Appunti di Mobile

Lezione 1 – Il mobile computing

Il MC è lo studio di quelle **problematiche** caratterizzate da mobilità dei dispositivi e degli utenti, eterogeneità delle capacità dei dispositivi, comunicazione wireless e non costante, dispositivi con sensoristica avanzata.

Le **discipline** trattate sono: data management (trattamento di dati particolari), sistemi distribuiti (le app sono spesso parte di un sistema distribuito a.e. cloud e servizi web), reti (nuovi protocolli come NFC o bluetooth), human-computer interaction (UX ruolo fondamentale), sistemi operativi (hardware limitato, risparmio energetico, privacy e sicurezza).

Caratteristiche dei dispositivi mobili: internet-centered devices (3G, 4G, WiFi) e rilevamento della posizione (GPS, giroscopio, ...), tante applicazioni per multimedia, interfacce touch-based.

Nello specifico:

- **Input/Output**
  - display (resolution, ...)
  - interface (3D touch, pen, voice, ...)
  - writing recognition
  - exspansion slot (micro sd, ...)
- **Comunicazione**
  - Local (WLAN, USB, NFC, ...)
  - Global (3G, LTE, ...)
- **Sensori e posizionamento**
  - GPS, ...
  - Accelerometro, compasso, ...
  - Sensore di prossimità, lettore di impronte digitali, ...
- **Risorse di calcolo**
  - Memoria (RAM e ROM), CPU, GPU, ...
- **Autonomia**
  - Tipo di batteria
  - Risparmio energetico
- **Supporto alle applicazioni**
  - Versione di sistema operativo
  - Browser, Java, ...
- **Supporto multimediale**
  - Audio, video, ...
- **Caratteristiche fisiche**
  - Dimensioni, peso, ...

I sensori

I **trasduttori** sono sensori che trasformano una forma di energia in un'altra. Sono di input (convertono una quantità in un segnale elettrico, chiamati sensori) e di output (convertono un segnale elettrico in una quantità, chiamati attuatori).

I sensori usati nei dispositivi mobili sono: microfono, camera, accelerometro, giroscopio, magnometro, barometro, GPS, sensore di profondità, sensore di luce, lettore di impronte digitali.

- I **sensori biometrici** sono elettro-cardio-gramma, battito cardiaco, saturazione ossigeno, livello di glucosio nel sangue, ...
- I **sensori indossabili** come gli wearable.
- I **sensori ambientali** raccolgono info come luce, suono, temperatura, ...
- I **sensori virtuali** cercano di fornire le info di un sensore fisico senza che sia effettivamente presente, tramite sensori remoti che spediscono info ad un client. È tutta un'astrazione che permette di calcolare cose, come i passi.

Lezione 2 – Tecniche di sviluppo

Caratteristiche di un OS per dispositivi mobili: vincoli real-time (a.e. quando si riceve una telefonata), gestione dell'energia (basso consumo), minore risorse computazionali (sopratutto per indossabili), connettività (non è costante), eccellente UX.
Altre caratteristiche:

- **No tecniche di paging**. Problema: se la memoria principale termina bisogna chiudere qualche app, questa procedura avviene all'insaputa dell'utente. Soluzione: è definito un ciclo di vita delle app dove ad ogni evento viene richiamato del codice definito dal programmatore.
- **GUI gestita con single thread**. Problema: se avviene una comunicazione di rete ed è bloccante, il main thread risulta occupato e quindi la GUI è bloccata. Soluzione: tutte le operazioni di IO vengono svolte da un thread secondario.
- **Push notifications**: sistema che permette ad un server di inviare info ad un client
- **Integrazione con servizi internet**: vengono integrate delle librerie per riuscire a comunicare con un web service remoto.
- **Integrazione store online**, per scaricare le app.

Differenze Android-iOS: diverso approccio al multitasking (in android si può eseguire codice in background, in iOS si può solo se legato ad alcuni eventi); astrazione dell'hardware (programmatore richiede posizione, l'OS gliela dà).

Modalità di sviluppo di una app: nativa (Java e Swift), cross-platform (C, web app, hybrid app come Cordova, conversione del codice come Xamarin o Unity).

Lezione 3 – Analisi e progettazione

Il **contesto d'uso** delle app per dispositivi mobili: interazione può avvenire mentre l'utente fa altre cose; l'utente potrebbe essere in mobilità; l'utente potrebbe non guardare lo schermo (assistente vocale); l'utente usa le app per risolvere un problema contingente (chiamare un amico, trovare una farmacia aperta, ...); i dispositivi devono essere sempre pronti all'uso.

Le sessioni di uso sono mediamente molto brevi quindi bisogna minimizzare i tempi di apprendimento, di set-up del servizio e di utilizzo.

Rispetto ai dispositivi tradizionali (come un pc) è ok avere poche funzionalità a patto che siano immediate da usare e semplici da imparare (photoshop desktop vs mobile).
Possibili problemi di una app mobile: troppe funzionalità, troppa interazione, troppe istruzioni, lessico troppo complicato.

Principi di progettazione delle interfacce

Gli strumenti di input sono diversi: i dispositivi tradizionali hanno la tastiera e il mouse, i dispositivi mobili hanno il touch screen, che ha un'interazione molto più sofisticata rispetto al mouse, e i pulsanti fisici.

La dimensione dello schermo è molto diversa: nei dispositivi mobili abbiamo GUI per schermi piccoli e, oltre a cambiare la schermata, viene adattato anche il flusso di navigazione.

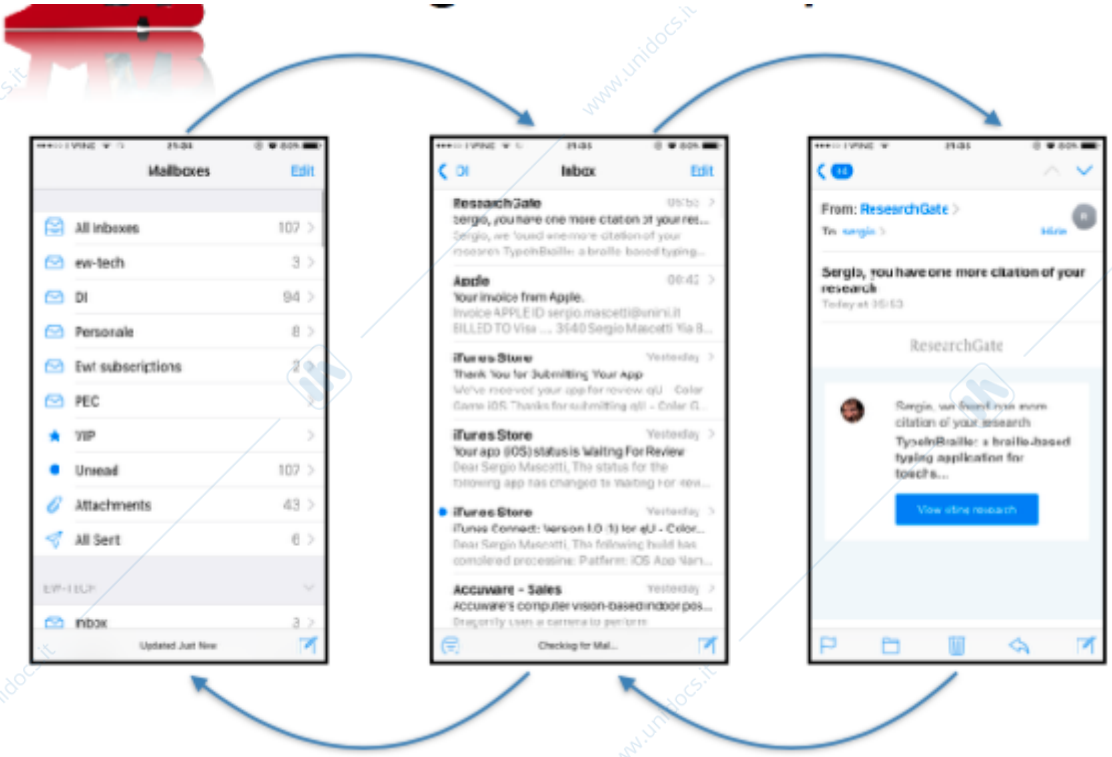
Consigli da seguire:

- **Integrità estetica**: l'aspetto deve riflettere la natura dell'applicazione (produttiva: semplice e lineare; ludica: grafica ricercata e divertente).
- La **consistenza** con gli standard dell'OS o con altre app che l'utente conosce permette di creare app più semplici e rapide da usare.
- **Affordance**: caratteristica di un oggetto o di un ambiente di suggerire la possibilità di compiere una azione.
- **Metafore**: riferimenti al mondo reale per far capire meglio le funzionalità.
- **Personalizzazione** per rendere più attraenti le app (problema: meno intuitive).
- **Minimo sforzo**: inserimento dell'input (come un testo) richiede uno sforzo, quindi chiederlo solo se indispensabile; elementi troppo piccoli sono difficili da leggere e da premere.
- **Pensa agli utenti**: lessico comune e no dettagli tecnici.

Progettazione delle interfacce in pratica

Prima di iniziare serve sapere quali utenti useranno l'app e quali funzionalità si vogliono implementare.

Schema di navigazione
1 schermata 1 task. Lo schema di navigazione definisce come l'utente si muoverà tra le varie schermate. Le funzionalità più importanti devono essere mostrate il prima possibile. La navigazione tra le schermate può avvenire in modo **sequenziale** (avanti e indietro tra le schermate; posso usare una barra di navigazione per sapere dove ci si trova)



o per **task** (da una schermata posso saltare ad un'altra qualsiasi; posso usare una bottom navigation bar o un menu laterale).

Alcune app usano le due modalità in maniera combinata:

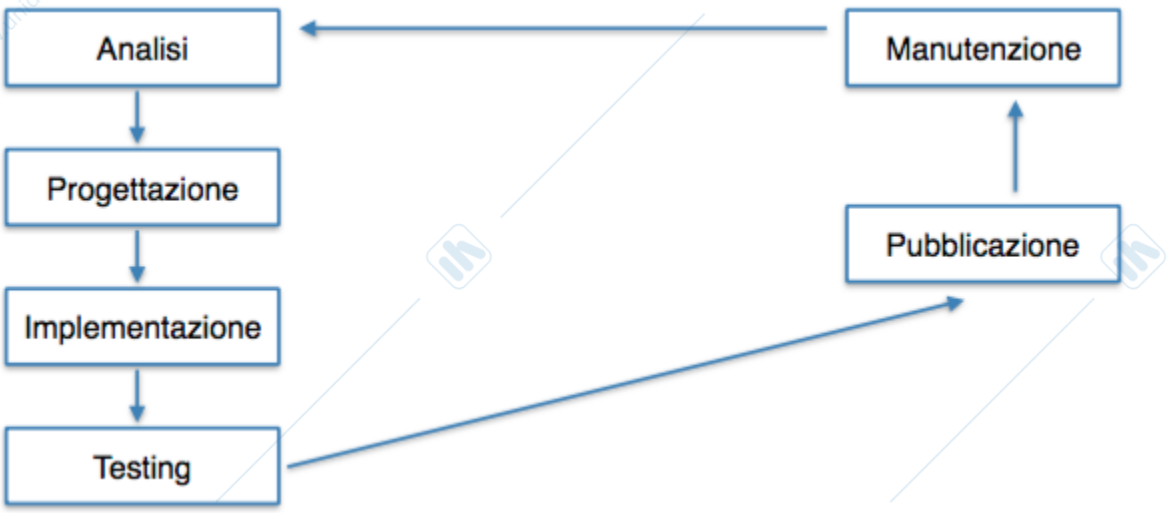
L'ideale è iniziare a progettare per i dispositivi mobili con schermo più piccolo per poi adattare la progettazione a schermi più grandi.

Progressive enhancement: progetto un sito che funzioni su tutti i browser e poi cerco di capire come renderlo migliore per alcuni browser.
Graceful degradation: progetto un sito che funzioni bene su alcuni browser e poi definisco quali funzionalità togliere per farlo funzionare su tutti i browser.

È meglio usare gli oggetti di interfaccia (pulsanti, caselle di testo, ...) forniti dal framework.

Non aspettare che la app sia completa e perfetta: individuare le funzionalità minime, implementarle stando attenti alla UX, pubblicare l'app e guardare il feedback.

Modello di sviluppo



Analisi: stare attenti al primo avvio dell'app da parte dell'utente (è tutto chiaro? servono istruzioni? servono dei dati iniziali?).

Progettazione: provare diverse soluzione prima di implementarne una.

Lezione 4 – Reti e architetture

Le reti wireless

I dispositivi mobili trasmettono dati a livello geografico utilizzando Internet. Ciascun dispositivo è identificato da un indirizzo IP assegnato automaticamente quando si collega a un router o a una rete cellulare.

Tipologie di reti wireless
- **WWAN** (Wireless Wide Area Network): ampia copertura e throughput (cioè la capacità di trasmissione effettiva) molto variabile (3G, 4G)
- **Reti satellitari**: copertura globale, alto costo, basso throughput
- **WLAN** (Wireless Local Area Network): copertura locale, alto throughput (WiFi)
- **WPAN** (Wireless Personal Area Network): copertura pochi metri, basso costo, medio throughput (Bluetooth)

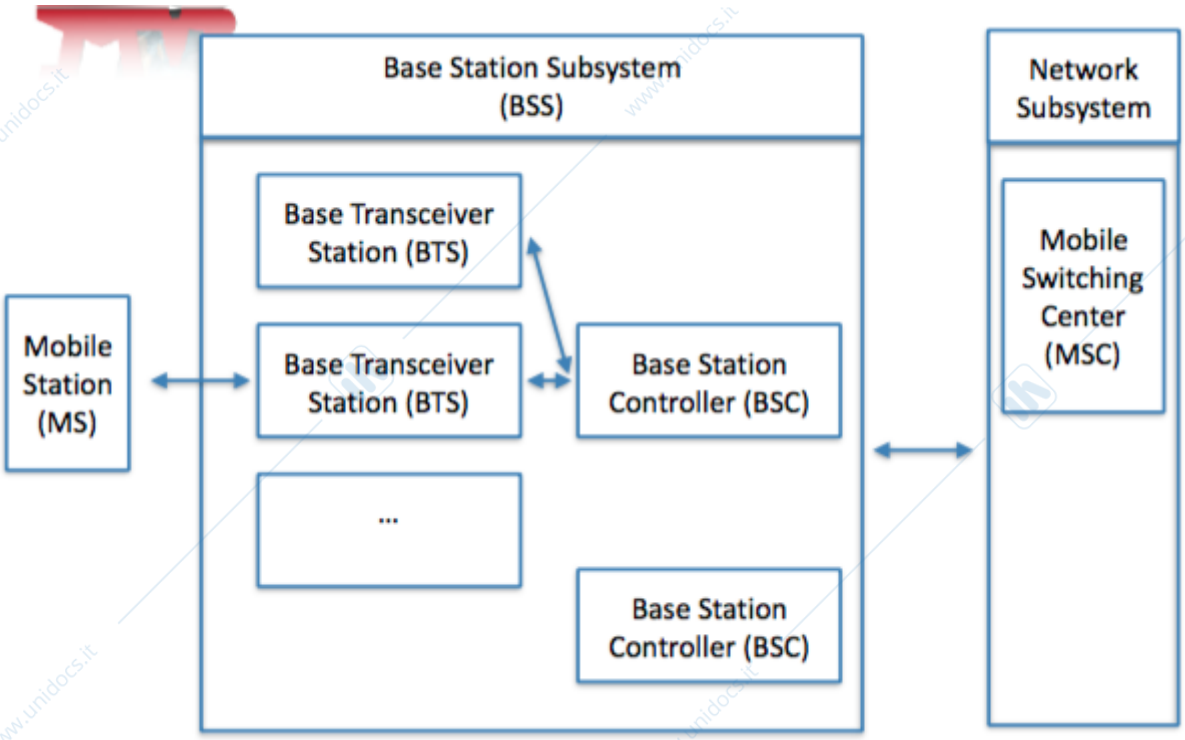
Le reti cellulare

Evoluzione: 1G ‘80 (telefonate in analogico), 2G ‘90 (telefonate in digitale, sms, pochi dati per la navigazione: GSM e EDGE), 3G 2000 (dati in commutazione di pacchetto: HSPA), 4G 2010 (LTE), 5G (in sperimentazione).

Elementi base di una rete GSM:

- **MS** (Mobile Station): terminale (identificato univocamente da un codice, IMEI) dotato di sim card. La sim card contiene il codice personale dell’utente IMSI, una chiave segreta di autenticazione e un PIN/PUK per l’accesso.
- **BSS** (Base Station Subsystem): gestisce connessione radio con MS.
  È composto da:
  - **BTS** (Base Transceiver Station): definisce una cella ospitando i radiotrasmettitori che parlano con le MS e gestisce il protocollo radio con le MS.
  - **BSC** (Base Station Controller): gestisce una o più BTS e l’inizializzazione del radiocanale, dell’handover e del frequency hoopping.

  Una **cella** è un’area dove la potenza del segnale di quella BTS è più forte rispetto alla potenza delle BTS circostanti. I contorni sono i punti dove la potenza di segnale delle BTS si equivale.
- **Network Subsystem**: realizza connessione tra MS e MS o tra MS e telefoni fissi; gestisce l’autenticazione delle MS.
  Il **MSC** (Mobile Service Switching Center) gestisce l’interfaccia tra rete mobile e rete fissa.
  Abbiamo diversi database: **HLR** (Home Location Register) contiene le info sugli abbonati alla rete e la loro posizione attuale; **VLR** (Visitor Location Register) ha l’elenco delle MS che si trovato in quel momento in un dato sottoinsieme di celle; **EIR** (Equipment Identify Register) contiene gli IMIE dei dispositivi che sono autorizzati ad operare nella rete; **AuC** (Authentication Center) contiene una copia delle chiavi segrete contenute nelle SIM degli abbonati.



Interfaccia radio

Frequenza: 900 e 1800 Mhz in Europa, 1900 Mhz in USA.
I 900 Mhz sono suddivisi in 248 portanti (124 in uplink, 124 in downlink). Ogni portante ha 8 canali **TDMA** (Time Division Multiple Access). Il massimo numero di comunicazioni è 124*8=992.
Il canale è stabilito da MS+BSS con un apposito protocollo.

Si possono riusare le frequenze grazie alla divisione in celle. Ogni cella può allocare 200 canali. Le singole celle sono raggruppate in cluster di 4, 7, 12 o 21 celle adiacenti. Ogni cluster usa tutte le frequenze.

- Nell'immagine un esempio di cluster a 7 celle
  - le lettere indicano alcuni gruppi di frequenze
- Osserva:
  - celle con la stessa lettera non sono mai contigue
  - così si evitano interferenze

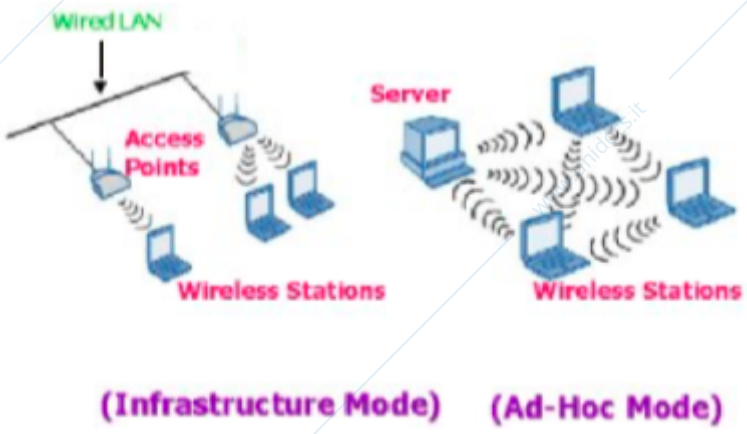
L'**handover** consiste in una riallocazione di un canale su iniziativa di un MS. Può succedere per due motivi: troppa interferenze sul canale usato dal MS oppure il MS si sta spostando.
Il MS controlla sempre la potenza del segnale della cella e delle celle vicine (ha una lista delle migliori 6 ordinate per potenza di segnale; questa lista viene passata al BSC una volta al secondo). L'handover viene gestito localmente se entrambe le celle sono gestite dallo stesso BSC.

Connettività locale: WPAN e WLAN

Tipo di rete	Copertura tipica	Funzione	Costo	Throughput tipico	Standards
WPAN	<10m	Rimpiazza i cavi, reti "personali"	Molto basso	0.1 – 4Mbps	IrDA, Bluetooth, ZigBee, …
WLAN	<100m	Estensione o rimpiazzo di rete ethernet	Medio-basso	20-100Mbps	802.11b,g, n, ac

- **WLAN**

Ci sono due modalità principali di funzionamento: ad infrastruttura e ad-hoc.



I protocolli usati vanno dal 802.11b al 802.11ac. Le frequenze sono o 2.4 o 5 Ghz; la copertura è fino a 250 m outdoor; il throughput (teorico) è fino a 1 Gbps.

- **Bluetooth**

Non necessita di line-of-sight (linea di vista: il percorso ottico in linea retta fra un dispositivo trasmettitore ed uno ricevitore), è a basso costo e basso consumo.
La versione 4 (BT LE, Low Energy) è a basso consumo ed è pensata per l'IoT. Prevede il *pairing* per l'accoppiamento e il *broadcast* per la pubblicazione di dati che possono essere ascoltati da altri dispositivi.

Le architetture per sistemi mobili

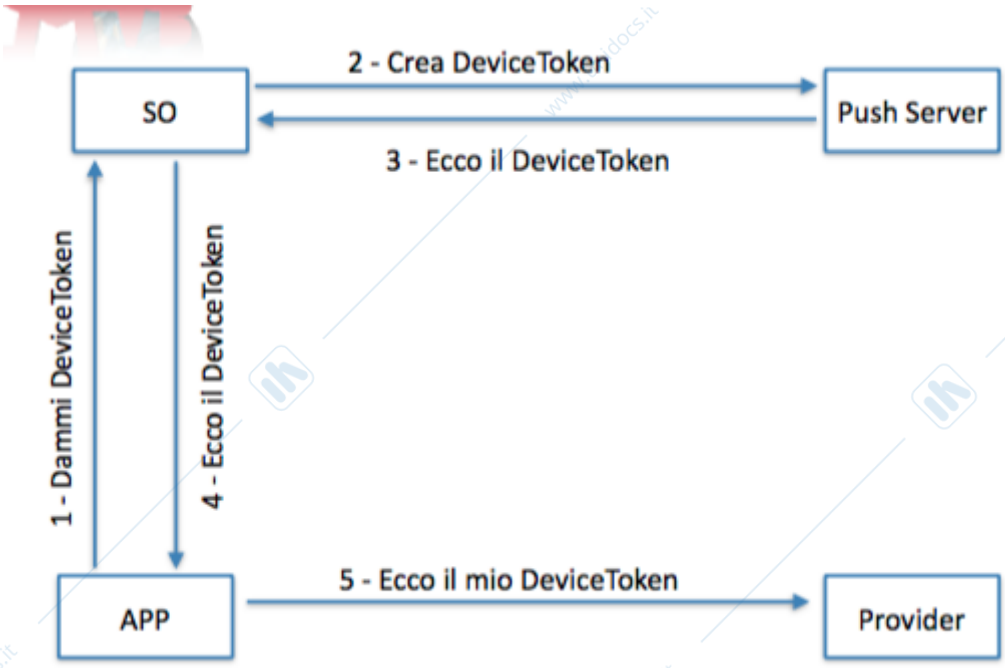
- Problemi della **computazione CPU bound**: i dispositivi mobili hanno minori capacità computazionali e minore quantità di memoria principale. Inoltre l'uso intensivo di CPU impatta sul consumo energetico.
  Soluzione: le operazioni CPU bound vengono eseguite su una macchina remota (quando c'è una rete decente, stando attenti alla privacy e alla sicurezza).

- Problemi dell'**esecuzione in background**: influenza la durata della batteria e i tempi di risposta delle funzionalità real time.
  Soluzioni: iOS limita le azioni in background; esecuzione in backgound event-driven, cioè l'applicazione è sospesa fino a quando un evento non la sveglia.

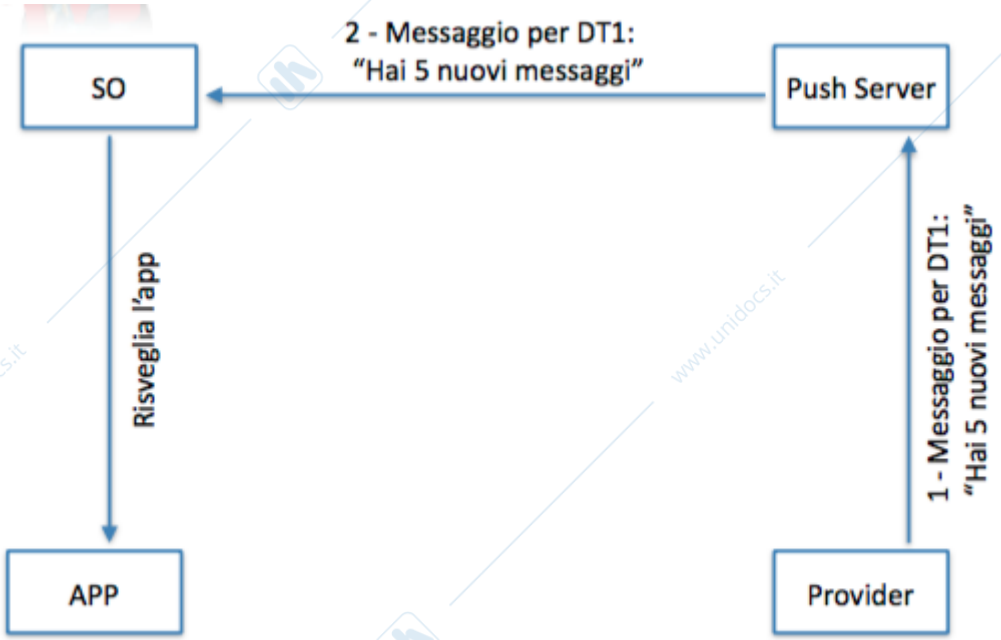
- Problemi della **memorizzazione persistente**: dispositivi hanno memoria limitata; un utente condivide le proprie info tra dispositivi diversi; i dispositivi mobili si possono rompere o possono essere rubati.
  Soluzioni: sincronizzazione e backup delle info; memorizzazione delle info su macchine in remoto (cloud) usando la memoria locale come cache.

- Problemi delle **comunicazione di rete**: i dispositivi utilizzano tanti tipi di connessione che variano per tipologia, velocità di trasmissione e politiche di sicurezza; è frequente che la connessione si interrompa e che quindi cambi l'IP o le porte si chiudano.
  Soluzioni: si fa in modo che sia sempre il server a contattare il dispositivo e non viceversa, usando porte standard (a.e. la 80) e protocolli basati su HTTP; il server usa le notifiche push per notificare un evento.

- Problemi della **comunicazione asincrona con il server**: solitamente quando un server vuole notificare un evento ad una app questa deve rimanere in background e o accettare connessioni o contattare periodicamente il server (pooling) o mantenere una connessione. Queste azioni sono impossibili per un dispositivo mobile perché l'IP può cambiare, il pooling fa un sacco di operazioni inutili e vengono richieste un sacco di risorse.
  Soluzione: le **notifiche push**. Viene attivato un solo servizio per le comunicazioni (implementato dal sistema operativo lato client) e un push server (ApplePushNotificationService per Apple, FirebaseCloudMessaging per Google) nell'infrastruttura dedicata al servizio. Come funziona: l'app server chiede al push server l'invio di una notifica, il push server invia la notifica al sistema operativo del dispositivo, il sistema operativo inoltra la notifica all'applicazione.
  Fase di setup: ogni coppia <applicazione, device> è identificata da un DeviceToken: l'app richiede il DeviceToken all'OS, l'OS riceve il token dal push server e lo invia all'app, l'app lo comunica al provider (il server che vuole inviare il messaggio all'app).



Fase di invio notifica: quando il provider vuole inviare una notifica all'app contatta il push server (gli invia il DeviceToken e il testo da mostrare); il push server verifica la validità del token e inoltra il messaggio all'OS del dispositivo; l'OS mostra il messaggio all'utente.



Tecnologie per l'implementazione delle architetture

Web Service: un server fornisce informazioni (HTML) finalizzate ad essere ricevute da un'applicazione. Servono dei parametri di input e output (codificati in Plain text, XML o JSON).

- **Plain text** (testo semplice), non va bene per strutture dati complesse.

- **XML** (Extensible Markup Language): linguaggio di markup che permette di definire formati di dati che sono leggibili dagli umani e dalle macchine. Un documento XML è valido se rispetta alcune regole (a.e. tutti i tag sono aperti e poi chiusi). Limiti: bisogna scrivere un sacco di roba (verbosità); non è facilmente leggibile dagli umani quindi aumenta la probabilità di errori; gli schemi sono difficili da definire.
- **JSON** (JavaScript Object Notation): come XML ma cerca di migliorare la verbosità e la leggibilità. Non c'è uno schema vero e proprio.

De/Serializzazione: sui client e sui server le info sono quasi sempre memorizzate come oggetti, ma sulla rete passano come XML o JSON. Quindi bisogna prendere un oggetto e convertirlo in XML o JSON (serializzazione o marshalling) e prendere un XML o JSON e convertirlo in un oggetto (deserializzazione o unmarshalling).

GSON è una soluzione di Google per la de/serializzazione automatica degli oggetti. Non serve scrivere il codice ma basta creare delle classi apposite.

XML e JSON riescono a mandare i dati testuali, ma come si fa con i **dati binari** (a.e. una immagine)? Uso la codifica in base64. L'idea è questa: definisco una conversione tra 64 caratteri ASCII e un valore numerico (a.e. A è 1, B è 2, ...).
Un simbolo rappresenta 6 bit (2^6=64) quindi per rappresentare 3 byte servono 4 simboli (3byte*8/6bit=4 simboli). Da binario a base64: suddivido l'input in gruppi di 3 byte poi codifico ciascun gruppo con 4 simboli base64.

Bit pattern	0 1 0 0 1 1	0 1 0 1 1 0	0 0 0 1 0 1	1 0 1 1 1 0
Index	19	22	5	46
Base64-encoded	T	W	F	u

Lezione 5 – Calcolo e uso della posizione

Sistema di riferimento geografico: **WGS84** (World Geodetic System 1984) è un sistema di coordinate (longitudine, latitudine, altitudine) usato per cartografia e navigazione. Ci sono due formati: gradi, minuti e decimali (41° 25' 01" N, 120° 58' 57" W) oppure gradi e decimali (45.454444, 9.214976).

Nella posizione non c'è il tempo; viene "passato" con la posizione poiché è fondamentale per calcolare lo spostamento: *<timestamp, position>*.

Il calcolo della posizione non è esatto, è approssimato. In termini matematici è una funzione (di distribuzione di probabilità) che associa ad ogni punto dello spazio la probabilità che il dispositivo si trovi lì. In termini pratici, l'area può essere una regione circolare o semantica (a.e. il comune di milano). Non ci sono però garanzie che il dispositivo si trovi effettivamente in quell'area.

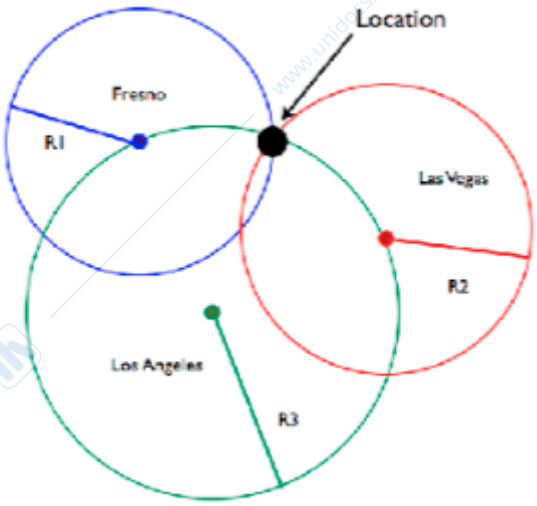
Il **livello di precisione** dipende dal tipo di applicazione: area regionale (200km) per il meteo e le news, area metropolitana (20km) per il traffico, area cittadina (2km) per le flotte, un isolato (100m) per le emergenze e la pubblicità georeferenziata, dintorni (5-50m) per i taxi e la navigazione, prossimità (1-5m) per la navigazione indoor.

Anche l'**orientamento** è importante: come è orientato il dispositivo sui tre assi, in quale direzione è orientato o si muove l'utente. Anche in questo caso c'è approssimazione.

Il calcolo della distanza tra due punti non usa il teorema di Pitagora (perché la terra è un geoide) ma la **formula di Haversine**.
Per quanto riguarda la rete stradale, si rappresenta come un grafo pesato (i nodi sono gli incroci, gli archi le strade e i pesi le distanze spaziali): la distanza è data dalla lunghezza del cammino minimo. I pesi possono essere calcolati in base alla distanza tra i due nodi, al tempo di percorrenza in base al traffico o degli altri utenti.

Geofencing: è la procedura per verificare entrata/uscita di un dispositivo da un'area target.

Trilaterazione: calcolo la posizione del dispositivo in base alla sua distanza da oggetti noti.



Calcolo della posizione outdoor

Tecniche:

1) GNSS (Global Navigation Satellite System)
Uso i satelliti in orbita (dei quali conosco la posizione e l'ora esatta). I satelliti inviano il proprio ID e l'informazione temporale, il dispositivo riceve queste informazioni e riesce a calcolare la distanza da questi satelliti sfruttando il tempo di propagazione del segnale. La tecnica è simile alla trilaterazione. Servono almeno 4 satelliti. Problemi: interferenze (meteo ed edifici) e posizione dei satelliti non precisa.

Sistemi esistenti: **GPS** (Global Position System, degli USA) o **GLONASS** (Russia).
In sviluppo: Galileo (Europa) e BEIDOU (Cina).

GPS
24 satelliti su 6 piani diversi. La precisione è di 3-30 metri.
Il **Differential-GPS** permette di correggere gli errori di interferenza grazie a un ricevitore fisso a terra.
L'**Assisted-GPS** prevede che ogni antenna della rete cellulare mantenga un elenco dei satelliti in vista

(così non bisogna aspettare che vengano trovati ogni volta).

2) Rete cellulare
Si usa un Cell-ID: ogni MS è connesso a una BTS di cui è nota la posizione. L'approssimazione è molto alta perché dipende dalla dimensione della cella.
Due soluzioni: parte del protocollo GSM prevede una tecnica per stimare la distanza MS-BTS con una precisione tra i 100 e i 1200 metri; in base al ritardo della propagazione del segnale si stima la distanza MS-BTS, si possono fare le triangolazioni e la precisione arriva a 50-125 metri.

3) WiFi Network based
Il protocollo 802.11 prevede che gli AccessPoint comunichino il proprio ID a tutti i dispositivi nell'area (anche quelli non connessi). Però la posizione degli AP non è nota, la scopro facendo in modo che siano gli utenti stessi a segnalarla mentre si spostano. Ma come faccio a sapere la posizione degli utenti? Uso le tecniche viste prima: quando un dispositivo sa dove si trova, comunica ad un location service la propria posizione e gli ID degli AP che vede. La precisione è sui 10 metri.

4) Soluzioni ibride
Inventato nel 2003 da Skyhook, ora Apple o Google dispongono del proprio location service.

Come faccio a conoscere l'**orientamento**? Posso usare il magnometro (ma è poco preciso) oppure la direzione di spostamento dell'utente. In quest'ultimo caso mi accorgo della rotazione solo dopo che l'utente si è spostato (però posso usare giroscopio e accelerometro per capire la rotazione).

Calcolo della posizione indoor

Problemi: maggiore precisione richiesta a fronte di una precisione disponibile minore (il segnale GPS non è disponibile, la posizione sulla rete cellulare è inaffidabile, non ci sono info per mettere la posizione con il WiFi). Per questo il posizionamento indoor è ancora poco comune.

Tecniche:

1) WiFi fingerprinting
Si campiona il segnale WiFi in diverse posizioni (un tizio va in giro e quando fa tap sulla mappa viene calcolata la potenza del segnale e inviata ad un server). L'algoritmo di calcolo può essere analitico oppure basato su tecniche di apprendimento.

Limiti: il campionamento è lungo; il segnale del campionato può variare (diverso tipo di AP, cambiamenti architettonici, ...). La precisione teorica è di pochi metri, quella reale anche di decine di metri.

2) BT beacons
I beacons sono trasmettitori di segnali radio installati appositamente per fornire info sulla posizione. Viene usata una tecnica di fingerprinting e la precisione è nell'ordine del metro.

3) RFID/NFC
RFID: protocolli per la comunicazione radio per identificare oggetti. L'hardware è composto da un lettore e da un tag e sono progettati per funzionare a meno di 1 metro. Non devo calcolare la posizione perché quando leggo il tag gli sono molto vicino e conosco già la sua posizione.
NFC: tipo particolare di RFID disponibile su alcuni dispositivi mobili; può funzionare sia da lettore che da tag; funziona a pochi centimetri.

4) Marcatori vivisi
Si distribuiscono nell'ambiente dei marcatori facilmente riconoscibili da un dispositivo (a.e. QR code). Il marcatore codifica la posizione.
È una tecnologia semplice, stabile ed economica; però è richiesta un'azione specifica dell'utente.

Soluzione avanzata: usando tecniche di computer vision posso scoprire la posizione e l'orientamento dell'utente rispetto al marcatore.

5) Computer vision
Idea: quando vedo un oggetto caratteristico di un edificio, so dove mi trovo. Questa tecnica funziona raccogliendo immagini dalla camera del dispositivo.

Problemi: riuscire ad avere una rappresentazione dettagliata dell'ambiente e identificare i punti caratteristici distinguendoli da altri simili. Quindi è una tecnica difficile da realizzare ma mi posso semplificare la vita acquisendo una rappresentazione 3D dell'ambiente in fase di setup.

L'approssimazione potrebbe essere nell'ordine dei decimetri o nell'ordine dei km se si sbaglia ad interpretare una immagine.

6) Sistemi inerziali
Tecniche di **dead reckoning**: conosco la posizione iniziale e dopo uno spostamento riesco a calcolare la posizione attuale usando i dati della spostamento (velocità, accelerazione, ...). Gli accelerometri hanno sempre un errore che inquina il calcolo finale. Però più passa il tempo, più l'approssimazione è precisa.

Tecniche usate dai sommergibili; non vanno tanto bene per i dispositivi mobili.

7) Sistemi radio
Queste tecniche necessitano di una infrastrutturale ambientale e di hardware sul dispositivo. Sono adatte per applicazioni industriali.

UWB (Ultra Wide Band) è un segnale radio meno soggetto ad interferenze (a differenza del WiFi).
Ultrasound: il dispositivo emette ultrasuoni che vengono ricevuti da microfoni installati nell'ambiente.

8) Tecniche ibride (una qualunque combinazione delle precedenti tecniche)
La combinazione di tecniche dà risultati migliori.

Alcune combinazioni utili:
- Segnale radio + computer vision: prima calcolo l'area in cui mi trovo, poi il luogo preciso.
- Marcatori visivi + sensori inerziali: calcolo la posizione quando l'utente inquadra un marcatore, poi seguo gli spostamenti con sensori inerziali.

Calibrazione: quando una tecnica mi fornisce una buona approssimazione, uso questa informazione per calibrare le altre tecniche (analogo a quanto avviene per l'outdoor).

Lezione 7 – Organizzazione interna del codice

Il pattern MVC

È un pattern di progettazione ad alto livello per applicazioni con interfaccia grafica.
L'idea è di suddividere l'applicazioni in tre componenti (Model, View, Controller) ognuna con funzionalità specifiche che comunicano tra loro mediante interfacce.

- **Model**

Memorizza le strutture dati dell'applicazione e definisce le operazioni sui dati. In più: crea e salva oggetti nella memoria persistente; de/serializza dati per invio/ricezione via rete.
Esempio: ho la classe model ContactList con la lista dei contatti e l'operazione aggiungi contatto

Le classi del model sono riutilizzabili tra applicazioni che condividono la stessa logica: creo un solo model per la stessa app sviluppata per iOS e per Android.
Classi pre esistenti: in Java ci sono delle classi che rappresentano delle strutture dati (liste, code, ...).

- **View**

Si occupa di presentare all'utente i dati del model per permettergli di interagire con essi.
Le classi della view sono spesso fornite all'interno di framework.

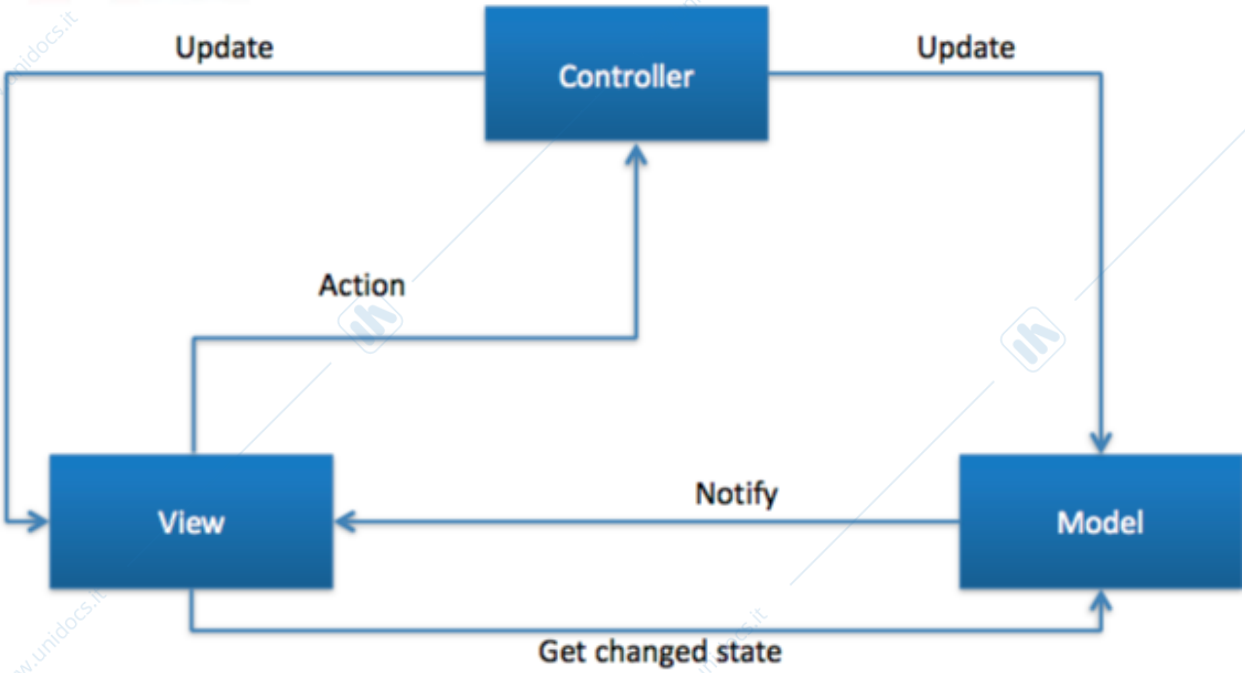
Gli oggetti della view si possono riutilizzare tra tutte le applicazioni che condividono gli stessi strumenti di interfaccia (anche se le funzionalità sono diverse).
Classi pre esistenti: esistono un sacco di framework.

- **Controller**

Gestisce il ciclo di vita dell'applicazione (start, stop, background); gestisce gli eventi generati dalla view e modifica di conseguenza il model; rileva le modifiche nel model e le comunica alla view.

Difficilmente il controller può essere riutilizzato o condiviso tra diverse app, perché rappresenta la logica applicativa.
Non esistono delle classi pre esistenti ma esistono dei problemi comuni nella programmazione controller. Alcuni framework che cercano di risolvere questi problemi: gli adapter aiutano a popolare una lista di elementi con il contenuto di una lista; gli AsyncTask aiutano a risolvere i problemi di sincronizzazione; la libreria Volley aiuta nelle comunicazioni HTTP senza bloccare l'interfaccia grafica.

Per riassumere:



Vantaggi del pattern MVC: utile come strumento concettuale per la progettazione del codice; aiuta a dare una struttura al codice; migliora il codice per leggibilità, mantenibilità e riusabilità.
Limiti del pattern MVC: raramente si ricorre a componenti separate nettamente, MVC è semplice e non si adatta ad applicazione complesse.

Componenti combinate:
- **View-Controller**: controller delegato alla gestione di una schermata (aggiornamento contenuti).
- **Model-Controller**: oggetto che svolge il ruolo di controller e memorizza dati specifici.
- **View-Model**: posso usarlo per gestire i dati di una singola schermata.