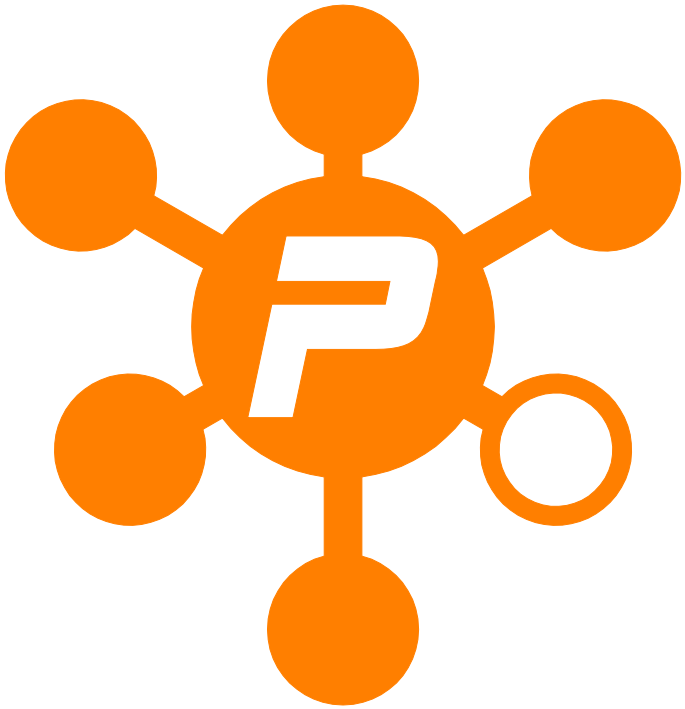
Università degli Studi di Salerno
Corso di Ingegneria del Software

Unisa Park
ODD
Versione 1.5



Data: 04/02/2017

Partecipanti:

Nome	Matricola
Maria Truvolo	0512103176
Federico Vastarini	0512103294

Scritto da:	Maria Truvolo e Federico Vastarini

Revision History

Data	Versione	Descrizione	Autore
14/01/2017	1.0	Prima stesura del documento.	Maria Truvolo
16/01/2017	1.1	Definizione del server remoto di gestione.	Federico Vastarini
19/01/2017	1.2	Definizione del server remoto dei dati.	Federico Vastarini
20/01/2017	1.3	Definizione del server locale.	Maria Truvolo
28/01/2017	1.4	Riferimenti per la tracciabilità dei requisiti	Maria Truvolo
04/02/2017	1.5	Revisione del testo	Maria Truvolo

Indice

1. Introduzione

1.1 Object Design Trade-offs (SDD 1.2)

1.1.1 Performance vs Centralization

Si è scelto di utilizzare un'architettura distribuita in modo da ridurre ed allo stesso tempo gestire il carico di lavoro delle macchine coinvolte. Il sistema non risiede quindi interamente nella sola università di Fisciano. *(RAD 5)*

1.1.2 Response time vs Scalability

Per garantire la più rapida risposta in alcuni casi si è preferito un approccio procedurale anziché ad oggetti. *(RAD 6.1)*

1.1.3 Clarity vs Dependance

Si è anteposta la scelta di sviluppare codice proprietario piuttosto che affidarsi a pacchetti di software prefatti, riducendo il prodotto software ad una mera accozzaglia di codici dipendenti. Questo garantisce inoltre la massima linearità e trasparenza nello sviluppo.

1.1.4 Modernity vs Stability

Essendo un progetto volto alla sperimentazione di nuove tecnologie ci si è affidati agli standard più recenti nello sviluppo dell'interfaccia web del client pur rimanendo nei limiti richiesti per una convalida efficace del codice.

1.2 Convenzioni e linee guida

1.2.1 Disponibilità dell'interfaccia

Il software è stato sviluppato per mantenere un'interfaccia coerente su tutti i browser

di desktop, tablet e cellulari.

Ci si è strettamente attenuti allo standard W3C nella stesura del codice HTML5 e CSS3.0. La regolarità di tutti i moduli del sistema è verificabile online tramite i validatori presenti sul sito stesso del W3C. *(RAD 3.2) (SDD 1.2)*

Ci si è accertati della sua perfetta funzionalità di visualizzazione e navigazione sui seguenti browser:

Desktop:

Mozilla Firefox
Internet Explorer
Chrome
Midori

Tablet:

Samsung Galaxy Tab S2

Mobile:

Chrome per Samsung
Browser interno di Samsung
Chrome per IPhone
Browser interno di IPhone
Browser interno di Huawei

Il codice Javascript è stato testato e verificato (sebbene su siti non ufficiali) per tutte le suddette interfacce.

1.2.2 Performance dell'interfaccia

Si è provveduto ad utilizzare tutte le pratiche di minimizzazione e compressione dati per garantire il delivery dei contenuti più rapido possibile seppur mantenendo una separazione coerente dei linguaggi utilizzati.

I test effettuati con Google Insight risultano in 8 regole passate su 10 ed un punteggio di 79/100 per mobile e 93/100 su desktop.

Si è scelto di mantenere il totale di byte necessari allo stream per la visualizzazione dell'intera mappa unitamente all'interfaccia per utilizzarla al di sotto dei 25K. *(RAD 2.3, 6.1, 6.3, 6.4) (SDD 1.2)*

1.2.3 Ottimizzazione delle immagini e dei contenuti

La scelta delle immagini, data la natura stessa del progetto è ricaduta sul formato SVG. Tutti i file sono stati rielaborati e corretti manualmente tramite codice.
L'intera interfaccia è stata inoltre testata per una perfetta comprensione da parte di individui affetti da daltonismo. *(RAD 3.2)*

1.2.4 Sicurezza

Sebbene sia soltanto un progetto di esempio sono state mantenute tutte le procedure minime per garantire la sicurezza del server e dei dati presenti su di esso. Seppure non appoggiandosi ad un sistema centralizzato di gestione e filtraggio degli stream dei dati, sono state incluse nelle chiamate critiche di accesso al database o di lettura delle risorse inviate dal client tutte le funzioni necessarie ad accertarsi che i dati siano puliti. *(SDD 2.5.2)*

1.2.5 Nomenclatura

Sono state utilizzate le regole più comuni relative ai seguenti linguaggi utilizzati: java, php, html5, css3, xml, javascript. Sono stati inoltre standardizzati e minimizzati i formati dei file .ini e .htaccess relativi alla gestione interna del server remoto. *(SDD 2.3)*

1.2.6 Struttura dati

La struttura dati utilizzata dal server remoto è composta da tabelle standard in MySQL accessibili da una classe di gestione nel server principale e da script autonomi nel server dati.
La struttura dati nel server locale si realizza invece con un buffer di byte di basso livello. Questa decisione consente di avere l'accesso più rapido possibile alla struttura essendo il buffer stesso al di fuori della portata del garbage collector. Il buffer viene manualmente distrutto alla fine del suo utilizzo. *(RAD 7.1) (SDD 2.4)*

1.2.7 Dialogo tra le parti

Il dialogo tra le parti componenti l'intero software sono state realizzate nel modo seguente: *(RAD 2.3) (SDD 2.1)*

Server Remoto Principale - Client: standard http (First Contact ed AJAX).

Server Remoto Dati - Client: standard http (AJAX).

Server Remoto Principale - Server Remoto Dati: shared database e CORS via Client.

Server Remoto Dati - Server Locale: protocollo base64+crc32+ack su http.

Server Locale - Rete di sensori: da definire.

1.2.8 Protocolli

Per la connessione del server remoto e locale si è optato per un tipo di connessione su protocollo http dove, a prescindere dall'header standard di dimensioni quanto più possibile contenute, è stato realizzato un pacchetto composto da una stringa di dati in base64 ed il suo relativo codice di ridondanza ciclico crc32. Questo permette il controllo del pacchetto trasmesso ed inoltre la possibilità di scambiare un ack coerente al contenuto ricevuto. *(RAD 2.3) (SDD 2.5.1)*

1.2.9 Documentazione del codice

E' stata fornita la documentazione completa del codice riga per riga nel rispetto della diversità di tutti i linguaggi di programmazione utilizzati con un tentativo di conciliarne le differenze. Sono stati omessi soltanto i blocchi di codice rappresentanti "dati" più che parti funzionali. *(RAD 6.4)*

1.3 Definizioni

- **Core**

Si intende per core del server remoto l'insieme dei files atti a garantire le funzionalità generiche di base relative ad esempio alla sicurezza, alla lettura e scrittura dei dati, di qualsiasi script richiesto dal client dal suo inizio alla sua fine.

- **Modulo**

Nell'approccio modulare della piattaforma del server remoto principale si intende per modulo un blocco di codice richiamato dal sistema risiedente su uno o più file che

caratterizza una specifica azione del server, senza alterarne le funzionalità di base.

- **Sistema di reindirizzamento delle chiamate**

E' il sistema che, a diversi livelli, si occupa di reindirizzare e gestire chiamate sincrone ed asincrone utilizzando il core come hub.

- **"Main" Frame**

Il container responsabile del delivery dei contenuti del client.

- **Mappa generata dinamicamente**

Per mappa dinamicamente generata si intende la componente sperimentale in SVG composta dinamicamente dal server remoto ed aggiornata dal client tramite chiamate asincrone. (Si è tralasciato lo sviluppo della dinamicità delle parti fondamentali delle mappe stesse essendo esso parte di un'interfaccia di amministrazione, non di utenza).

- **Contenuto pseudo-statico**

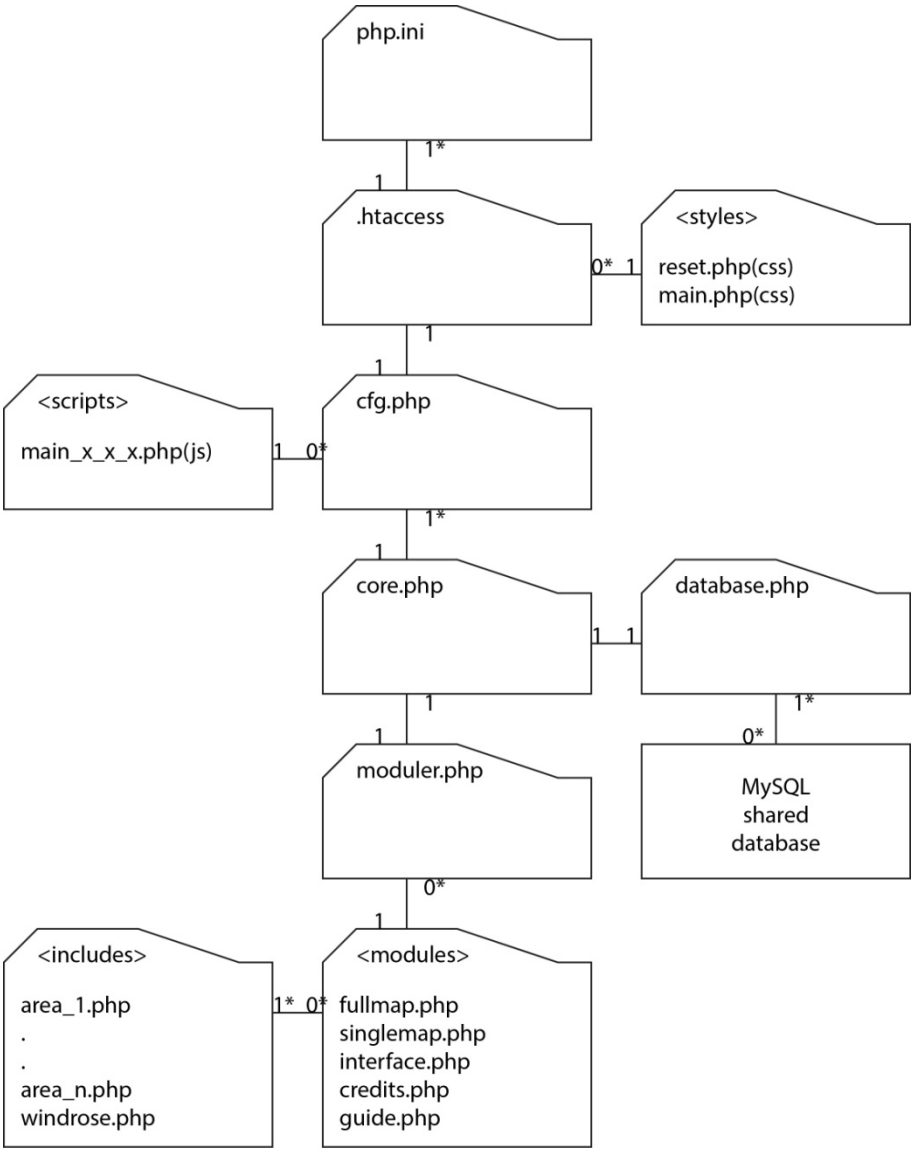
L'insieme di contenuti generati dal sistema che pur non essendo totalmente statici nel senso canonico della parola si possono considerare tali data la scarsa se non nulla frequenza di aggiornamento.

2. Package

Il software distribuito si compone di 3 pacchetti fondamentali (il layer dei sensori viene considerato al di fuori dello scopo della produzione di questo software, essendo lo scopo stesso del software quello di adattarsi al meglio ai sensori e non viceversa).

2.1 Server remoto di gestione del sistema

Il server remoto di gestione si basa su un modello modulare proprietario e si compone delle parti proposte nel seguente schema: *(SDD 2.2.3)*



- **php.ini**: File con procedure di configurazione del server per adattarsi alle necessità della piattaforma.
- **.htaccess**: File di configurazione, gestione e reindirizzamento trasparente degli accessi a livello pre-applicazione.
- **cfg.php**: File di configurazione dell'applicazione.
- **core.php**: Core script.
- **database.php**: Classe di gestione del database e delle funzionalità specifiche dell'applicazione.
- **moduler.php**: Classe di gestione del reindirizzamento e filtraggio dei moduli.
- **interface.php**: Modulo di creazione dell'interfaccia pseudo-statica del Main Frame del client.
- **fullmap.php**: Modulo di creazione della mappa completa generata dinamicamente. *(RAD 8.2.1)*
- **singlemap.php**: Modulo di creazione, gestione e reindirizzamento al caricamento delle singole mappe generate dinamicamente. *(RAD 8.2.2)*
- **credits.php**: Modulo per la visualizzazione delle informazioni pseudo-statiche sul software.
- **guide.php**: Modulo per la visualizzazione della guida pseudo-statica all'utilizzo del software.
- **main_x_x_x.php**: Modulo di gestione dello script progressivo del client.
- **area_x.php**: File per la creazione delle informazioni dinamiche relative alla mappa numero x.
- **windrose.php**: File per la creazione di elementi grafici statici sulle mappe.
- **reset.php**: File di standardizzazione degli stili nelle interfacce client.
- **main.php**: File di stile principale per la realizzazione dell'interfaccia crossbrowser.

Essendo sviluppato in php la struttura dell'applicazione non è né strettamente object oriented né mappata su un modello MVC. Si propone comunque l'analisi delle interfacce delle due classi di gestione del database e dei moduli.

2.1.1 Gestore del database del progetto

Nome:
Database

Descrizione:
Classe di gestione del database del server remoto principale. Fornisce funzioni di connessione al database e di lettura e scrittura di dati relativi al progetto. Richiede per il corretto funzionamento la corretta configurazione nel file cfg.php dei parametri di connessione ad un database MySQL.

Dipende da:
mysqli
per la realizzazione della connessione e delle query al database MySQL

Campi:
private **$connection**
Object. Definisce la connessione corrente attiva

private **$tablePrefix**
String. Definisce il prefisso utilizzato per le tabelle. Può essere vuota.

public **$resultArray**
String[]. array di stringhe di dati relativa al risultato di una query di lettura.

public **$resultCount**
integer. Numero di risultati ottenuti da una query.

Metodi:
function **__construct**()
metodo di default per l'inizializzazione del processo di apertura di una connessione al database

private function **selectRows**($f_query).
metodo principale per l'estrazione di dati dal database. Richiede come unico

parametro la stringa di selezione in formato MySQL. Restituisce, una volta popolata, la **$resultArray**.

public function **getAreas**()
restituisce un'array contenente le informazioni relative a tutte le aree di parcheggio.

public function **getArea**($i)
restituisce un'array contenente i dati relativi all'area contrassegnata dall id $i.

public function **getSpots**($i)
restituisce un'array contenente i dati relativi agli slot dell'area contrassegnata dall id $i.

public function **escapeString**($f_string)
prende un parametro stringa e restituisce la stringa con l'escape sicuro in modo che possa essere inclusa in una query.

2.1.2 Gestore dei moduli del progetto

Nome:
Database

Descrizione:
Classe di gestione del database del server remoto principale. Fornisce funzioni di connessione al database e di lettura e scrittura di dati relativi al progetto. Richiede per il corretto funzionamento la corretta configurazione nel file cfg.php dei parametri di connessione ad un database MySQL.

Dipende da:
mysqli
per la realizzazione della connessione e delle query al database MySQL

Campi:
private **$currentModule**
String. Il modulo valido corrente.

Metodi:

public function **getModule**()

si accerta della validità del modulo richiesto dal client tramite chiamata diretta od asincrona e, nell'eventualità, aggiorna il campo **$currentModule** in modo da fornire al core il modulo corretto e quindi intraprendere un'azione od inviarlo al client.

2.2 Server remoto di gestione dei dati

Il server remoto di gestione dei dati si basa su modelli di file atomici.

Ogni file è quindi un'entità a sé stante e si occupa di espletare la propria funzione in modo rapido e completo. Si è scelto quest'approccio data la necessita di fornire servizi estremamente eterogenei tra dialoghi verso il client e dialoghi verso il server locale. Inoltre qualsiasi aggiunta di chiamate ridondanti non farebbe altro che rallentare la semplicità di esecuzione del codice stesso, che in questo caso risulta critica.

Nel caso in cui il server non sia raggiungibile lo script del client è istruito a mostrare un messaggio relativo al malfunzionamento in tutte le aree in cui sia necessaria la connessione a dati non pseudo-statici. *(RAD 6.2, 6.3, 8.2.3) (SDD 2.2.3)*

Gli elementi costituenti questo server sono:

.htaccess

realizza la funzionalità CORS, che altrimenti sarebbe negata all'intero sistema rendendo impossibili le chiamate asincrone.

getareas.php

restituisce al client il numero di spots occupati per ogni area, accedendo alla lettura dell'ultimo stream in base64 ricevuto dal server locale. *(RAD 8.2.1)*

getbitmap.php

restituisce l'ultimo stream in base64 ricevuto dal server locale. Viene richiamato nella visualizzazione delle singole aree in modo da evidenziare gli spot liberi sull'interfaccia del client. *(RAD 8.2.2)*

setbitmap.php

riceve lo stream in base64 tramite il GET di una connessione http del server locale in modo "connectionless".*(RAD 8.2.4)*

setbitmapack.php

riceve lo stream dei dati dal server locale secondo il protocollo stabilito e, nel caso i dati siano coerenti con il codice di ridondanza ciclico, salva lo stream di dati e restituisce lo stesso codice crc. In caso contrario restituisce un codice di errore

utilizzabile dall'updater del server locale per intraprendere eventuali azioni. *(RAD 8.2.4)*

gettime.php

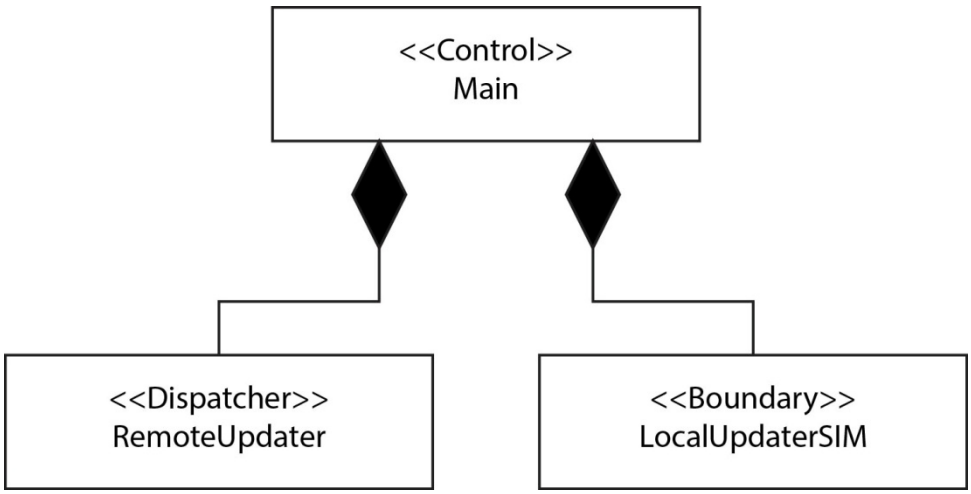
fornisce una proiezione di quando un'area avrà posti liberi disponibili nel caso sia completamente occupata. La funzione statistica da utilizzare è attualmente in fase di studio. (Nell'attesa di ricevere dati reali è stato comunque aggiunto un placeholder in modo da non pregiudicare il funzionamento del resto del sistema). *(RAD 8.2.9)*

2.3 Server locale

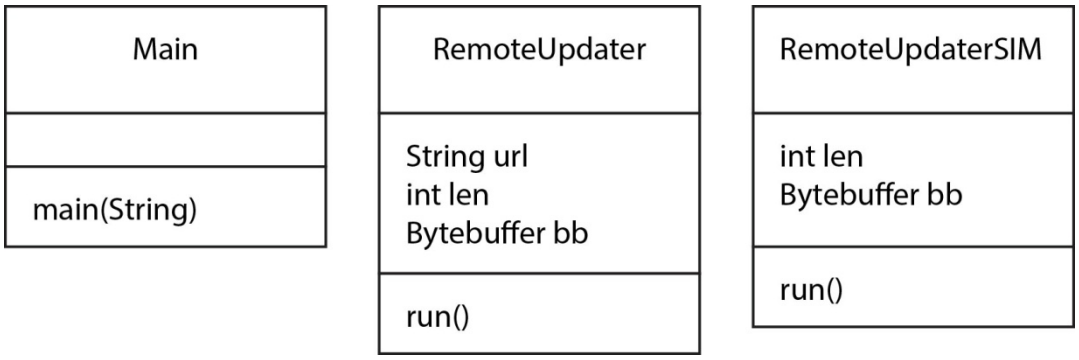
Il server locale è completamente scritto in Java 7 senza l'aiuto di alcun IDE e si basa sullo standard MVC. Esso si occupa di strutturare il layer tra il server remoto e la rete di sensori. Allo stato attuale del progetto la classe di connessione alla rete di sensori è stata sostituita da una classe in grado di simulare il loro stato. *(SDD 2.2.2)*
Esso è composto nel modo seguente:

Elenco delle classi:
Main.class
RemoteUpdater.class
LocalUpdaterSim.class

Overview del package e delle relazioni interne:



Overview delle classi:



Tutti files sono predisposti per la creazione di una guida dal comando javadoc.
Segue una descrizione dettagliata delle singole classi.

2.3.1 Main

Nome:

Main

Descrizione:

Classe principale del software di server locale. Contiene il metodo main e realizza la creazione dei due threads di interfaccia con il server remoto e la rete di sensori. E' inoltre responsabile della creazione del byte buffer di basso livello e della sua distruzione prima della chiusura del processo.

Package:

localServer

Dipende da:

java.nio.ByteBuffer

per la realizzazione del buffer dei dati di basso livello al di fuori della portate del garbage collector.

java.lang.IllegalArgumentException

per la gestione dell'eccezione lanciata dalla creazione del buffer nel caso non venga passato un intero.

localServer.LocalUpdaterSim

per la creazione del thread di interfacciamento con il server remoto.

localServer.RemoteUpdater

per la creazione del thread di interfacciamento con il layer di sensori.

Costanti:

int **dataLength**

definisce la lunghezza del buffer utilizzato per la rappresentazione in bitmap dello stato dei parcheggi. Deve essere equivalente alla costante sul server remoto per la

ricezione e gestione dei dati. Nel caso del progetto Unisapark si è scelto di utilizzare un valore di 1024, pari alla gestione di 8192 posti di parcheggio simultaneamente.

Metodi:
void **main**(String[] args)
metodo di default per l'inizializzazione del processo.

2.3.2 Remote Updater

Nome:
RemoteUpdater

Descrizione:
Classe relativa alla connessione e scambio dati con il server remoto. In modo rapido ed efficace legge lo stato attuale della bitmap dal byte buffer, trasforma i dati in una stringa base64, ne calcola il crc e trasmette tutto al server remoto. Dopodiché confronta l'ack ricevuto con lo stesso crc. *(RAD 9.1)*

Package:
localServer

Estende:
Thread

Dipende da:
java.nio.ByteBuffer
per la realizzazione del buffer dei dati di basso livello al di fuori della portate del garbage collector.

java.net.URL
per la creazione dell'oggetto URL contenente il riferimento alla risorsa del server remoto.

java.net.URLConnection

per la realizzazione della connessione al server remoto.

java.net.MalformedURLException
per gestione dell'eccezione nel caso in cui la stringa della risorsa non sia corretta.

java.io.InputStreamReader
per la lettura dello stream ricevuto dalla risorsa remota (ack).

java.io.BufferedReader
per la gestione della lettura dello stream di input ricevuto dalla risorsa remota.

java.io.IOException
per la gestione dell'eccezione dovuta ad errori di streaming per le classi InputStreamReader e BufferedReader.

java.util.zip.CRC32
per la creazione del codice di ridondanza ciclico associato alla stringa dei dati inviati.

org.apache.commons.codec.binary.Base64
per la creazione della stringa in Base64 associata ai dati in binario della bitmap.

Campi:
private String **url**
Definisce la stringa della risorsa remota alla quale connettersi per l'upstream dei dati.

private ByteBuffer **bb**
Definisce il riferimento al blocco di memoria di basso livello dalla quale prendere i dati.

private int **len**
Definisce la lunghezza del blocco di memoria.

Metodi:

public **RemoteUpdater**(String url, ByteBuffer bb, int len)

Costruttore di default della classe. Inizializza la stringa dell'url, il riferimento al buffer e la sua lunghezza.

public void **run**()

Standard override del metodo run della classe Thread. Questo metodo si occupa della generazione della stringa base64, del codice crc e dell'invio dei dati al server remoto. Il tutto avviene periodicamente ogni 5 secondi. Questo metodo è richiamato dal metodo main della classe Main ed interrotto all'uscita dallo stesso. *(RAD 8.2.4)*

2.3.3 Local Updater SIM

Nome:

LocalUpdaterSIM

Descrizione:

Classe relativa all'aggiornamento del buffer di dati e la simulazione dello stato dei sensori. Questa classe è utilizzata nel test case. Nel caso del deployment del sistema reale questa classe è sostituita dal quella di connessione alla rete di sensori.

Package:

localServer

Estende:

Thread

Dipende da:

java.nio.ByteBuffer

per la realizzazione del buffer dei dati di basso livello al di fuori della portate del garbage collector.

java.util.Random

per la realizzazione della simulazione dello stato casuale dei dati nel buffer.

Campi:

private ByteBuffer **bb**
Definisce il riferimento al blocco di memoria di basso livello dalla quale prendere i dati.

private int **len**
Definisce la lunghezza del blocco di memoria.

Metodi:
public **LocalUpdaterSIM**(ByteBuffer bb, int len)
Costruttore di default della classe. Inizializza il riferimento al buffer e la sua lunghezza.

public void **run**()
Standard override del metodo run della classe Thread. Questo metodo si occupa dell'aggiornamento del buffer simulando il traffico dell'area di parcheggio e lo stato progressivo dei sensori relativi alle aree. Il tutto avviene periodicamente ogni x secondi. Questo metodo è richiamato dal metodo main della classe Main ed interrotto all'uscita dallo stesso.

2.4 Client

Tralasciando l'aspetto grafico dell'interfaccia, l'analisi del client si riduce allo script in javascript relativo alle chiamate asincrone verso entrambi i server remoti con il solo scopo di popolare il Main Frame e le mappe caricate con i dati aggiornati.