astraendo dai dettagli implementativi.

Abbiamo così la possibilità di esprimere in modo conciso e formale (eseguibile da un calcolatore) dei modelli del sistema sin dalle prime fasi del processo di sviluppo, cosa che è fondamentale per instaurare un rapporto più produttivo con il committente.

Sarà compito del progettista poi fare zooming sui componenti del modello, concentrandosi sui dettagli implementativi.

MODELLO ESEGUIBILE DEL SISTEMA

Si veda il file a questo link: analysisModel.qak.

NOTE:

1) Ai fini della simulazione e del testing ho introdotto un attore chiamato *client_simulator* che simulasse stato per stato tutto il comportamento di un cliente e che interagisse tramite messaggi sia con la smartbell sia con il waiter. Sono consapevole che non sia richiesto dai requisiti modellare l'entità cliente, tuttavia questo modello grezzo che ho realizzato potrà servire in futuro come punto di partenza se ci verrà richiesto di realizzare ad esempio un'applicazione (es: su smartphone) che permetta al cliente di interagire con il waiter e con la smartbell.
   È emersa tuttavia una problematica:

   La necessità di *sincronizzare il comportamento del cliente con quello del waiter* poichè il cliente deve fare le richieste giuste al momento giusto (es: dire che è pronto per ordinare quando è già seduto al tavolo e non quando ci si sta ancora recando). In questo sprint preliminare mi sono proposto di risolvere la problematica inserendo dei messaggi request-reply.

2) Ho introdotto anche un'entità chiamata *situation_observer* per iniziare a formalizzare la problematica del Manager che vuole sapere la situazione corrente della stanza. Verrà approfondita in uno sprint successivo.