Location-Aware Applications

Introduction

Ing. MASSIMO FICCO

E-mail: **ficco@unina.it**

Context-aware computing

*Descrive la specifica capacità di un infrastruttura informativa di reagire al "**contesto**" [1].*

Il contesto dipendere da diversi fattori:

Esterni

- identità dell'utente;
- posizione fisica;
- condizioni del tempo;
- data e ora del giorno;
- ….. se l'utente cammina o guida;
- etc..

Context-aware computing

Interni

- carica residua del dispositivo;
- display del dispositivo;
- tipo di connettività (*wireless* o *wirline*);
- etc.

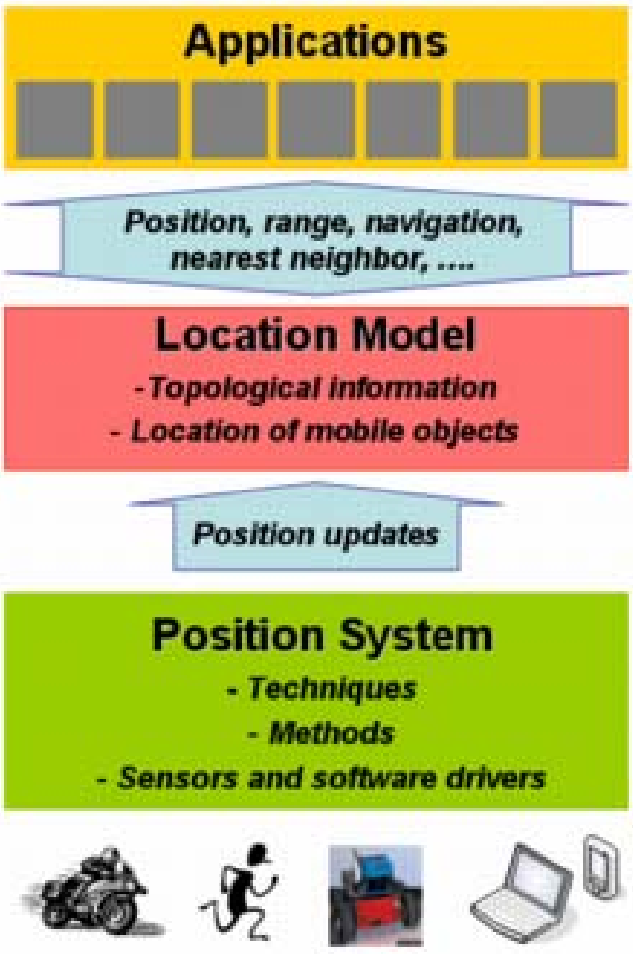
Location-aware Applications

L'evoluzione dei computer mobili, delle tecnologie di *location-sensing*, e le reti *wireless* hanno portato alla nascita di una nuova classe di applicazioni: ***Location-Aware Applications***.

Esempi:

- guide museali;
- servizi di emergenza (*911*);
- servizi di *tracking* (es. sistemi di automazione industriale);
- servizi di navigazione;
- etc.

Modello di un *Location System*



Un ***Location System*** è strutturato su più livelli. In particolare è composto da 4 componenti:

- *Le applicazioni*
- *Interfaccia di programmazione (API)*
- *Location Model*
- *Position System*

Le applicazioni

Le ***applicazione*** utilizzano le informazioni di *Location* ottenute dai livelli sottostanti per:

- determinate l'esatta posizione del dispositivo mobile;
- il percorso per raggiungere un certa destinazione;
- la scoperta di tutti gli oggetti all'interno di una certa area;
- ottenere informazioni circa oggetti nei dintorni di una certa posizione;
- etc..

Application Programming Interface (API)

Le **API** sono un interfaccia di programmazione ad alto livello che permette lo sviluppo di applicazioni di *location* indipendenti dalla tecnologia sottostante.

☞ L' *Open Mobile Alliance* (OMA) *Location Working Group* (LOC), che contiene il lavoro del *Location Interoperability Forum* (LIF), definisce delle specifiche per l'interoperabilità (*end-to-end*) di *Mobile Location Services* [2].

☞ *Java Community Process* (JCP) ha definito una *Java Specification Requests* (JSRs) per lo sviluppo di applicazioni **(JSR-179)** [3] nei dispositivi *Connected Limited Device Configuration* (CLDC).

Location Model

E’ uno schema per rappresentare le informazioni di *Location* [4].

- ***Geometric Location Model***: le informazioni di *location* sono rappresentate tramite coordinate.
- ***Symbolic Location Model***: vengono utilizzate coordinate simboliche.
- ***Hybrid Location Model***: coordinate simboliche e coordinate geometriche.

Symbolic Location Model

- ***Set-based models***: utilizza un set di location simboliche (es. le stanze di un edificio)
- ***Hierarchical models***: utilizza relazioni di contenimento (es. edificio - piano - stanza).
- ***Graph-based models***: le coordinate sono rappresentate come vertici di un grafo (relazioni di connessione tra location).

Position System

Il ***sistema di positioning*** aggiorna le informazioni di *Location*. L'output di tali sistemi vengono "*mappati*" sul modello utilizzato.

Differenti hardware, metodi e tecniche sono utilizzate per derivare differenti fenomeni fisici e logici:

- misure GPS;
- *time of flight* di emissioni ad ultrasuono;
- *Beacon* di prossimità;
- etc.

Le tecnologie

Le principali **tecnologie** utilizzate per derivare i fenomeni fisici e logici (*row data*) sono:

- per scenari outdoor:
  - GPS
  - GSM
  - etc.
- per scenari indoor:
  - 802.11 wireless LAN
  - Bluetooth
  - IRDA
  - Ultrasuoni
  - RFID
  - etc.

Metodi

I **metodi** rappresentano algoritmi per trasformare *raw sensor data* in misure canoniche secondo una rappresentazione nota: angoli, prossimità, distanza, posizione, etc.

Queste informazioni possono essere rappresentate in forme differenti:

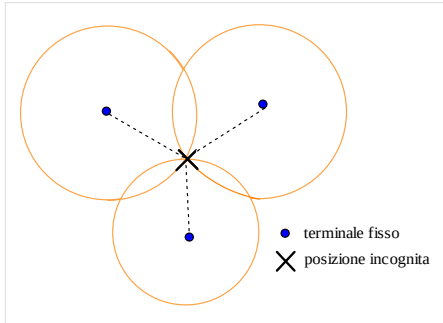
- geometriche (assolute o relative)
- simboliche

Differenti metodi:

- *time of arrival (TOA);*
- *time different of arrival (TDOA);*
- *angle of arrival (AOA);*
- *signal strength indication (SSI).*

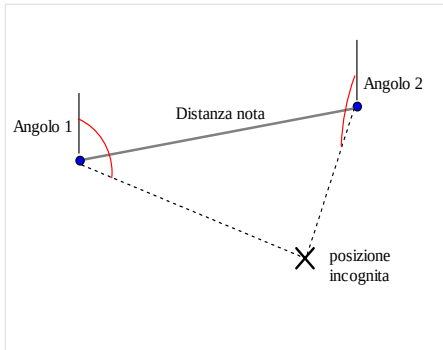
Tecniche

Le **tecniche** combinano differenti tipi di misure (es. misure di distanza con la posizione dei sensori) [5].



Le tecniche più importanti:

- ***triangulation***
  - *Lateration* e *angulation;*
- ***scene analysis***
  - basandosi sulle caratteristiche dell'ambiente di riferimento;
- ***proximity***
  - vicinanza rispetto a delle locazioni note.



Requisiti di un *Positioning System*

I requisiti per lo sviluppo di un *Positioning System* sono:

- accuratezza (stima dell'errore di localizzazione);
- precisione (funzione di distribuzione dell'errore);
- scalabilità (numero di oggetti che possono essere localizzati con una certa infrastruttura);
- copertura (indoor/outdoor);
- complessità nella fase di configurazione (in termini di tempo);
- costi (per l'infrastruttura e la configurazione);
- etc.

La privacy

La ***privacy*** definisce come e dove le informazioni di *Location* dell'utente vengono usate e salvate (tipo di restrizione applicata all'accesso a tali informazioni).

☞ ***Client-side***: sia la tecnica che la tecnologia di location è sul dispositivo mobile.

- Le informazioni di location sono sotto il controllo dell'utente.

☞ ***Server-side***: le informazioni di location sono determinate tramite un infrastruttura fissa.

- La privacy dell'utente è compromessa;
- Necessità di un *framework* per la distribuzione delle informazioni tramite opportune *policy*.

Location API per *Java 2 Micro Edition*

Java Community Process (JCP) ha recentemente emesso una *Java Specification Requests* (JSRs) per sviluppare applicazioni di *Location* per *Connected Limited Device Configuration* (CLDC).

Le specifiche ***JSR-179*** definiscono un *package* opzionale che fornisce le seguenti principali funzioni:

- ottenere la posizione e l'orientamento dei dispositivi mobili;
- accedere ad un *Landmark database*;

Location API per *Java 2 Micro Edition*

Ogni funzionalità utilizza specifici oggetti:

Location class:

rappresenta l'insieme di informazioni di *location* del dispositivo

Landmark class:

rappresentano location conosciute

L'oggetto *Location*

L'oggetto *Location* contiene le seguenti informazioni:

- coordinate e loro accuratezza;
- velocità e direzione (se disponibili);
- *timestamp* (che indica il momento in cui è fatta la misura);
- informazioni sul metodo di localizzazione utilizzato (*Angle-ofArrival*, *Satellite*, *Short-Range*, *Time-of-Arrival*….)

L'oggetto *Location*

L'oggetto *Location* può anche contenere un oggetto *AddressInfo* che contenga informazioni di tipo testuale sulla posizione. Tali informazioni sono suddivise nei seguenti campi:

- strada;
- codice postale;
- città;
- nazione;
- distretto;
- nome edificio;
- piano edificio;
- stanza edificio.

L'oggetto *Location*

Alle informazioni dell'oggetto *Location* si può accedere attraverso i metodi messi a disposizione come ad esempio:

- *getAddressInfo()*
- *getLocationMethod()*
- *getQualifiedCoordinates()*.

L'oggetto *Landmark*

Gli oggetti *Landmark* possono essere utilizzati per rappresentare luoghi frequentemente utilizzati (casa, ufficio ristoranti,…).

Gli oggetti *Landmark* hanno:

- un nome;
- una descrizione testuale;
- delle coordinate;
- un *AddressInfo* (opzionale).

Possono essere raggruppati in una singola o un insieme di categorie (un gruppo di *Landmark* simili per l'utente);

I *Landmark* sono memorizzate in una *repository* chiamata *Landmarkstore*.

L'oggetto *Landmark*

```
<?xml version="1.0" ?>
<Map>
        <Landmark>
          <name>Landmark1</name>
          <description>Stanza edificio</description>
          <latitude>45</latitude>
          <longitude>50</longitude>
          <altitude>0</altitude>
          <extension>Flat5</extension>
          <street>10 washington Street</street>
          <postal_code>12345</postal_code>
          <city>Palo Alto</city>
          <county>Santa Clara County</county>
          <state>California</state>
          <country>United States of America</country>
          <country_code>us</country_code>
          <district>distretto</district>
          <building_name>Edificio1</building_name>
          <building_floor>terzo</building_floor>
          <building_room>gialla</building_room>
          <building_zone>gialla</building_zone>
          <crossing1>crossing</crossing1>
        </Landmark>
        ......
```

Location API per *Java 2 Micro Edition*

La classe *LocationProvider* rappresenta un modulo capace di determinare la posizione del terminale (può essere implementato attraverso un qualsiasi tecnica, metodo e tecnologia di *location*).

Ogni dispositivo può avere più *LocationProvider* installati, ognuno associato ad un differente *Position System.*

La classe *Criteria* permette di specificare il criterio per scegliere il miglior *LocationProvider* che soddisfi i requisiti dell'utente.

La classe *Criteria*

la classe *Criteria* che consente di definire un insieme di parametri che rappresentino un criterio di selezione del *LocationProvider*.

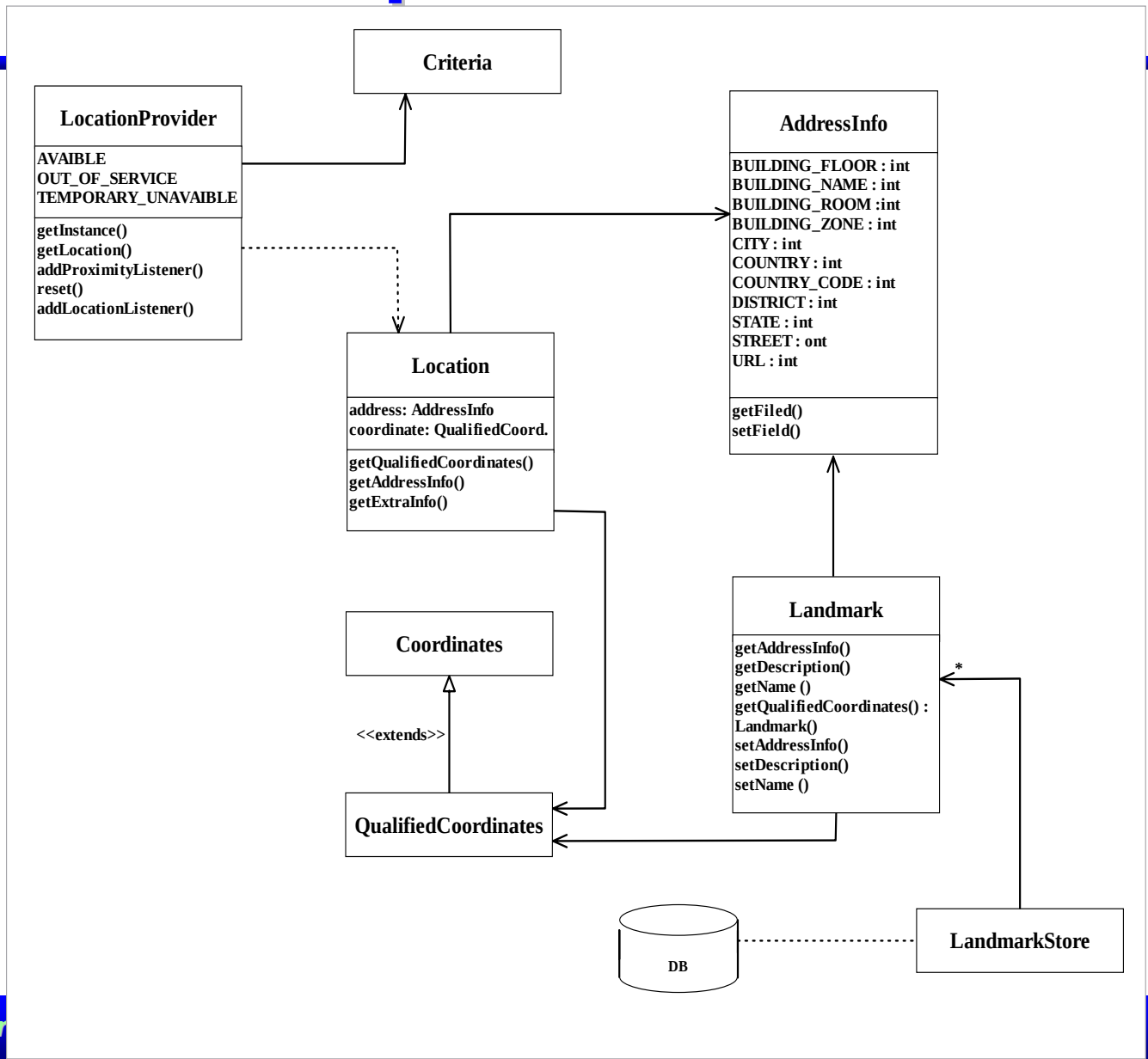
Criteria Field	*Default value*
Horizzontal accuracy	*NO_REQUIREMENT*
Vertical accuracy	*NO_REQUIREMENT*
Preferred response time	*NO_REQUIREMENT*
Power consumption	*NO_REQUIREMENT*
Cost allosew	*Tue (allowed to cost)*
Speed required	*False (not required)*
Altitude required	*False (not required)*
Address info required	*False (not required)*

Location API per *Java 2 Micro Edition*

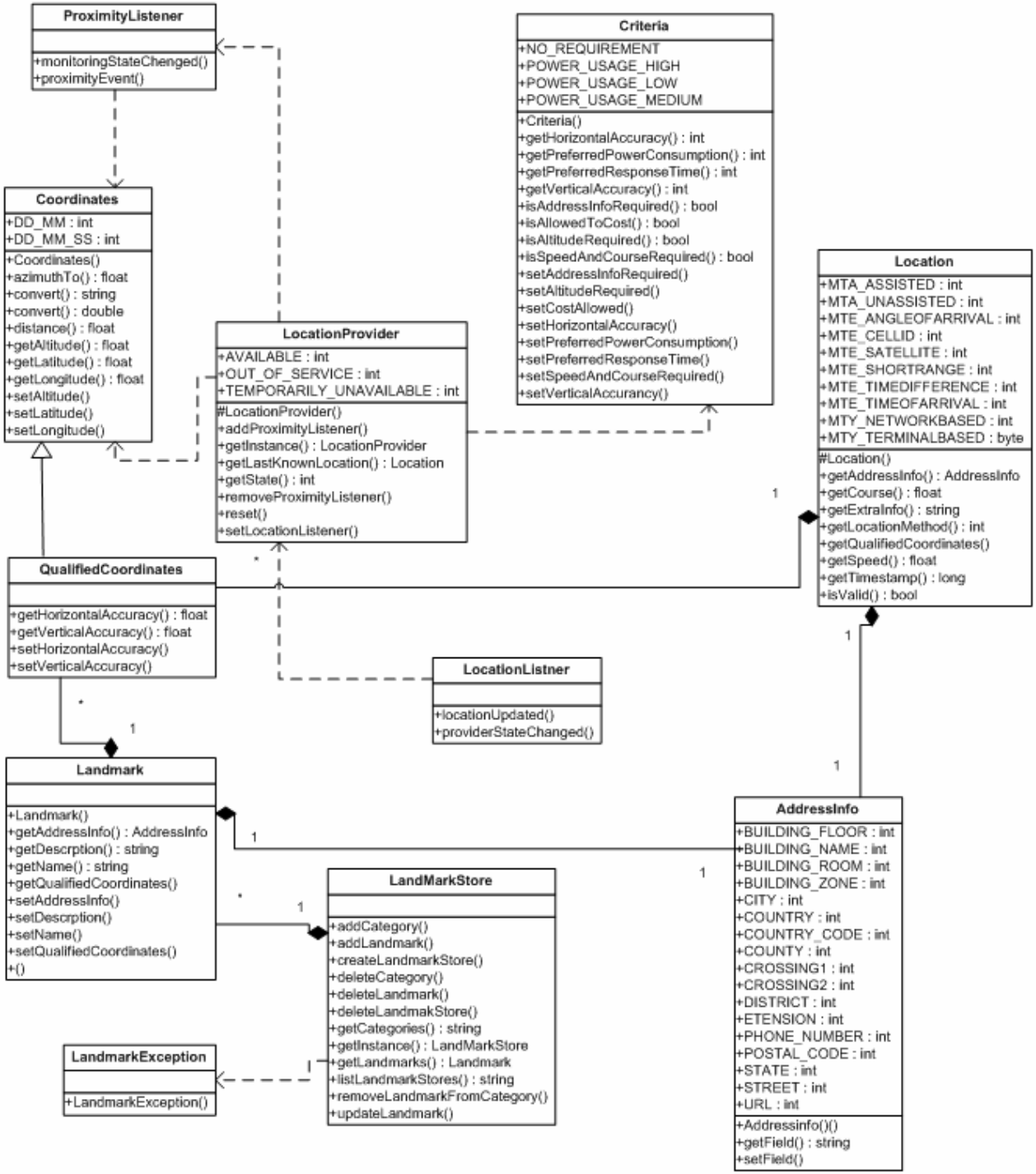
Le applicazioni possono ottenere le informazioni di *Location* in modo periodico o tramite *query* asincrone.

Per gli aggiornamenti periodici viene utilizzato un *LocationListener* (le applicazioni implementano questa interfaccia ed ricevono, tramite essa, informazioni regolari generate da un particolare *LocationProvider*).

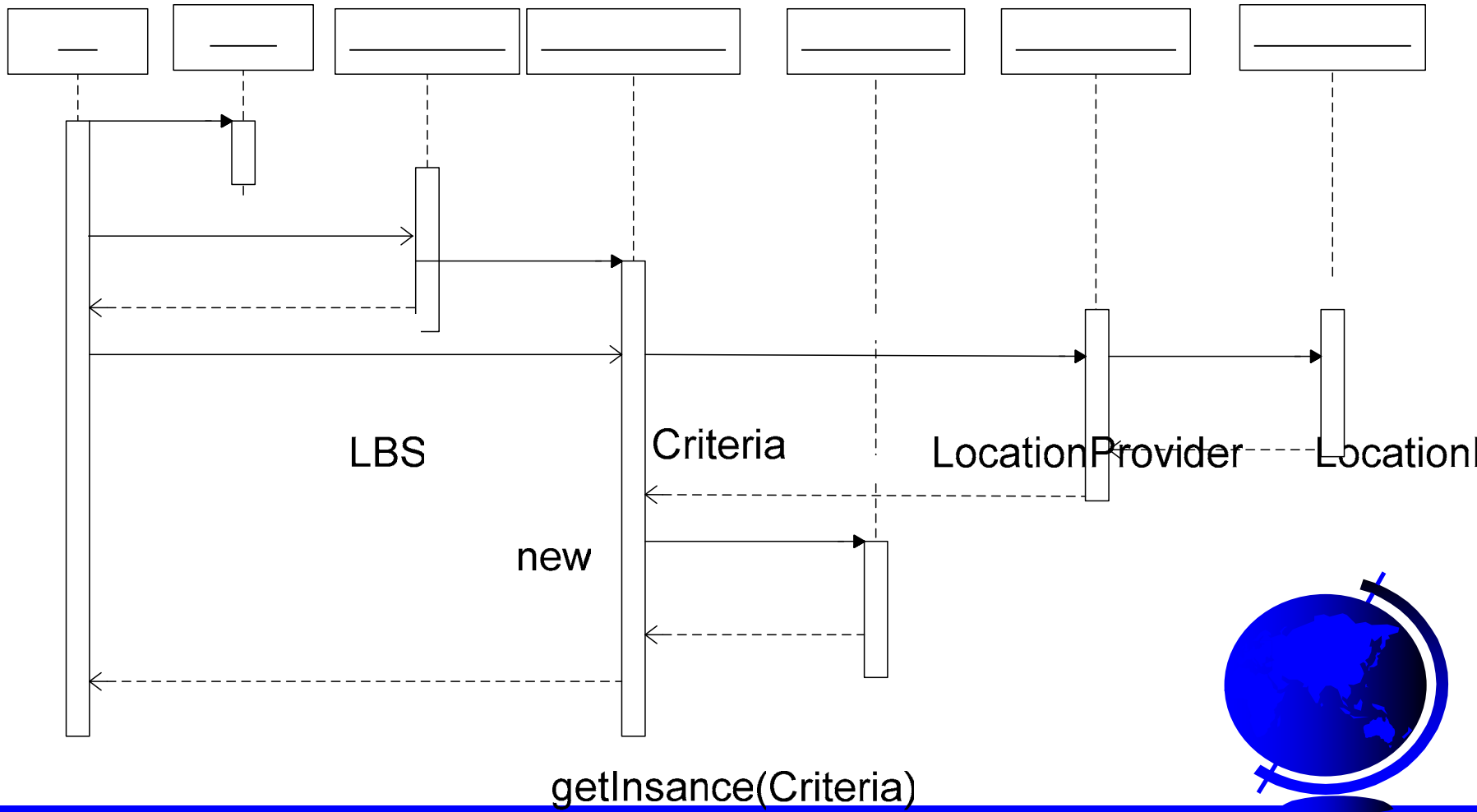
Location API per Java 2 Micro Edition



Loca dition


ProximityListener
+monitoringStateChenged()
+proximityEvent()
Criteria
+NO_REQUIREMENT
+POWER_USAGE_HIGH
+POWER_USAGE_LOW
+POWER_USAGE_MEDIUM
+Criteria()
+getHorizontalAccuracy() : int
+getPreferredPowerConsumption() : int
+getPreferredResponseTime() : int
+getVerticalAccuracy() : int
+isAddressInfoRequired() : bool
+isAllowedToCost() : bool
+isAltitudeRequired() : bool
+isSpeedAndCourseRequired() : bool
+setAddressInfoRequired()
+setAltitudeRequired()
+setCostAllowed()
+setHorizontalAccuracy()
+setPreferredPowerConsumption()
+setPreferredResponseTime()
+setSpeedAndCourseRequired()
+setVerticalAccurancy()
Coordinates
+DD_MM : int
+DD_MM_SS : int
+Coordinates()
+azimuthTo() : float
+convert() : string
+convert() : double
+distance() : float
+getAltitude() : float
+getLatitude() : float
+getLongitude() : float
+setAltitude()
+setLatitude()
+setLongitude()
LocationProvider
+AVAILABLE : int
+OUT_OF_SERVICE : int
+TEMPORARILY_UNAVAILABLE : int
#LocationProvider()
+addProximityListener()
+getInstance() : LocationProvider
+getLastKnownLocation() : Location
+getState() : int
+removeProximityListener()
+reset()
+setLocationListener()
Location
+MTA_ASSISTED : int
+MTA_UNASSISTED : int
+MTE_ANGLEOFARRIVAL : int
+MTE_CELLID : int
+MTE_SATELLITE : int
+MTE_SHORTRANGE : int
+MTE_TIMEDIFFERENCE : int
+MTE_TIMEOFARRIVAL : int
+MTY_NETWORKBASED : int
+MTY_TERMINALBASED : byte
#Location()
+getAddressInfo() : AddressInfo
+getCourse() : float
+getExtraInfo() : string
+getLocationMethod() : int
+getQualifiedCoordinates()
+getSpeed() : float
+getTimestamp() : long
+isValid() : bool
QualifiedCoordinates
+getHorizontalAccuracy() : float
+getVerticalAccuracy() : float
+setHorizontalAccuracy()
+setVerticalAccuracy()
LocationListner
+locationUpdated()
+providerStateChanged()
Landmark
+Landmark()
+getAddressInfo() : AddressInfo
+getDescrption() : string
+getName() : string
+getQualifiedCoordinates()
+setAddressInfo()
+setDescrption()
+setName()
+setQualifiedCoordinates()
+()
LandMarkStore
+addCategory()
+addLandmark()
+createLandmarkStore()
+deleteCategory()
+deleteLandmark()
+deleteLandmakStore()
+getCategories() : string
+getInstance() : LandMarkStore
+getLandmarks() : Landmark
+listLandmarkStores() : string
+removeLandmarkFromCategory()
+updateLandmark()
LandmarkException
+LandmarkException()
AddressInfo
+BUILDING_FLOOR : int
+BUILDING_NAME : int
+BUILDING_ROOM : int
+BUILDING_ZONE : int
+CITY : int
+COUNTRY : int
+COUNTRY_CODE : int
+COUNTY : int
+CROSSING1 : int
+CROSSING2 : int
+DISTRICT : int
+ETENSION : int
+PHONE_NUMBER : int
+POSTAL_CODE : int
+STATE : int
+STREET : int
+URL : int
+Addressinfo()()
+getField() : string
+setField()

Sequence Diagram: richiesta *location*



Bibliografia

[1] Patterson, C.A., Muntz, R.R., Pancake, C.M.: Challenges in Location-Aware Computing, in IEEE Pervasive Computing, 2(2), pp. 80-89. IEEE CS Press (2003).

[2] FiCom Location API Working Group: FiCom Location API 2.0 Interface specification, 2002.

[3] Java Community Process: Location API for J2ME Specification 1.0 Final Release (2003).

[4] Becker, C., and Durr, F.: On location models for ubiquitous computing, in Personal and Ubiquitous Computing (2005), Springer-verlag LNCS, vol. 9(1), pp. 20-31, (2005).

[5] Hightower, J., Borriello, G.: Location Sensing Techniques, in Technical report UW-CSE-01-07-01. University of Washington (2001).

[6] Beresford, A., and Stajano, F.: Location Privacy in Pervasive Computing, in IEEE Pervasive Computing, 2(2), pp. 46-55. IEEE CS Press (2003).