REQUIREMENT ANALYSIS

Ambiente	
ServiceArea	Area della tearoom in cui il waiter può restare in attesa di task e in cui si trova il serviceDesk da cui prendere il tè da servire ai clienti.
Hall	Locale in cui i clienti possono notificare il loro interesse ad entrare nella tearoom ed eventualmente aspettare che ci sia un tavolo libero.
Smartbell	Dispositivo posizionato nella hall che permette ai clienti di notificare l'interesse ad entrare nella tearoom. Il successo di tale operazione è subordinato all'assenza di una temperatura corporea superiore ai 37.5°, in tal caso la smartBell è in grado di rifiutare autonomamente il cliente.

Cliente	
notify	Azionamento della smartBell. Innesca il controllo della temperatura corporea, in caso di temperatura inferiore ai 37.5° al cliente sarà assegnato un identificativo unico ed eventualmente messo in attesa di un tavolo libero. In caso di temperatura superiore ai 37.5°, sarà chiesto al cliente di uscire.
clientIdentifier	Identificatore unico che viene attribuito ad un cliente.

Manager	
currentState	Per currentState si intende: • Cosa sta facendo il waiter • Il numero di tavoli occupati • Il numero di tavoli liberi ma non ancora puliti • Il numero di tavoli puliti e pronti per essere assegnati • Barman, tea pronto.

Waiter	
accept	Si intende l'azione di andare alla entranceDoor, accompagnare il cliente ad un tavolo libero ed essere pronto a tornare al tavolo quando i clienti sono pronti per l'ordinazione.

Entità del sistema

Il sistema è composto da tre entità interagenti (**waiter, barman, smartbell**) e dalle entità esterne **cliente** e **manager**, che rappresentano, rispettivamente, il ciclo di vita di un cliente all'interno del sistema e il manager in grado di osservare lo stato della stanza attraverso un web server collegato all'applicazione.

Messaggi

Dai requisiti, emerge che le entità in gioco hanno necessità di interagire fra loro tramite l'ausilio di **messaggi**.

Si pensi per esempio al caso della smartbell che all'ingresso di un client invia una richiesta al waiter quando un client entra nel sistema e la sua temperatura è giudicata consona. Allo stesso modo, un client notifica, per cui invia un messaggio, la sua presenza alla smartbell e comunica con il waiter durante l'ordine. I termini *notify, inform* etc. sottintendono uno scambio di uno o più messaggi.

La nozione di messaggio rimane così però, troppo generica, nel seguito del documento faremo riferimento a 3 tipi di messaggi:

- Dispatch: messaggio inviato dal sender a un preciso receiver, senza aspettarsi una risposta
- Request/Reply: sender invia una request a un preciso receiver e si aspetta un reply da quest'ultimo
- Event: messaggio inviato dal sender senza avere in mente un receiver preciso, viene emesso e poi il receiver si disinteressa di chi possa riceverlo, sono i receiver che devono manifestare il loro interesse a ricevere tali eventi.

Attori

Inoltre, per ogni entità è possibile identificare uno **stato** in cui si trova, cioè una sua configurazione data dalla storia di interazione passata con le altre entità, che ne determina il comportamento attuale.

Ad esempio, guardando quanto verrà discusso ed approfondito ulteriormente nel modello ed in seguito in fase di analisi del problema, si può pensare che il barman sia libero di ricevere ordini o che stia preparando un tè, per cui l'interazione con esso cambia in base al suo stato. Vale lo stesso per le altre entità.

Alla luce di quanto detto, è possibile modellare le entità in gioco come **attori** e formalizzarne il comportamento tramite metamodello qak (per cui si rimanda a https://github.com/anatali/iss2020LabBo/tree/master/it.unibo.qakactor/userDocs).

Gli attori sono entità dotate di stato (macchine a stati finiti), capaci di interagire tramite messaggi (visti prima) dalla precisa semantica (per cui si rimanda sempre al link precedente).

Per poter definire le varie entità come attori si fa riferimento a un prodotto della nostra software house che già fornisce un modo per poter rappresentare entità come macchine a stati che interagiscono tramite lo scambio di messaggi.

Per formalizzare i requisiti si è scelto come modello il **metamodello qak** perché consente di avere una visione completa e sintetica degli aspetti fondamentali (logici) del sistema e permette una prototipazione veloce.

Infatti, tramite opportune librerie fornite dalla nostra software house (si rimanda a https://github.com/anatali/iss2020LabBo/tree/master/unibolibs) è possibile ottenere un prototipo funzionante direttamente a partire dal modello.

Ciò risulta molto vantaggioso perché riduce di molto l'abstraction gap fra l'applicazione e la sua implementazione, in questa maniera è sufficiente scrivere il modello concentrandosi soltanto su di esso, senza preoccuparsi della loro implementazione a basso livello.

Il modello viene convertito dalla software house in codice kotlin immediatamente eseguibile.

Modello

Gli attori qak individuati sono: **waiter, smartbell, barman** (e **client** come entità esterna per simulare il comportamento di un client all'interno del sistema).

Il modello di dominio qak è consultabile presso:

https://github.com/virtualms/IssProject/tree/master/modelRequirements

Modello. Semantica di interazione fra gli attori

Le seguenti tabelle illustrano la comunicazione fra waiter, smartbell, barman e client, rimarcando quanto presente nel modello. Per ogni tabella, sono presenti i messaggi che coinvolgono l'attore considerato.

Waiter

Request waitTime: waitTime(ID)
Reply waitTimeAnswer: waitTime(TIME)

Request makeTea : makeTea(TEA, TABLE)
Reply teaReady : teaReady(TEA , TABLE)

Request readyToOrder: readyToOrder(TABLE)
Reply atTable: atTable(TABLE)

Dispatch order: order(TEA)

Request readyToPay: readyToPay(TABLE)
Dispatch moneyCollected: moneyCollected(AMOUNT)

Dispatch dirtyTable: dirtyTable(TABLE)

Messaggi	Tipo/semantica	Payload	Da	A	Spiegazione
waitTime(ID)	Request	-ID: id del client	Smartbell	Waiter	Si richiede al waiter il tempo di attesa stimato

waitTimeAnswer (TIME)	Reply	-TIME: tempo di attesa stimato	Waiter	Smartbell	Risposta contenente il tempo di attesa stimato
makeTea(TEA, TABLE)	Request	-TEA: tipo di tea -TABLE: tavolo a cui consegnare l'ordine	Waiter	Barman	Il waiter invia una richiesta che contiene l'ordinazione del cliente.
teaReady(TEA, TABLE)	Reply	-TEA: tea preparato -TABLE: tavolo a cui consegnarlo	Barman	Waiter	Il barman, preparato un ordine, invia la reply al waiter per dirgli che deve ritirarlo
readyToOrder(TABLE)	Request	-TABLE: tavolo da cui ordinare-TABLE: tavolo da cui ordinare	Client	Waiter	Il client è pronto ad ordinare
atTable(TABLE)	Reply	-TABLE: tavolo da cui prendere l'ordine-TABLE: tavolo da cui prendere l'ordine	Waiter	Client	Il waiter notifica l'arrivo al tavolo e prende l'ordine
order(TEA)	Dispatch	-TEA: tea che il client vuole ordinare	Client	Waiter	Il client ordina il tea
readyToPay(TABLE)	Request	-TABLE: tavolo pronto a pagare	Client	Waiter	Il client è pronto a pagare
moneyCollected(AMOUNT)	Dispatch	-AMOUNT: quantità di soldi	Client	Waiter	Il client dà il denaro al waiter
dirtyTable(TABLE)	Dispatch	-TABLE: tavolo sporco	Waiter	Waiter	Il waiter si notifica della presenza di tavoli sporchi

Smartbell

Request enter: enter(TEMP) *Reply tempNotOk : tempNotOk(OK)* *Reply tempOk : tempOk(TIME, ID)* *Request waitTime: waitTime(ID)* *Reply waitTimeAnswer: waitTimeAnswer (TIME)*					
Messaggi	**Tipo/semantica**	**Payload**	**Da**	**A**	**Spiegazione**
enter(TEMP)	Request	-TEMP: temperatura del cliente	Client	Smartbell	Richiesta di ingresso nel sistema di un cliente
tempNotOk(OK)	Reply	-OK: info sulla temperatura	Smartbell	Client	La temperatura è > della soglia
tempOk(TIME, ID)	Reply	-TIME: tempo di attesa stimato -ID: id del cliente	Smartbell	Client	La temperatura è <= alla soglia
waitTime(ID)	Request	-ID: id del client	Smartbell	Waiter	Si richiede al waiter il tempo di attesa stimato
waitTimeAnswer (TIME)	Reply	-TIME: tempo di attesa stimato	Waiter	Smartbell	Risposta contenente il tempo di attesa stimato

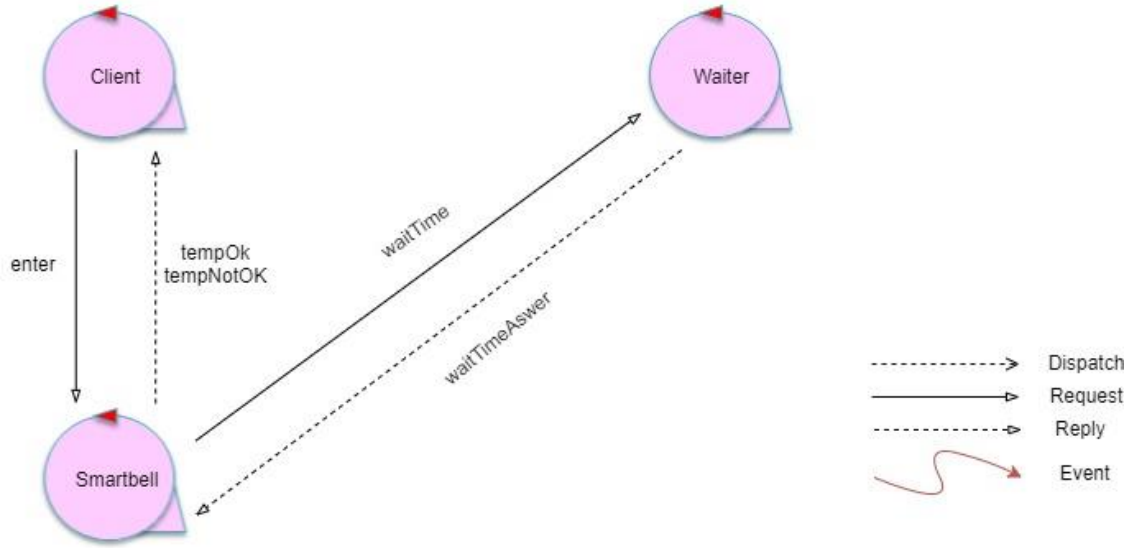
Barman

Request makeTea : makeTea(TEA, TABLE) *Reply teaReady : teaReady(TEA , TABLE)*					
Messaggi	**Tipo/semantica**	**Payload**	**Da**	**A**	**Spiegazione**
makeTea(TEA, TABLE)	Request	-TEA: tipo di tea -TABLE: tavolo a cui consegnare l'ordine	Waiter	Barman	Il waiter invia una richiesta al barman che contiene l'ordinazione del cliente.

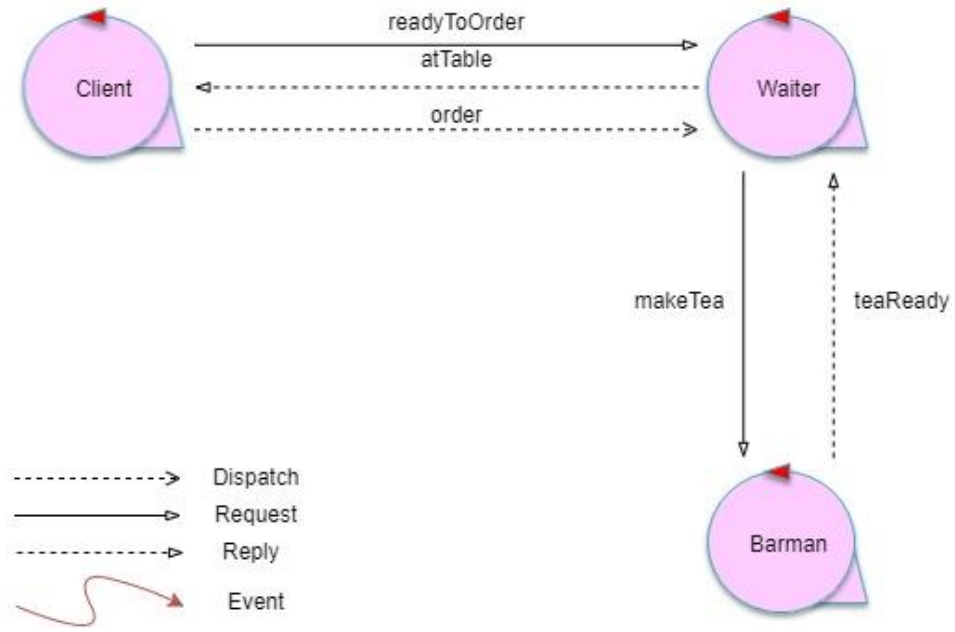
teaReady(ARG , TABLE)	Reply	-TEA: tea preparato -TABLE: tavolo a cui consegnarlo	Barman	Waiter	Il barman, preparato un ordine, invia la reply al waiter per dirgli di venirlo a prendere

Grafici

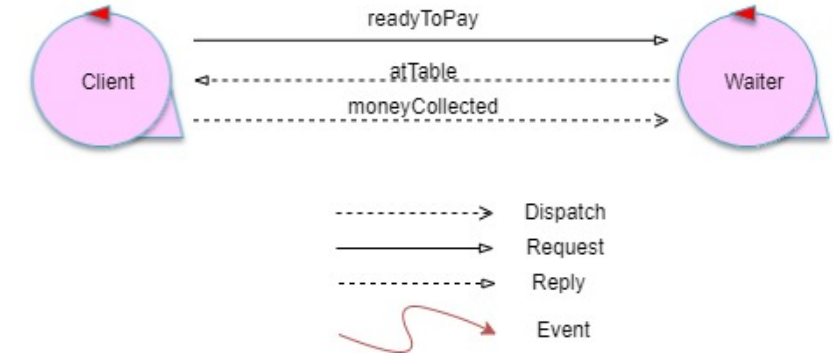
Ingresso di un client ed ordine



Ordine



Pagamento



Current state del sistema

Come già introdotto precedentemente, il current state del sistema consente al manager di consultare informazioni relative allo stato della sala e del waiter.

Più nello specifico deve descrivere, alla luce dei requisiti, le seguenti situazioni:

- **Stato dei tavoli**
- **Stato del waiter**

- **Stato del barman**

Lo stato dei tavoli indica in che condizione sono i vari tavoli presenti nella sala (pulito, sporco, occupato).

Lo stato del waiter indica cosa sta facendo in quel momento il waiter all'interno della sala.

Lo stato del barman indica cosa stia facendo il barman, in linea di massima starà o aspettando una comanda oppure preparando un'ordinazione.

Test plan

Il corretto funzionamento del sistema può essere verificato considerando **il corretto susseguirsi degli stati** per l'entità o le entità scelte ad un determinato input per il sistema, quindi una corretta risposta del sistema.

Consideriamo le seguenti situazioni:

- **Ingresso di un client nel sistema:** il cliente invia un messaggio alla smartbell e la smartbell misura la temperatura del client.
    - **Temperatura > soglia:** la smartbell verifica che il parametro è fuori soglia ed invita il cliente ad abbandonare la tearoom(sistema). Gli altri attori non vengono informati (non viene mandato alcun messaggio) del transito del client. Ci si aspetta quindi che il waiter non agisca ed il client non sia presente nel sistema.
    - **Temperatura <= soglia:** la smartbell verifica che il parametro è nel range di valori validi ed accoglie la richiesta del client. Invia un messaggio al waiter che risponde con un tempo di attesa. Il client è quindi nel sistema e gli vengono forniti id e tempo di attesa massimo.
- **Pulizia di un tavolo:** i tavoli possono essere puliti, occupati o sporchi. Un client, per questioni igieniche, può essere fatto accomodare solo ad un tavolo pulito ed un tavolo può essere pulito solo quando il client lo ha abbandonato (non quando è al tavolo e quindi il tavolo è occupato) ed è stato pulito dal waiter. Si verifichi per cui che un client venga fatto sempre accomodare ad un tavolo libero e pulito.