Il documento di Progettazione

1. Introduzione. Scopo del documento. Descrizione sintetica delle funzionalità più importanti.

2. Architettura del Sistema. Descrizione sottosistemi e moduli. Metodo di progettazione (oggetti/funzionale). Presentare possibili soluzioni diverse.

3. Modello dei Dati

e del Controllo. Come sono organizzati i dati e il controllo all’interno della nostra architettura.

Modello dei dati → Client-Server

Modello del controllo → Basata su eventi-interrupt driver

4. Modelli UML: diagrammi delle classi e di sequenza.

* Diagrammi classi del sistema (architettura del sistema)
* Diagrammi sequenze per ogni singolo caso d’uso.

5. Progettazione dell’interfaccia Utente

6. Glossario

2. Architettura del Sistema.

Possibili sottosistemi

* Gestione comunicazione
* Gestione locale (tavoli e notifiche)
* Gestione sensori (rilevazione shake, gps)
* Gestione memoria (nome, descrizione locale, modalità applicazione, …)
* (gestione autenticazione)

Modello ad oggetti. Sistema decentralizzato con oggetti che si passano il controllo.

(Cenni su considerazioni relative a qualità ~ accoppiamento, coesione, …)

Mostrare varie alternative :

* unico sottosistema comunicazione/diviso in 2 → abbiamo scelto il secondo per maggiore coesione e accoppiamento

3. Modello dei Dati e del Controllo.

* Modello dei dati → Client-Server

Non si sono dati centralizzati ma sono tutti interni ai vari sottosistemi. Non c’è necessità o utilità nel tenere dati centralizzati.

* Modello del controllo → Basata su eventi-interrupt driver.

Programma organizzato in modo massiccio ad eventi. Quando accade un evento sono chiamati solo i sottosistemi associati.

4. Modelli UML: diagrammi delle classi e di sequenza.

A Diagrammi delle classi.

Possibili classi:

* Locale. Classe che rappresenta l’intero locale ed è in pratica il programma. Al suo interno contiene un oggetto (singleton) di Stanza virtuale, di Tavoli, di Notifiche Shake.

Metodi

- void creaStanza() (possibile metodo privato)

- List<Chiamata> vediListaChiamateCameriere()

- void prendiInCaricoChiamataCameriere(Chiamata)

- List<Tavolo> vediListaTavoliLiberi()

- void modificaTavolo(Tavolo) # o id

- void liberaTavolo(Tavolo)

- void occupaTavolo(Tavolo)

- void chiudiStanza()

- Menu visualizzaMenu()

- DescrizioneLocale visualizzaDescrizione()

(possibile metodo statico)

Possibile idea. Oggetto “singleton” della classe stanza virtuale creato dal gestore. Gestore applica metodo per avviare stanza virtuale(bluetooth,wifi,..). Cliente ,tramite classe che si occupa della comunicazione, notifica il gestore della sua volontà di unirsi alla stanza virtuale: il gestore applica metodo partecipa all’oggetto.

* Gestione comunicazione (Nearby)

Metodi classe ComunicazioneCliente (singleton):

* void accediStanza(ClienteBasic)
* void chiamaCameriere()
* void selezionaTavolo(Tavolo)
* void richiediListaTavoliLiberi( Consumer<Collection<Tavolo>> (, int sec) ) (passa una callback)
* void lasciaStanza()
* boolean isConnected()

(possibile metodo statico)

Metodi classe ComunicazioneGestore (singleton):

* void avviaStanza() (possibile metodo privato)
* void onChiamataCameriere( Consumer<Chiamata> ) (passa una callback) (le callback potrebbero essere passate al costruttore e i metodi privati)
* void onSelezioneTavolo( Consumer<Tavolo> ) (passa una callback)
* void onRichiestaTavoliLiberi( Supplier<List<Tavolo>> ) (passa una callback)
* void chiudiStanza()
* GestoreTavoli (singleton)
  + List<Tavolo> tavoli (final) (eventualmente con un’altra classe per immutable)
  + map<Tavoli,Clienti> occupati (private)

costruttore (passo il numero di tavoli) (Possibilità di mettere numero di posti nei tavoli)

* + List<Tavolo> vediListaTavoliLiberi()
  + void modificaTavolo(Cliente, Tavolo libero)
  + void liberaTavolo(Tavolo)
  + void occupaTavolo(Tavolo, Customer)
  + Tavolo getTavolo(Cliente)
* GestoreChiamate
  + void aggiungiChiamata(Chiamata)
  + void togliChiamata(Chiamata )
  + List<Chiamata> vediListaChiamate()
* Chiamata (immutable) (solo gestore)

costruttore con Cliente e ora (Date)

* Cliente getCliente()
* Date getOra()
* Tavolo (Possibilità sottoclassi cliente e gestore) (immutable)
* Cliente (immutable)
* GestoreMemoria (forse)
* Prodotto (immutable)
  + nome
  + prezzo
  + descrizione
* DescrizioneLocale (immutable)
  + nome
  + body (descrizione)
  + posizione
  + immagine

* Menu
  + List<Prodotto>
* Messaggio (classe usata dentro ComunicazioneGestore e ComunicazioneCliente)

Rappresenta la classe degli oggetti che sono serializzati per poi essere trasmessi tramite nearby (come bytes).

Funge da anello di congiunzione tra oggetti di alto livello che vogliamo trasmettere/ricevere (tavoli,chiamate, …) e i bytes che effettivamente trasmettiamo.

Enum TipoRichiesta

costruttore(TipoRichiesta , Serializable)

(in alternativa metodo statico

buildForTableRequest(Tavolo))

Object getContent()

TipoRichiesta getRequestType()

Classe Sensor

* void onShakeDetected( Runnable runnable)
* void onEntrance( Runnable runnable )

B) Diagrammi sequenza

Chiamiamo applicazione il codice interfaccia grafica.

In alcuni casi d’uso mettiamo anche l’utente (dove necessario)

* Scegli cliente

Applicazione <-> CustomerStorage

* Scegli gestore

Applicazione <-> ManagerStorage

* Inserimento nome

Applicazione <-> CustomerStorage

* Inserimento pin

Applicazione <-> ManagerStorage

* Visualizza menu

Applicazione <-> CustomerStorage

( Classi usate ma non specificate : LocalDescription. Indichiamo tali classi come valori)

* Visualizza descrizione locale

Applicazione <-> CustomerStorage

( Classi usate ma non specificate : LocalDescription. Indichiamo tali classi come valori)

* Impostazioni

Applicazione <-> CustomerStorage

* Accedi alla stanza virtuale

Applicazione <-> CustomerComunication <-> ManagerComunication

* Visualizza mappa

Applicazione <-> CustomerStorage

( Classi usate ma non specificate : LocalDescription. Indichiamo tali classi come valori)

* Notifica di benvenuto

Utente <-> Sensor

(Rimandare al caso d’uso accesso stanza virtuale)

* Immetti numero tavolo

Applicazione <-> CostumerComunication <-> ManagerComunication <-> TableHandler

( Classi usate ma non specificate : Table, ManagerTable, Message )

* Chiama cameriere

Utente <-> Sensor <-> CostumerComunication <-> ManagerComunication <-> WaiterNotificationHandler

( Classi usate ma non specificate : WaiterNotification , Message )

* Esci

Applicazione <-> CostumerComunication <-> ManagerComunication

* Attiva/disattiva notifica benvenuto

Applicazione <-> CustomerStorage

* Attiva/disattiva uso sensori

Applicazione <-> CustomerStorage

* Esci da modalità cliente

Applicazione <-> CustomerStorage

* Modifica nome

Applicazione <-> CustomerStorage

* Visualizza menù

Applicazione <-> ManagerStorage

( Classi usate ma non specificate : LocalDescription. Indichiamo tali classi come valori)

* Visualizza descrizione

Applicazione <-> ManagerStorage

( Classi usate ma non specificate : LocalDescription. Indichiamo tali classi come valori)

* Impostazioni

Applicazione <-> ManagerStorage

* Crea stanza virtuale

Applicazione <-> Local <-> ManagerComunication

* Lista tavoli occupati

Applicazione <-> Local <-> TableHandler

( Classi usate ma non specificate :Table,ManagerTable. Indichiamo tali classi come valori)

* Lista notifiche cameriere

Applicazione <-> Local <-> WaiterNotificationHandler

( Classi usate ma non specificate : WaiterNotification. Indichiamo tali classi come valori)

* Chiudi stanza

ApplicazioneGestore <-> ManagerComunication <-> CustomerComunication <-> ApplicazioneCliente

( Classi usate ma non specificate : Message. Indichiamo tali classi come valori)

* Aggiungi tavolo

Applicazione <-> Local <-> TableHandler

( Classi usate ma non specificate :Table,ManagerTable. Indichiamo tali classi come valori)

* Rimuovi tavolo

Applicazione <-> Local <-> TableHandler

( Classi usate ma non specificate :Table,ManagerTable. Indichiamo tali classi come valori)

* Modifica tavolo

Applicazione <-> Local <-> TableHandler

( Classi usate ma non specificate :Table,ManagerTable. Indichiamo tali classi come valori)

* Prendi in carico

Applicazione <-> Local <-> WaiterNotificationHandler

( Classi usate ma non specificate : WaiterNotification. Indichiamo tali classi come valori

* Esci da modalità gestore

Applicazione <-> ManagerStorage

2-3-4(UML)

4(Diagramma sequenza)-Interfaccia grafica