Agenzia Nazionale per le Nuove Tecnologie,
l'Energia e lo Sviluppo Economico Sostenibile

Ministero dello Sviluppo Economico

RICERCA DI SISTEMA ELETTRICO

Realizzazione di una piattaforma integrata per il data fusion di segnali provenienti da sistemi sensoriali per applicazioni di smart city integrate nella rete della pubblica illuminazione

L. Sciavicco, S. Panzieri, G. Ulivi, F. Pascucci,
C. Foglietta, F. Moretti, P. Cicolin

S. Pizzuti

ROMA TRE
UNIVERSITÀ DEGLI STUDI

Report RdS/2011/323

REALIZZAZIONE DI UNA PIATTAFORMA INTEGRATA PER IL DATA FUSION DI SEGNALI PROVENIENTI DA SISTEMI SENSORIALI PER APPLICAZIONI DI SMART CITY INTEGRATE NELLA RETE DELLA PUBBLICA ILLUMINAZIONE

L. Sciavicco, S. Panzieri, G. Ulivi, F. Pascucci, C. Foglietta, F. Moretti, P. Cicolin (Dipartimento di Informatica e Automazione – Università degli Studi ROMA TRE),
S. Pizzuti (ENEA)

Novembre 2011

Report Ricerca di Sistema Elettrico
Accordo di Programma Ministero dello Sviluppo Economico – ENEA
Area: Razionalizzazione e risparmio nell'uso dell'energia
Progetto: Tecnologie "smart" per l'integrazione della illuminazione pubblica con altre reti di servizi energetici e loro ottimizzazione
Responsabile Progetto: Mauro Annunziato, ENEA

Sommario

Indice delle figure

Abstract

In questo documento viene spiegato lo scopo, la struttura e il funzionamento della piattaforma integrata “Smart Town” per la integrazione di smart services nella rete di illuminazione pubblica, realizzata da Roma Tre ed **ENEA.** Il documento spiega lo sviluppo attuale del sistema e i possibili sviluppi e quindi i miglioramenti da realizzare nel prossimo futuro.

La piattaforma che viene proposta si pone come il nodo delle comunicazioni fra le diverse infrastrutture presenti nella Smart City. L’idea è che la piattaforma sia indipendente dalle infrastrutture presenti come livello più basso, ma consenta l’acquisizione dei dati in maniera centralizzata per poterli analizzare ed eventualmente modificare il comportamento e quindi controllare le infrastrutture al di sotto.

L’utilità principale di questa piattaforma è, quindi, l’integrazione dei dati provenienti dai diversi sensori. Ogni infrastruttura ha una struttura verticale, poiché i dati provenienti dai sensori sono analizzati e acquisiti per attuare procedure di controllo del sistema, in maniera anche automatica, ma senza analizzare i dati di altri sistemi.

La piattaforma è una struttura orizzontale che consente la comunicazione tra i sistemi verticali, in modo da migliorare l’efficienza dei diversi sistemi sotto un punto di vista energetico, monetario e di welfare dei cittadini. La piattaforma è in grado di redistribuire ai ricercatori i dati che ha raccolto fino al momento attuale, ed è anche in grado di accorpare dati in maniera organica fornendo indici e analisi di cause probabili quando accadono guasti.

Al momento la piattaforma è stata pensata per acquisire i dati provenienti da tre infrastrutture: l’illuminazione pubblica, la rete di edifici e l’infrastruttura di mobilità. È stato realizzato un grande sforzo per cercare di rendere la piattaforma più generica possibile, in maniera che l’aggiunta di nuove infrastrutture non debba causare la ristrutturazione del codice. In caso di nuovi sistemi è necessario solamente scrivere un Thread, ossia una funzione ciclica, in grado di copiare i dati dal server del sistema a quello della piattaforma Smart Town.

Introduzione

La piattaforma integrata necessaria al progetto Smart Town è stata ispirata dalla struttura degli attuali sistemi **SCADA**. Un sistema SCADA (Supervisor Control and Data Acquisition) è composto principalmente da un DataBase su cui sono scritti e letti tutti i valori necessari a far funzionare il sistema che si sta controllando. Questi sistemi SCADA sono in grado di monitorare e controllare sistemi industriali e grandi infrastrutture anche sparse su un territorio molto vasto.

Ispirati da quanto descritto finora, si è deciso di costruire questa piattaforma integrata che mantiene alcuni dei vantaggi presenti nei sistemi SCADA.

In particolare la piattaforma è costituita da un DataBase, una serie di moduli che svolgono funzioni quali attuazione, acquisizione dati e gestione degli allarmi ed una web interface che fornisce degli specifici servizi agli utenti, quali consultazione dei valori acquisiti dai sensori, visualizzazione di serie storiche, grafici e istogrammi.

La piattaforma è stata realizzata per acquisire ed elaborare una gamma di tipologie di sensori eterogenee tra loro, per cui il DataBase e le tabelle che lo costituiscono sono stati realizzati secondo una struttura generalizzata, in modo tale da rendere tale piattaforma estendibile a futuri upgrade, a seguito ad esempio dell'interfacciamento con nuove tipologie di sensori. In questo momento il sistema è stato progettato per monitorare dati dedicati all'*illuminazione pubblica* (modulo Lighting), al *consumo degli edifici* (modulo Building) e dati dedicati all'*infomobilità* (modulo Mobility).

L'eterogeneità dei dati monitorati ha suggerito l'idea di aggiungere al sistema delle funzionalità aggiuntive, che i sistemi SCADA non hanno, ovvero la capacità di operare la *data fusion* (fusione di diverse tipologie di dati sensoriali) al fine di attuare un *situation assessment,* un processo che consente di estrarre informazioni ad alto livello sullo stato del sistema aggregando in maniera appropriata dati di più basso livello.

Al fine di mantenere la piattaforma robusta ai malfunzionamenti, i moduli di acquisizione e attuazione sono stati progettati in modo tale che possano svolgere le proprie funzionalità in maniera indipendente dagli altri moduli, per cui svolgono delle funzioni dedicate e molto specifiche. Il controllo degli allarmi è gestito da un modulo dedicato, che

ciclicamente verifica se i valori dei sensori monitorati sono entro i valori di soglia specificati.

Il progetto, come mostrato in seguito, a livello implementativo è suddiviso in due sezioni: una prettamente dedicata alla comunicazione dei dati ed al controllo degli allarmi, l'altra dedicata prettamente all'interfacciamento web ed alla fornitura di servizi web tramite apposite **servlet**. A livello logico è suddiviso in quattro strati: strato di presentazione, strato di business intelligence, strato di **datawarehouse** e strato di interfacciamento con i sensori.

ARCHITETTURA GENERALE

La suddivisione a livello logico è strutturata nella maniera seguente, illustrata anche nella **Figura 1:**

- Presentation Layer
- Application Layer
- Data Layer
- Sensor-Actuator Layer

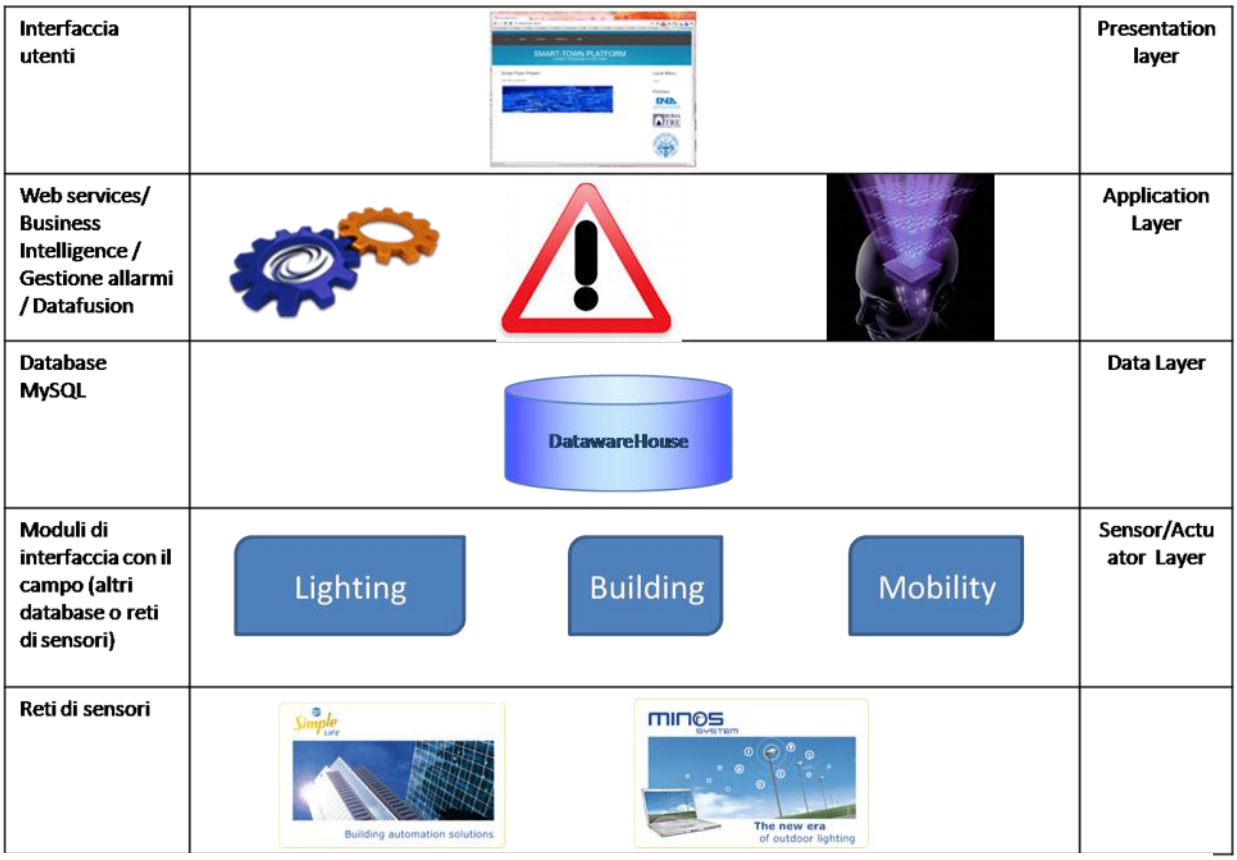


Figura 1. Visualizzazione dei diversi strati che compongono la piattaforma Smart Town

Il **presentation layer** è costituito da pagine web statiche e dinamiche (html e jsp). In particolare tale strato offre la possibilità agli utenti di interrogare il DataBase tramite il proprio web browser al fine di ricavare informazioni riguardanti i dati acquisiti dai sensori.

Attualmente è possibile visualizzare in forma tabellare tutti i sensori monitorati dalla piattaforma ed i relativi valori "statici" (es. ubicazione fisica, valori di soglia, descrizione), visualizzare graficamente l'andamento storico di uno specifico sensore o scaricare i valori storici in formato testo o xml.

L'**application layer** contiene le funzionalità dedicate alla business intelligence del sistema Smart-Town, ovvero quell'insieme di processi dedicati alla raccolta e l'analisi delle informazioni. In tale strato risiede quindi l'attività di data fusion orientata al situation assessment, di controllo degli allarmi, di creazione e l'aggiornamento di un interfaccia operatore per la visualizzazione delle criticità del sistema, di creazione e l'aggiornamento di un file di log contente tutti gli eventi rilevanti avvenuti durante il processo e di espletamento di servizi web tramite servlet.

Il **Data layer** è lo strato relativo al datawarehouse, ovvero lo storage dei dati. Per la gestione di tale DB è stato utilizzato il DBMS (DataBase Management System) MySql. Per garantire una sufficiente generalizzazione è stato creato un numero esiguo di tabelle nel DataBase *SmartTownDB*:

- SCADA
- Historian
- Alarm
- ClassAlarm
- DbAddress
- Measurement

Il **sensor layer** è lo strato contenente i moduli di interfacciamento con i sensori. Ogni modulo si interfaccia con una specifica categoria di sensori e svolge la funzionalità di ricezione o di attuazione. I moduli non si interfacciano i maniera diretta con i sensori ma con dei DataBases dedicati a loro volta all'acquisizione/attuazione dei dati dai/sui sensori. Tali DataBases sono basati sulla tecnologia Minos e Simple Life di Umpi.

Si prevede in un prossimo futuro di poter aggiungere altri DataBases o collegamenti ad altri DataBases per la ricezione di dati provenienti da altri sistemi presenti all'interno della smart city.

La suddivisione a livello implementativo è costituita da due pacchetti di lavoro principali:

1. DataCommunication
2. ServerSmartTown

Il pacchetto ***DataCommunication*** è una *Java Application* contenente delle classi dedicate alla alle funzionalità descritte nell'application layer (DataBaseManager, LogsManager, ScadaInterface) e dei Thread che svolgono le attività descritte nel sensor Layer ed in parte nell'application layer (LightingDBreader, BuildingDBreader, MobilityDBreader, AlarmsManager).

Il pacchetto ***ServerSmartTown*** è una *Java Web Application* contenente alcune classi dedicate alle funzionalità descritte nell'application layer (DataBaseManager, LogsManager) e delle servlet dedicate all'espletamento delle funzionalità descritte nel presentation layer (TableServlet, GraphGenerationServlet).

I due pacchetti benché logicamente e strutturalmente differenti hanno come punto di contatto il DataBase che abbiamo creato e vanno a scrivere sul medesimo file di log testuale. Per questo è evidente come alcune funzionalità appaiono in entrambi i pacchetti di lavoro.

PRESENTAZIONE DELLA PIATTAFORMA SMART TOWN

LA PIATTAFORMA SMART TOWN

La piattaforma tecnologica SMART TOWN realizzata per **Enea** in collaborazione con l'Università degli Studi di Roma Tre, è ispirata alla struttura canonica dei Sistemi SCADA. E' suddivisa in vari livelli, per favorirne la modularità e la scalabilità della architettura.

I tre livelli applicativi del Sistema Scada di ENEA

Poiché l'applicazione è 3-Tier possiamo suddividere questi tre livelli nei seguenti:

- **Livello applicativo:** GUI utente realizzata per mezzo di una **Web Application**, che offre la possibilità agli utenti di interrogare la Base di Dati per mezzo di pagine web dinamiche, rilevare i sensori disponibili o i dati di un determinato edificio.
- **Livello di Data Storage ed Elaborazione dei Dati:** è dedicato alla conservazione permanente dei Dati e alla loro eventuale elaborazione: parliamo cioè di Viste, Stored Procedure, etc. La Base di Dati al momento è "light" su piattaforma SUN-ORACLE con impiego di motore MySQL.
- **Livello di Trasmissione Dati agli Attuatori ed alla Ricezione dei Dati dai Sensori:** è quel livello "fisico" che si occupa della trasmissione dei Dati ed è gestito da Telecom Italia.Ovviamente poiché ci occorre realizzare un Sistema Informativo che emuli con il software l'hardware, è stato realizzato il package **"DataCommunication"** proprio per sopperire alla indisponibilità hardware, che sarebbe assai costosa in termini di risorse.
    - Allarmi
    - Gestione del DataBase emulativo dei Sensori e degli Allarmi
    - Gestione dei Logs di Sistema

Un secondo livello di organizzazione del codice, riguarda la lettura dei Dati provenienti dai Sensori:

- BuildingDBReader

- LightingDBReader
- MobilityDBReader

Ovviamente nella classificazione delle classi Java si riflette anche l'impiego degli Oggetti usati per la lettura dei dati (tipicamente un **reader** che accede ad una Base di Dati).
Per esempio il sistema poiché è capace di monitorare i consumi energetici degli Edifici, raccoglie i dati relativi agli apparati di illuminazione.

DataCommunication Class

Questa è la classe principale, che **istanzia la terna LMB (Lighting, Mobility, Buildings) .**
La classe istanzia anche un "Gestore degli Allarmi" (AlarmsManager) e una Interfaccia SCADA (ScadaInterface) la quale serve a gestire la gestione degli Allarmi.
La classe avvia poi i threads necessari per la lettura della terna LMB.

Gestione dei Logs di Sistema

Il pacchetto di **"DataCommunication"** ha anche il compito di gestire i LOGS di Sistema, ovvero di tenere traccia di tutto quello che accade nel corso del tempo, quando il Sistema nel suo complesso è in funzione.
Questa gestione viene implementata creando dei semplici file di testo.

- **LogsManager:** Scrive su di un file di testo ogni "evento" che si verifica entro **SmartTown,** ad esempio tiene traccia delle interrogazioni afferenti alla Base di Dati (ovvero le query) oppure le nuove acquisizioni dati sulla **terna LMB (Lighting, Mobility, Buildings).**
- **LogsMessagesListener:** In questo caso, questo componente è un "Listener" ovvero un vero e proprio oggetto che si preoccupa di restare in ascolto delle richieste inoltrate alle **sockets**. Risponde producendo uno **"Stream"** in output.

La Classe Utility

In genere durante lo sviluppo del sistema **SmartTown,** esiste la necessità di scrivere una classe di utilità che raccoglie varie funzionalità applicative, ma può anche nascere l'esigenza inversa, ovvero di raccogliere alcune "costanti", ovvero alcune variabili statiche, alcuni metodi che non sono associati ad un compito specifico. Qui in genere vengono raccolti in maniera centralizzata queste necessità per poi essere richiamate snellendo la stesura e rilettura del codice sorgente.
Nella classe **Utility**, trovano posto per esempio le impostazioni degli **Allarmi**, le proprietà di **Data Path** necessarie per esempio alla classe di **"DataCommunication"** per la emulazione software dei **Sensori**, quindi la classe raccoglie un po' di "triggers" ovvero qui intesi come "settaggi" reimpostati come si trattasse di "leve" di un equalizzatore(mixer).

Reader sui Database

La emulazione software della **terna LMB** necessita di una "interfaccia" con una Base di Dati sottostante che emula la acquisizione in **Real Time** dei dati provenienti dalla Rete distribuita di Sensori e Rilevatori.
Queste classi Java che gestiscono l'interfaccia alla Base di Dati emulante, consentono anche di leggere come questa emulazione sia stata realizzata, e quale approccio di sviluppo sia stato seguito.
Per comprendere queste classi occorre appoggiarsi al Livello secondo del modello 3-Tier, in particolare al livello detto anche di "Datawarehouse" che emula i Sensori, le Misurazioni, gli Allarmi attraverso una Base di Dati.

Quindi tutto ciò che viene mostrato all'utente sul livello superiore, quello di "front-end" ovvero la GUI Web di **SmartTown** si appoggia su questo livello sottostante, che si preoccupa di svolgere le interrogazioni alla Base di Dati e di restituire a Video i risultati di

queste interrogazioni, in maniera del tutto automatica.
Dobbiamo tenere presente la gestione di alcuni oggetti che richiedono delle necessarie risorse al sistema e che sono:

- Connessione al Database
- Operazioni da eseguire sul Database (per esempio SELECT, INSERT, DELETE, UPDATE etc.)
- Esecuzione dei comandi (detti anche “operazioni”)
- Gestione della Lettura/Scrittura in ingresso/uscita dei Dati del Database
- Gestione della Disconnessione dal Database
- Gestione delle Eccezioni
- Gestione dei Thread
- Chiamate a varie altre operazioni accessorie come per esempio “Gestione dei Log”.
  Le interrogazioni alla Base di Dati possono essere “portate fuori” per mezzo di strutture dati anche assai complesse oppure a mezzo di “oggetti” standard, come per esempio un **“RecordSet”.**

ELABORAZIONE DEI DATI

Il Processo di Elaborazione

Il processo di elaborazione dei dati del sistema **Smart-Town** è svolto dal pacchetto di lavoro DataCommunication. In particolare ogni modulo di acquisizione dati (lighting, building e mobility) legge periodicamente nei DataBases che si interfacciano in maniera diretta con i sensori. Se tali moduli rilevano dei nuovi dati, vengono inseriti nel DataBase *SmartTownDB*. Viceversa ogni modulo di scrittura dati (lighting, building e mobility) verifica periodicamente la presenza di valori in scrittura su *SmartTownDB* e successivamente, qualora rilevasse la presenza di valori di attuazione, avvia la procedura di attuazione sotto forma di file XML da inviare ai DataBases che si interfacciano direttamente con i sensori. Parallelamente a tale attività di acquisizione attuazione svolta dagli appositi moduli, il modulo di controllo degli allarmi quando rileva un aggiornamento sulle tabelle di *SmartTownDB* verifica se tali valori rientrano nei parametri specificati (tali parametri sono indicati nella tabella *SCADA*, che deve essere compilata prima dell'avvio del processo). Se i valori acquisiti non rispettano i parametri di soglia, viene segnalata l'anomalia nell'interfaccia operatore, come mostrato nella **Figura 2.**

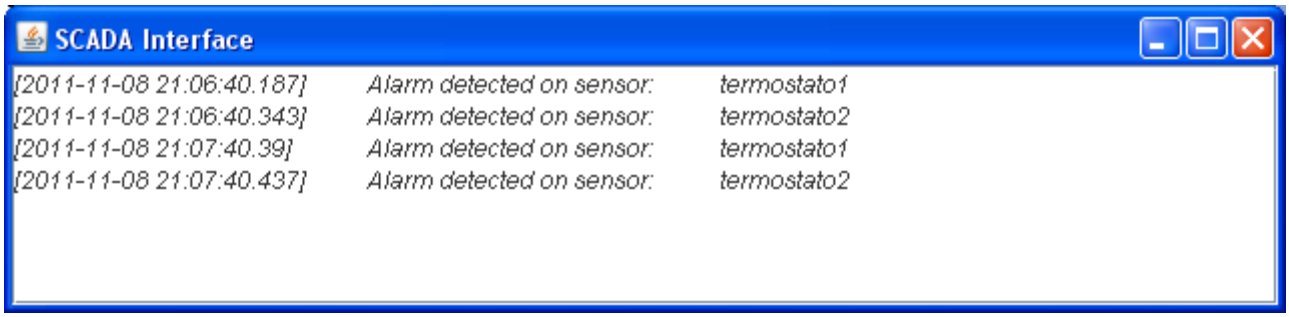


Figura 2. Visualizzazione di allarmi relativi a diversi sensori per "superamento" di soglia

operazioni classiche come INSERT, SELECT, UPDATE, CANCEL e DELETE operando sulle tabelle dei DataBases che sono anche relazionate fra loro ove occorre.

Parallelamente viene gestito un sistema di Logging centralizzato su repository per mezzo di file testuali, consultabili facilmente. Il logging permette di evidenziare e di tenere traccia di qualsiasi interrogazione venga svolta sul server SmartTown:

- Visualizzazione di errori SQL
- Connessione/Disconnessione alla Base di Dati
- Queries eseguite sulla Base di Dati
- Aggiornamenti eseguiti sulla Base di Dati (UPDATE)
- Esecuzione di una "operazione generica" sulla Base di Dati (potrebbe essere composta)

Lo svolgimento di tali attività parallele è stato ottimizzato con un'opportuna gestione degli sleeping time in modo tale da non caricare di eccessivo lavoro la Macchina Server.

Il Database

Di seguito viene fornita la descrizione dettagliata del DataBase *SmartTownDB*, il cuore del data layer dell'architettura generale descritta precedentemente. Il diagramma ER (Entity-Relationship), descrive la relazione che intercorre tra le tabelle del DataBase *SmartTownDB*.

Il DataBase come abbiamo visto è strutturato nelle seguenti tabelle:

- SCADA
- Historian
- Alarm
- ClassAlarm
- DbAddress
- Measurement

La tabella ***SCADA*** contiene le informazioni "statiche" su tutti i sensori monitorati sulla piattaforma. I suoi campi, infatti, eccetto quello inerente all'ultimo valore acquisito, devono essere inseriti manualmente da un amministratore nella fase di start up del processo, in quanto devono essere immesse informazioni che devono essere note a priori, come ad esempio i valori di soglia, il modulo a cui tale sensore fa riferimento (lighting, building, mobility), la tipologia di valore (Analogico o Digitale) ecc..

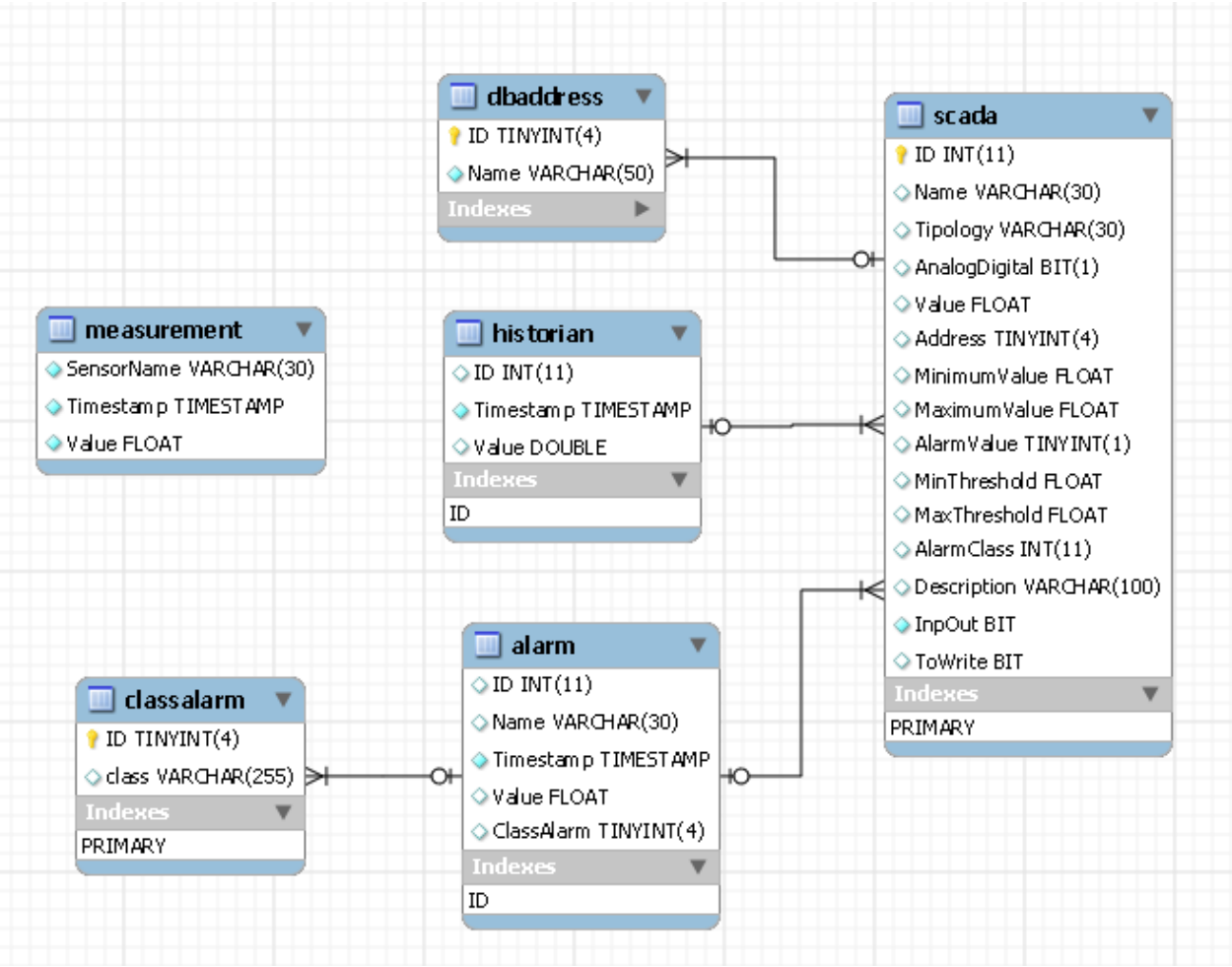


Figura 3. Diagramma relazionale del DataBase smarttowndb, presente all'intorno della piattaforma

La tabella ***Historian*** contiene lo storico di tutte le acquisizioni di ogni sensore ed è quindi la tabella contenente il numero maggiore di record.

La tabella ***Alarm*** è dedicata alla segnalazione degli allarmi. A seguito di ogni allarme rilevato viene inserito un record nella tabella con relative informazioni sull'allarme quale il timestamp del rilevamento dell'allarme, il sensore su cui si è rilevato il guasto e la classe di allarme (cui fa riferimento la tabella ***ClassAlarm*)**. La classe di allarme distingue tre tipologie di allarme:

1. Insorgenza: viene solamente segnalato il guasto, successivamente l'allarme cessa.

2. Insorgenza e riconoscimento: viene segnalato il guasto, l'allarme rimane attivato fino a quando non avviene un riconoscimento da parte di un operatore.
3. Insorgenza riconoscimento e rientro: viene segnalato il guasto, l'allarme rimane attivato fino a quando un operatore non rileva il guasto e segnala il rientro, ovvero la riparazione del guasto.

La tabella ***dbAddress*** contiene l'indirizzo dei DataBases dei moduli di interfacciamento con i sensori (lighting, building, mobility). La tabella *measurement* è una tabella di supporto utilizzata dal processo come appoggio.

Il diagramma ER, in **Figura 3,** evidenzia una relazione tra la tabella *SCADA* e le tabelle *dbAddress* e *Historian*. In particolare lo scada:ID di *measurement* ha un vincolo di integrità referenziale con l'ID della tabella *SCADA* con una relazione 1:n tra *SCADA* e *Historian*. Mentre il campo Address della tabella *SCADA* ha un vincolo di integrità referenziale sull'ID di *dbAddress*, con una relazione 1:n tra *dbAddress* e SCADA.

Dettagli Implementativi della Base di Dati

Gestione della base dati, emulazione di un sistema Scada

La base di Dati SCADA per la piattaforma Smart Town, è realizzata entro MySQL ed è costituita da varie Tabelle, fra loro relazionate.

Abbiamo diversi Database, dei quali riassumiamo nomi e funzionalità applicative:

Smart Town db

Il database in questione possiede diverse Tabelle, al momento nessuna vista e nessuna Stored Procedure. Le tabelle sono le seguenti: (tutti i nomi figurano in minuscolo)

Nome Tabella	Funzionalità	Commento
alarm	NP	Contiene informazioni sugli Allarmi dei Sensori
classalarm	NP	Definisce le possibili tipologie di Allarme.
dbaddress	NP	Definisce i Database Intermedi cui si vanno a leggere i dati dei Sensori o scrivere i valori da Attuare.
historian	NP	Contiene tutte le Misurazioni provenienti dai Sensori e immagazzinate nei relativi Database in cui si fa riferimento in fase di Configurazione.
measurement	NP	Tabella di transazione ad uso interno per la piattaforma.
scada	NP	Tabella che contiene tutte le informazioni di un Sensore.
sensor	NP	Tabella che contiene informazioni sui Sensori.

SmartTown – Sezione di insieme delle tabelle della base di Dati.

Legenda: NP = non pervenuta

Dettaglio dei campi colonna del db smartown

CAMPI COLONNA DELLE TABELLE IN MYSQL	COMMENTO EVENTUALE
Table alarm ========== id, Name, Timestamp, Value, AlarmClass ---------- id int(10) Name varchar(30) Timestamp timestamp Value float AlarmClass tinyint(4)	id: Identificativo dell'Allarme Name:nome del Sensore su cui è stato rilevato l'allarme Timestamp: Data e Ora della rilevazione dell'Allarme Value: Valore dell'Allarme AlarmClass: identificativo relazionato alla **tabella ClassAlarm.** Indica la tipologia di Allarme
Table classalarm ========== ID, class ---------- ID int(11) PK class varchar(255)	

CAMPI COLONNA DELLE TABELLE IN MYSQL	COMMENTO EVENTUALE
Table dbaddress ========== ID, Name ---------- ID tinyint(4) PK Name varchar(50)	
Table historian ========== ID, Timestamp, Value ---------- ID int(11) Timestamp timestamp Value double	ID: Id della acquisizione, vincolato preferenzialmente all'Id della **Tabella SCADA** **Timestamp:** Data ed ora della acquisizione Value: valore della acquisizione
Table measurement ========== SensorName, Timestamp, Value ---------- SensorName varchar(30) Timestamp timestamp Value double	

CAMPI COLONNA DELLE TABELLE IN MYSQL	COMMENTO EVENTUALE
Table scada ========== ID, Name, Tipology, AnalogDigital, Value, Address, MinimumValue, MaximumValue, AlarmValue, MinThreshold, MaxThreshold, AlarmClass, Description, InpOut, ToWrite ---------- ID int(11) PK Name varchar(30) Tipology varchar(30) AnalogDigital bit(1) Value float Address tinyint(4) MinimumValue float MaximumValue float AlarmValue tinyint(1) MinThreshold float MaxThreshold float AlarmClass int(11) Description varchar(100) InpOut tinyint(1) ToWrite	ID: Identificativo del Sensore, Name: Nome del Sensore, Tipology: Tipologia di raggruppamento del Sensore AnalogDigital: Valore analogico o Digitale Value: valore rilevato dal Sensore **Address:** identificativo relazionato alla **tabella DbAdress** contenente il Database in cui andare a leggere/scrivere il valore MinimumValue:valore minimo del Sensore MaximumValue:valore massimo del Sensore AlarmValue:valore di stato d'Allarme del Sensore MinThreshold: soglia minima del Sensore MaxThreshold:soglia massima del Sensore **AlarmClass:** Identificativo relazionato alla **tabella ClassAlarm**. Indica la tipologia di Allarme Description: Descrizione del Sensore InpOut: Indica se il valore è di input o di output ToWrite: booleano indicante se tale valore I/O è da inviare agli Attuatori(solo nel caso sia un Output).

```
Table sensor
============
ID, Name, Tipology,
AnalogDigital, Value,
Address, MinValue,
AlarmValue, MinThreshold,
MaxThreshold, AlarmClass,
Description, MaximumValue
------------
ID                int(11)
PK
Name
varchar(30)
Tipology
varchar(30)
AnalogDigital     bit(1)
Value             float
Address
varchar(20)
MinValue          float
AlarmValue        float
MinThreshold      float
MaxThreshold      float
AlarmClass        int(11)
Description
varchar(100)
MaximumValue      float
```

Gli altri database oltre a Smart-town

Abbiamo altri database associati alla applicazione, ciascuno contenente delle informazioni relative agli edifici (Buildingdb), uno relativo alle luci (lightingdb) ed un altro relativo alla "mobilità".

BUILDING DB

LIGHTING DB

MOBILITY DB

Gli altri Databases sono "accessori" nel senso che ci forniscono informazioni relative alla terna **LMB(Lighting, Mobility, Building).**

Dettagli Implementativi del Pacchetto Software

Viene ora analizzata nel dettaglio la struttura e l'implementazione dei pacchetti di lavoro descritti precedentemente.

Il pacchetto **DataCommunication** permette di svolgere una serie di funzionalità basilari della piattaforma.

- Comunicazione in ingresso, ovvero ricezione dei dati dai sensori
- Comunicazione in uscita, ovvero attuazione sui sensori
- Gestione degli allarmi
- Gestione della connessione ai DataBase
- Gestione dei log
- Gestione dell'interfaccia

E' stata realizzata una o più classi specifiche per ciascuna funzionalità precedentemente descritta.

DataBaseManager

Classe dedicata alla gestione della base di dati. Instaura una connessione basata sul DBMS MySQL, permette di effettuare operazioni sui DataBase e ne gestisce la disconnessione.

I membri privati di questa classe sono dedicati alla definizione dei parametri necessari per istaurare una connessione con un DataBase e sono i seguenti:

1. *Private Connection connection*
2. *Private String dbName*
3. *Private String path*
4. *Private String user*
5. *Private String password*

All'interno della classe sono presenti diversi metodi. Alcuni sono dedicati alla lettura e scrittura dei campi privati elencati precedentemente. Tutto questo consente di mantenere l'information hiding dei diversi parametri della classe.
I metodi sono i seguenti:

- *getPath()*
- *setPath(String path)*
- *getUser()*
- *setUser(String user)*
- *getPassword()*
- *setPassword(String password)*
- *getQuery()*
- *setQuery(String query)*
- *getConnection().*

È stato implementato un metodo privato in grado di fornire informazioni riguardanti gli errori che si verificano all'interno del DataBase SQL. Il metodo ha la seguente struttura *displaySQLErrors(SQLException e)*.
Esistono anche metodi pubblici all'interno della classe *DataBaseManager*:

- ***loadDriver()*:** tale metodo serve a gestire il Driver di connessione alla base di dati tramite il connettore jdbc; può anche essere gestito tramite un log testuale; e contiene anche la cattura delle eccezioni del tipo *SQLException*.
- ***connectToDB()*:** il metodo serve a gestire la connessione al DataBase, gestendo il driver di connessione alla base di dati; gestisce la stringa necessaria per effettuare la connessione al DataBase o ai DataBases in maniera parametrica; contiene la cattura delle eccezioni del tipo *SQLException*; tale metodo annota anche la connessione al DataBase e a quale DataBase ci si collega scrivendo sul log testuale.
- ***disconnectFromDB()*:** il metodo effettua la disconnessione al DataBase; come il metodo precedente, prevede la cattura di eccezioni e scrive sul logging testuale annotando la disconnessione al DataBase e il nome di tale DataBase.

Accanto a questi metodi, esistono i metodi CRUD (*Create*, *Read*, *Update* e *Delete*).
Tali metodi consentono di eseguire le classiche operazioni su DataBase SQL.
In particolare tali funzionalità sono suddivise in tre categorie, tipiche dei progetti JAVA:

1. ***executeQuery*:** consente di effettuare esclusivamente operazioni di SELECT, ed è l'unica funzionalità che è possibile, almeno al momento, eseguire dall'esterno attraverso le pagine web. In tale maniera non è possibile effettuare esternamente aggiornamenti al DataBase quali rimozione/modifica di record o tabelle; in un futuro prossimo stiamo pensando di aggiungere dall'esterno il comando di modifica di alcuni record che possono comunicare con specifici attuatori;
2. ***executeUpdate*:** consente di effettuare qualsiasi tipo di aggiornamento sul DataBase: consente la modifica di permessi, inserimento/modifica/cancellazione di campi di tabelle, di intere tabelle o di record di tabelle e l'aggiunta di vincoli referenziali;
3. ***executeOperation*:** questo metodo contente di effettuare qualsiasi tipo di operazione, sia di tipo SELECT sia di tipo UPDATE; si usa quando non si sa a priori che tipo di modifica viene effettuata sui DataBases.

AlarmsManager

La classe è dedicata alla gestione degli allarmi. La classe viene definita come un *Thread* che ciclicamente verifica l'ultimo valore di ciascun sensore presente nella tabella *SCADA* e lo confronta con i valori di soglia. In caso il valore corrente assuma valori al di fuori della soglia segnala l'anomalia tramite un'apposita interfaccia, descritta in seguito. Inoltre tale allarme viene anche segnalato tramite il file di log.

I campi della classe *AlarmsManager* sono dedicati alla definizione dei parametri necessari per instaurare la connessione con il server dove risiede il DataBase della piattaforma, chiamato *SmartTownDB*, la definizione di un *dbManager* appartenente alla classe ***DataBaseManager*** descritta precedentemente e un'interfaccia su cui verranno mostrati all'operatore i messaggi di allarme. Sono anche definiti altri parametri peculiari come idle time, flag, e path del file contenente i parametri di configurazione iniziale. I parametri sono quindi i seguenti:

1. *Private String serverDBName;*

2. *Private String serverPath;*
3. *Private String serverUser;*
4. *Private String serverPassword;*
5. *Private int idleTime;*
6. *Private DataBaseManager serverDBManager;*
7. *Private ScadaInterface scadaInterface;*
8. *Private boolean flag;*
9. *Private String propertiesPath.*

Tale classe definisce un metodo privato chiamato *void init()*. Tale metodo serve a inizializzare i valori di alcuni campi membro della classe stessa. Tali valori vengono letti da un file che elenca le diverse proprietà, proprio dei programmi java. In questo modo si può evitare la ricompilazione del codice nel caso in cui si vogliano cambiare i valori di questi parametri. Anche questo metodo fornisce modalità di cattura di eccezioni, in questo caso l'eccezione è di tipo *IOException*, ovvero legata alla scrittura o lettura su files. La stringa che definisce la connessione al server viene ottenuta leggendo il file delle proprietà. Una volta creata la stringa questa viene data al gestore di DataBase, chiamato *serverDBManager* che è un'istanza, ossia un oggetto, della classe *DataBaseManager*.

Accanto al metodo privato, esistono diversi metodi pubblici:

- ***Run()*:** il metodo avvia di fatto il thread chiamato *AlarmsManager*; il thread serve a verificare la correttezza nominale del valore ottenuto dal sensore; per correttezza nominale si intende la verifica che il valore sia compreso in un range di minimo e di massimo; in caso di superamento delle soglie, viene generato un messaggio di errore contente anche l'istante temporale del rilevamento dell'anomalia; l'allarme viene quindi mostrato nella finestra di allarme chiamata *ScadaInterface* e quindi viene scritto un record nella tabella *Alarms* con il relativo sensore coinvolto nell'anomalia; per ultimo viene loggato sul file testuale;
- *setInterface(ScadaInterface scadaInterface)*: metodo che imposta il riferimento all'opportuna istanza della classe di tipo *ScadaInterface*;

- setFlag(): metodo che consente di settare un flag a 1 che indica l'avvio del thread *AlarmsManager*;
- *unsetFlag()*: metodo che consente di settare il medesimo flag a 0, indicando il termine del thread *AlarmsManager*.

Logs Manager

La classe ha l'obiettivo di scrivere su di un file di testo ogni "evento" che si verifica sul server *SmartTown* ed in particolare di segnalare connessioni e disconnessioni dal DataBase, operazioni CRUD, rilevazione di allarmi sui sensori, etc. La maggior parte dei metodi sono "statici" e quindi possono accedere unicamente a attributi e metodi statici. In questo modo è possibile accedere ai metodi senza creare un'istanza della classe stessa.

Le variabili della classe sono le seguenti:

- *Private static Date today*
- *Private static Date previousDay*
- *Private static String propertiesPath*

Esiste un solo metodo pubblico della classe che è pubblico, chiamato *logEvent(String event)*. Questo metodo prende come argomento una stringa. Legge dal file delle proprietà il path del file su cui deve scrivere l'evento accaduto. Successivamente crea un oggetto di tipo Date per avere data e ora dell'istante in cui l'evento viene loggato. La scrittura è delegata ad un oggetto della classe *PrintWriter* che prevede anche all'eventuale creazione del file. Alla fine si prevede la chiusura del flusso in scrittura. Tale metodo prevede anche la cattura di eventuali eccezioni di tipo *FileNotFoundException* or *IOException*.

Utility Class

La classe utility contiene alcune variabili del processo non associate ad una specifica classe ma comuni a tutte le classi della piattaforma. Non contiene metodi ma solamente delle variabili membro statiche quali trigger di allarme o il path del file di configurazione delle proprietà.

La classe *main* del pacchetto di lavoro è *DataCommunication*, la quale avvia il processo creando un'istanza delle classi descritte, come mostrato in **Figura 4.**

Le classi reader sono descritte successivamente nei relativi moduli.

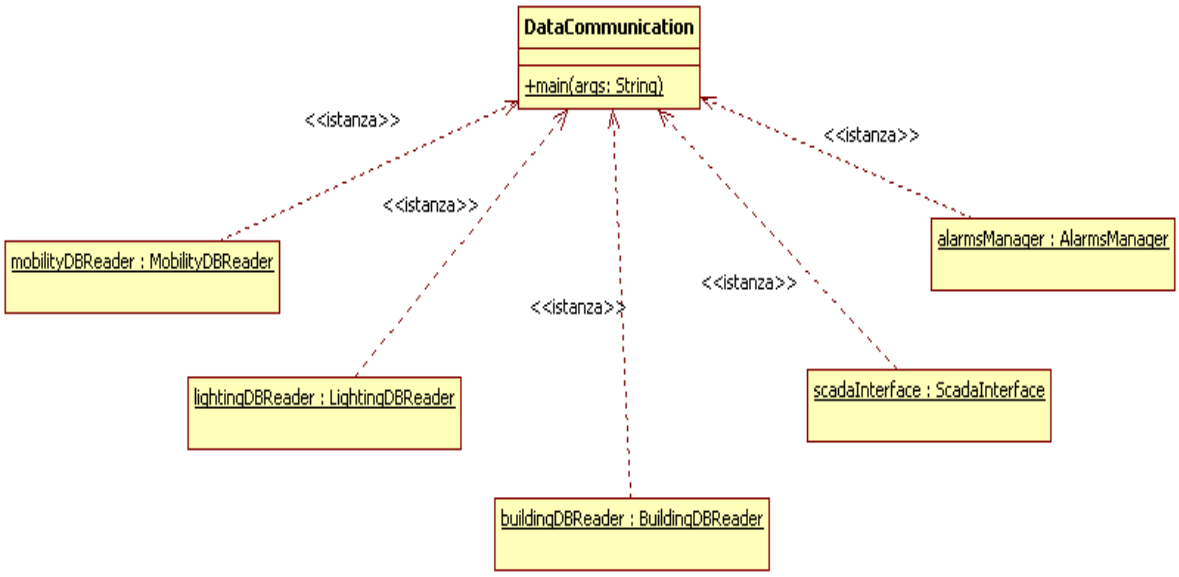


Figura 4. Rappresentazione della classe main della piattaforma

Il pacchetto **ServerSmartTown** è la Web Application che consente l'interfacciamento tra la piattaforma ed il Web. Le funzionalità che il pacchetto prevede sono:

1. Visualizzazione dei dati presenti nel DataBase *SmartTownDB* attraverso un web browser sotto forma di tabelle o grafici
2. Possibilità di scaricare i dati richiesti tramite web browser su un file di testo o in formato XML
3. Possibilità di effettuare attuazione sui sensori attraverso il Web Browser (ancora da implementare)

Tali funzionalità sono consentite dalle Servlet, ovvero delle classi Java specifiche eseguibili tramite un Web Server, solitamente a seguito di richieste di tipo http.
In particolare il server SmartTown è costituito da due servlet:

1. *GraphGenServlet*
2. *TableServlet*

GraphGenServlet

Questa servlet mostra graficamente il risultato di una richiesta di dati da parte di un client tramite web browser. In particolare tale servlet instaura una connessione con il DataBase *SmartTownDB* ed esegue una query specifica in base alla richiesta del client. Una volta processata la richiesta http da parte del client ed immessi i parametri la servlet interroga il DataBase e mostra il grafico risultante attraverso una pagina HTML dinamica.

In alternativa la servlet anziché mostrare il risultato attraverso una pagina web offre la possibilità di scaricare i dati in formato testo od in formato XML.

TableServlet

Questa servlet mostra il risultato di una richiesta di dati da parte di un client tramite web browser sotto forma di tabella attraverso una pagina HTML creata dinamicamente. Anche questa servlet offre la possibilità di scaricare i dati in formato testo o XML

Diagrammi UML
delle Classi del Package DataCommunication

DataBaseManager

-connection: Connection
-dbName: String
-path: String
-user: String
-password: String
-query: String

<<create>>+DataBaseManager(dbName: String, path: String, user: String, password: String)
+getPath(): String
+getUser(): String
+getPassword(): String
+getQuery(): String
+getConnection(): Connection
+setPath(path: String)
+setUser(user: String)
+setPassword(password: String)
+setQuery(query: String)
-displaySQLErrors(e: SQLException)
+loadDriver()
+connectToDB()
+disconnectFromDB()
+executeQuery(query: String): ResultSet
+executeUpdate(update: String): int
+executeOperation(operation: String): ResultSet

Utility

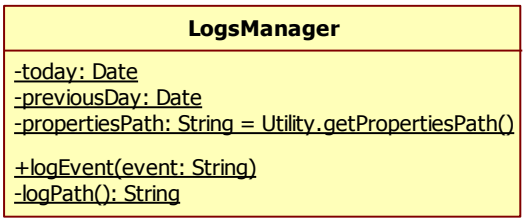
-alarmTrigger: boolean = false
-writingTrigger: boolean = false
-propertiesPath: String = "C:\\Users\\Fabio\\Documents\\NetBeansProjects\\DataCommunication\\datacommunication.properties"

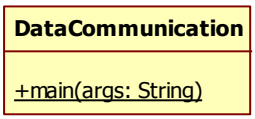
+getPropertiesPath(): String
+setPropertiesPath(propertiesPath: String)
+setAlarm()
+unsetAlarm()
+readAlarm(): boolean
+setWritingTrigger()
+unsetWritingTrigger()
+readWritingTrigger(): boolean

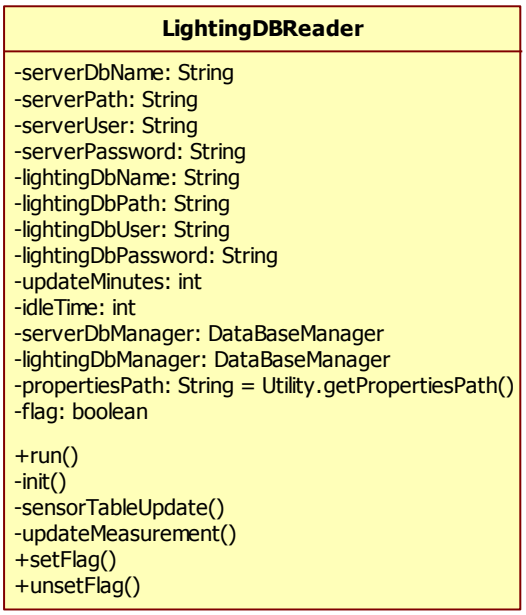
ScadaInterface

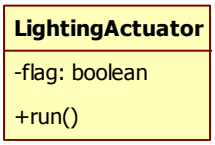
-main: Container = this.getContentPane()
-textArea: JTextArea = new JTextArea()
-scrolledtextArea: JScrollPane = new JScrollPane(textArea)

<<create>>+ScadaInterface()
+appendText(text: String)


LogsManager
-today: Date
-previousDay: Date
-propertiesPath: String = Utility.getPropertiesPath()
+logEvent(event: String)
-logPath(): String


DataCommunication
+main(args: String)


LightingDBReader
-serverDbName: String
-serverPath: String
-serverUser: String
-serverPassword: String
-lightingDbName: String
-lightingDbPath: String
-lightingDbUser: String
-lightingDbPassword: String
-updateMinutes: int
-idleTime: int
-serverDbManager: DataBaseManager
-lightingDbManager: DataBaseManager
-propertiesPath: String = Utility.getPropertiesPath()
-flag: boolean
+run()
-init()
-sensorTableUpdate()
-updateMeasurement()
+setFlag()
+unsetFlag()


LightingActuator
-flag: boolean
+run()

BuildingDBReader

-serverDbName: String
-serverPath: String
-serverUser: String
-serverPassword: String
-buildingDbName: String
-buildingDbPath: String
-buildingDbUser: String
-buildingDbPassword: String
-updateMinutes: int
-idleTime: int
-serverDbManager: DataBaseManager
-buildingDbManager: DataBaseManager
-propertiesPath: String = Utility.getPropertiesPath()
-flag: boolean

+run()
-init()
-sensorTableUpdate()
-updateMeasurement()
+setFlag()
+unsetFlag()

AlarmsManager

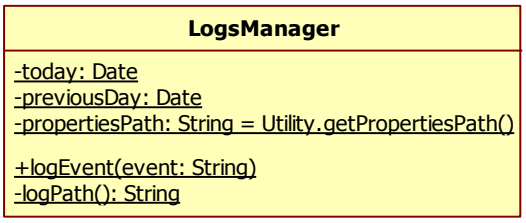
-serverDbName: String
-serverPath: String
-serverUser: String
-serverPassword: String
-idleTime: int
-serverDbManager: DataBaseManager
-scadaInterface: ScadaInterface
-flag: boolean
-propertiesPath: String = Utility.getPropertiesPath()

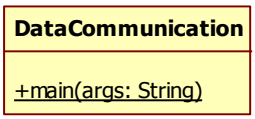
+run()
-init()
+setInterface(scadaInterface: ScadaInterface)
+setFlag()
+unsetFlag()

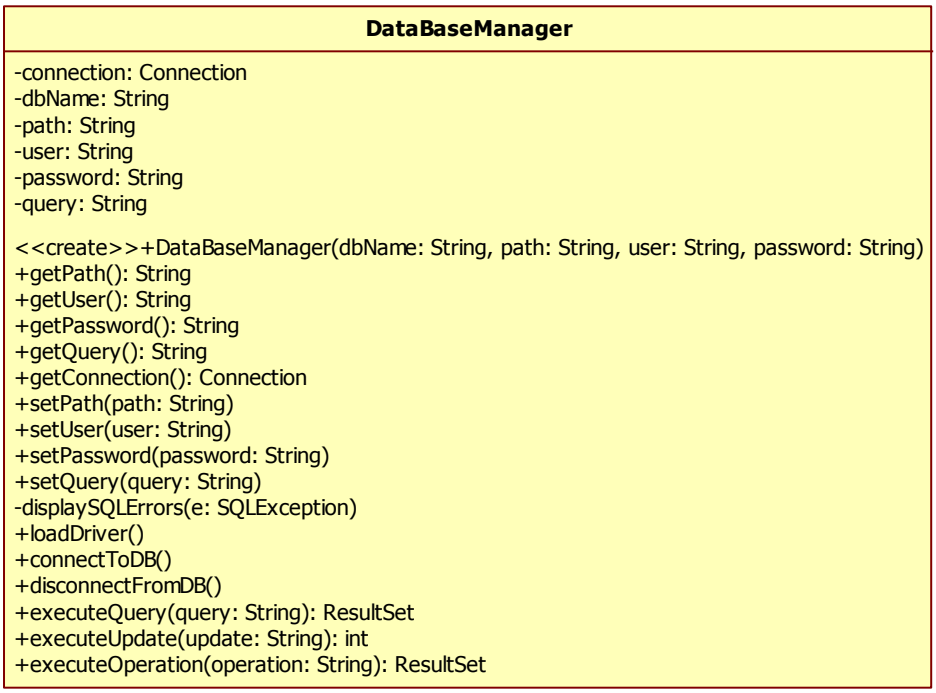
ScadaInterface

-main: Container = this.getContentPane()
-textArea: JTextArea = new JTextArea()
-scrolledtextArea: JScrollPane = new JScrollPane(textArea)

<<create>>+ScadaInterface()
+appendText(text: String)


LogsManager
-today: Date
-previousDay: Date
-propertiesPath: String = Utility.getPropertiesPath()
+logEvent(event: String)
-logPath(): String


DataCommunication
+main(args: String)


DataBaseManager
-connection: Connection
-dbName: String
-path: String
-user: String
-password: String
-query: String
<<create>>+DataBaseManager(dbName: String, path: String, user: String, password: String)
+getPath(): String
+getUser(): String
+getPassword(): String
+getQuery(): String
+getConnection(): Connection
+setPath(path: String)
+setUser(user: String)
+setPassword(password: String)
+setQuery(query: String)
-displaySQLErrors(e: SQLException)
+loadDriver()
+connectToDB()
+disconnectFromDB()
+executeQuery(query: String): ResultSet
+executeUpdate(update: String): int
+executeOperation(operation: String): ResultSet

DIAGNOSTICA E DATA FUSION

Abbiamo in parte parlato della diagnostica nel paragrafo precedente, relativamente al Logging della applicazione e alla gestione degli Allarmi.

La funzionalità di Data Fusion al momento non è stata implementata. Questa potrebbe prevedere l'impiego di apposite viste sul DataBase, in particolare il *BuildingDB* al fine di operare un processo di elaborazione dei dati finalizzata alla diagnostica, monitoraggio e controllo del comportamento energetico degli edifici.

Lo scopo è quindi quello di acquisire dati sui consumi e aggregarli in vari livelli di astrazione in modo tale da attuare un situation assessment, ovvero identificazione delle possibili cause e possibili soluzioni a delle problematiche emerse a seguito dell'elaborazione di dati sensoriali. E' prevista l'applicazione di una serie di metodologie per attuare il situation assessment, con lo scopo di individuare le più adatte ai nostri scopi.

La problematica del situation assessment deriva dal contesto della data fusion multi-sensoriale e riguarda essenzialmente l'automatizzazione del processo cognitivo eseguito da un supervisore. Infatti, per situation assessment si intende il processo autonomo di divenire cosciente di determinate situazioni o di determinati allarmi. Negli ultimi anni tale problema sta riscontrando sempre maggior interesse e nuovi campi di applicazione: dalla robotica alla diagnostica medica, dal riconoscimento di pattern al riconoscimento di anomalie, dall'inferenza statistica al campo del decision making.

L'analisi dei dati provenienti dai diversi sensori e, in generale, dai diversi impianti sensoristici è obbligatorio e vitale, a causa della grande mole di informazioni che giungono alla nostra piattaforma. La piattaforma può consentire sia l'analisi di dati in maniera verticale, ossia discriminando i dati in base alla sorgente sensoristica, sia in maniera orizzontale ossia scegliendo alcuni tipi di dati provenienti da sensori diversi.

La piattaforma essendo general-purpose, offre la capacità di fornire i dati in maniera grezza sia alcuni algoritmi già implementati per l'analisi di metriche e di dati ad alto livello di astrazione.

Teoria dell'Evidenza

Tra le diverse metodologie, una che è in grado di ottenere buoni risultati con diversi gradi di astrazione è la teoria dell'evidenza (evidence theory). Tale termine è stato ideato da **Shafer** che rivede la teoria di **Dempster** sulle probabilità indotte. Per questo motivo questa teoria è anche chiamata Dempster-Shafer.

Esistono diversi approcci a questa teoria: quelli di tipo probabilistico, come ad esempio Dempster nel 1967 o quello basato sui random sets, ma anche approcci di tipo assiomatico, come quello di Shafer nel 1976 o il Transferable Belief Model di Smets. In questo caso si illustrano alcune caratteristiche della teoria dell'evidenza usando l'approccio assiomatico di Shafer.

Si consideri un insieme chiamato *frame of discernment*, ossia un insieme che contiene tutte le risposte ad una definita domanda. Il frame of discernemnt è rappresentato dal simbolo $\Theta = \{\theta_i, i = 1, \dots, n\}$. Si deve inoltre considerare l'insieme potenza di Θ, ossia *power set* 2^{Θ}, che contiene tutti i possibili sottoinsiemi generati a partire dall'unione degli elementi di Θ.

Si consideri l'esistenza di una misura numerica rappresentante il grado di *belief* ossia di credibilità su ciascuna ipotesi $H \subseteq \Theta$. Tale funzione deve rispettare i seguenti vincoli:

$$Bel\ \emptyset = 0$$
$$Bel\ \Theta = 1$$

$$Bel\ H \geq \sum_{i \in I} -1^{I+1} Bel\ H_i : \emptyset \neq I \subseteq 1, \dots, n \quad , \forall n \geq 1, \forall H, H_i \subseteq \Theta : H_i \subseteq H$$

La funzione $Bel: 2^{\Theta} \rightarrow [0,1]$ che rispetta tali vincoli è definita funzione di belief e rappresenta quanto è possibile provare o dedurre che l'ipotesi considerata a partire dalle informazioni a nostra disposizione.

Dalla funzione di belief è possibile definire un'altra funzione chiamata *Basic Belief Assignment* (BPA) che viene definita come $m: 2^{\Theta} \rightarrow [0,1]$ che dove rispettare il seguente teorema.

TEOREMA. *Se Bel è una funzione belief, allora la funzione*

$$m\ H\ = \sum\{\ -1\ ^{H-B}\ Bel\ B\ : B \subseteq H\}$$

è una funzione di insiemi $m: 2^{\Theta} \rightarrow [0,1]$ *che soddisfa le seguenti assunzioni:*

$$m\ \emptyset\ = 0$$
$$m\ H\ \geq 0, \forall H \subseteq \Theta$$
$$\sum m\ H\ : H \subseteq \Theta\ = 1$$

è tale che $Bel\ H\ = \sum m\ B\ : B \subseteq H\ , \forall H \subseteq \Theta.$

La funzione *m* è interpretata come una distribuzione di masse di probabilità sui sotto-insiemi non vuoti di Θ, tale che $m(B)$ è intesa come "la misura di belief che corrisponde esattamente ad *H*" (Shafer, 1976). Quindi la funzione belief *Bel(H)* è il belief totale collegato ad *H* che è la somma dei belief collegati ai sottoinsiemi propri di *H*.
Si definiscono *insiemi focali* (focal sets) i sottoinsiemi di *H* che hanno $m\ B\ \geq 0$.

Accanto a queste due funzione ne esista una terza, detta funzione di *plausibilità* (o plausibility) direttamente legata alla funzione belief. La funzione di plausibilità indica quanto l'ipotesi considerata può essere probabile partendo dalle informazioni a nostra disposizione.

Partendo dall'insieme *H* che ha valore *Bel(H)* come belief, si deduce che l'incertezza in H è legata all'insieme H^c, ossia il suo complementare, per cui $dou\ H\ = Bel\ (H^c)$. In generale, minore è il dubbio, maggiore è il valore della funzione di plausibilità $Pl\ H\ = 1 - dou\ H\ = 1 - Bel\ H^c$.

Un concetto molto importante è quello concernente le regole di combinazione. Infatti, una volta associate le diverse masse *m* alle diverse ipotesi è fondamentale la maniera in cui queste regole devono essere combinate.

Per come è stata definita la funzione belief e basic probability assignment, la *regola di combinazione di Dempster* risulta facilmente ottenibile, considerando r funzioni di belief indipendenti sugli *n* elementi di Θ:

$$m\ H\ = c \sum m_1\ B_1\ \ldots m_r\ B_n\ : B_1 \cap \ldots \cap B_n = H\ , \forall H \neq \emptyset$$

dove

$$c = 1 - \sum\{m_1\ B_1\ \ldots m_r\ B_n\ : B_1 \cap \ldots \cap B_n = \emptyset\}$$

La normalizzazione che viene eseguita nel calcolo del coefficiente *c* consente una ri-distribuzione dell'incertezza sugli insiemi appartenenti al power set. L'incertezza è descritta dal coefficiente numerico $\sum\{m_1\ B_1\ \ldots m_r\ B_n\ : B_1 \cap \ldots \cap B_n = \emptyset\}$.

Esistono altre regole di combinazione che consentono di ottenere risultati diversi, in base alla situazione utilizzata.

Logica Fuzzy

Il termine "logica fuzzy" è stato coniato da **Lofti Zadeh** nel 1965, basandosi sullo sviluppo della teoria degli insiemi fuzzy. Un sottoinsieme fuzzy A di un insieme (crisp) X è caratterizzato dall'assegnamento a ogni elemento $x \in X$ del *grado di appartenenza* (degree of membership) dell'elemento x a A. Ad esempio se X è un gruppo di persone, A è l'insieme fuzzy di persone anziane di X. Ora se X è un insieme di proposizioni allora ad ognuno dei suoi elementi si dovrebbe assegnare il loro grado di verità, che potrebbe essere "assolutamente vero", "assolutamente falso", o un grado intermedio tra i due. Ciò risulta evidente se si considerano informazioni imprecise e vaghe come "questa persona è anziana". Ovviamente come è stato fatto per gli insiemi fuzzy, esistono anche degli operatori che consentono la combinazione di proposizioni, tramite ad esempio la congiunzione (conjunction), la disgiunzione (disjunction) o la negazione (negation).

Esistono due direzioni maggiori per la logica fuzzy. La logica fuzzy *in senso lato*, quella più vecchia e più nota che è stata utilizzata per il controllo di tipo fuzzy, e nell'analisi dell'incertezza del linguaggio naturale.

Per ulteriori informazioni si rimanda alle seguenti monografie: Novak nel 1989, **Zimmermann** del 1991, **Klir-Yuan** 1996, e l'ultima di **Nguyen** del 1999.

La logica fuzzy *in senso stretto* è la logica simbolica con una nozione di verità nello spirito della logica classica, quindi la definizione della sintassi, della semantica, degli assiomi, della completezza, della deduzione che preserva la verità e di molti altri principi, sia per la logica proposizionale sia per la logica dei predicati. È un ramo che analizza la logica multi-valore basata sul paradigma dell'inferenza sotto incertezza. La logica fuzzy è una disciplina relativamente nuova, che può servire sia come base per la logica fuzzy in senso lato sia di indipendente interesse logico. Una monografia di base è quella di Hajek del 1998, ulteriori riferimenti sono **Turunen** 1999 e Novak et al. 2000.

L'insieme standard di *gradi di verità* (truth degrees) è l'intervallo reale $[0, 1]$ con il suo ordinamento naturale, descritto dal simbolo $\leq$, quindi *1* indica l'assoluta verità mentre *0* indica falso. In realtà ognuno può lavorare con differenti domini, finiti o infiniti, ordinati o

solo parzialmente ordinati. Le funzioni di verità dei connettivi (connectives) devono comportarsi in maniera classica sui valori estremi *0* e *1*.

È universalmente accettata la *t-norma,* o norma triangolare, come funzione verità della congiunzione. La funzione t-norma è un operatore binario sull'intervallo [0,1] e deve essere commutativa, associativa, non-decrescente e avere 1 come unità. La t-norma più famosa è l'operatore di minimo. Dualmente, la *t-conorma* è utilizzata come funzione verità dell'operatore di disgiunzione. La funzione verità della negazione e deve essere non-crescente. La funzione di negazione più utilizzata è quella chiamata *negazione di Lukasiewicz* (Lukasiewicz negation) definita come $1 - x$.

L'*implicazione* è a volte trascurato, ma è di fondamentale importanza per la logica fuzzy in senso stretto. Una possibilità semplice ma logicamente meno interessante è di definire l'implicazione da congiunzione e negazione, oppure da disgiunzione e negazione, usando la corrispondente tautologia della logica classica.

Tali implicazioni sono dette *S-implicazioni* (S-implications). Molto più interessante è la definizione di *R-implicazione* (R-implication). Una R-implicazione è definita come il residuo (residuum) di una t-norma.

Sia * la t-norma e il residuo $\rightarrow$, allora si ha che $x \rightarrow w = \max\ z \quad x * z \leq y\}$.
Tale operazione è ben definita se e solo se la t-norma è continua a sinistra (left-continuous).

Hajek nel 1998 ha definito la logica *proposizionale fuzzy di base* (basic fuzzy propositional logic- BL) delle t-norme continue, ossia una t-norma che ha anche la proprietà di continuità e quindi che piccoli cambiamenti degli argomenti portano a piccoli cambiamenti nel risultato dell'operatore di t-norma. Matematicamente la continuità è specificata nel seguente modo.

DEFINIZIONE *(Continuità). L'operatore * di t-norma è continuo se per ogni* $\epsilon > 0$ *esiste un* $\delta > 0$ *tale che per ogni* $x_1 - x_2 < \delta$ *e per ogni* $y_1 - y_2 < \delta$ *allora* $x_1 * y_1 - x_2 * y_2 < \epsilon$.

Le formule sono quindi costruite dalle variabili proposizionali usando i connettori &

(congiunzione), → (implicazione) e il valore costante *0* che indica falso. La negazione $\neg\varphi$ è definita come $\varphi \to 0$. Data una t-norma continua * e quindi il suo residuo →, ogni valutazione *e* di variabili proposizionali con gradi di verità in [0,1] si estende univocamente alla valutazione $e * (\varphi)$ di ogni formula φ usando * e → come funzioni verità per *&* e →.

Ogni formula φ è una *t-tautologia* (t-tautology) o BL-tautologia standard se $e * \varphi = 1$ per ogni valutazione *e* e ogni t-norma continua *.

Le seguenti t-tautologie sono considerate come *assiomi* della logica fuzzy di base.

$A1 - \varphi \to \psi \to \psi \to \chi \to \varphi \to \chi$

$A2 - \varphi \,\&\, \psi \to \varphi$

$A3 - \varphi\&\psi \to (\psi\&\varphi)$

$A4 - (\varphi\&(\varphi\&\psi)) \to (\psi\&(\psi \to \varphi))$

$A5a - \varphi \to \psi \to \chi \to (\varphi\&\psi \to \chi)$

$A5b - \varphi\&\psi \to \chi \to (\varphi \to (\psi \to \chi))$

$A6 - \varphi \to \psi \to \chi \to \psi \to \varphi \to \chi \to \chi$

$A7 - 0 \to (\varphi)$

Modus ponens è l'unica regola di deduzione che esiste in questo approccio. Il teorema della completezza standard (vedi **Cignoli** et al. 2000) afferma che una formula φ è una t-tautologia se e solo se si può dimostrare all'interno della logica proposizionale di base.

MODULI E LORO INTEGRAZIONE

Modulo per l'Illuminazione Pubblica

E' stato implementato un modulo *LightingDBReader* per la interconnessione con la Base di Dati *LightingDB*. Tale modulo acquisisce dati dai DataBases che si interfacciano con i sensori (in particolare telecamere) dedicati all'analisi ambientale e del traffico (veicolare e pedonale) orientata alla gestione dell'illuminazione pubblica urbana e li immagazzina in *SmartTownDB*.

Il modulo *LightingActuator* è in fase di testing, e svolge la funzione di lettura dei dati di attuazione su *SmartTownDB* e di attuazione di tali sui sensori creando un opportuno file XML da inviare al DataBase che si interfaccia direttamente con i sensori dedicati all'illuminazione pubblica.

Dettagli Implementativi del Modulo di Illuminazione

LightingDbReader è la classe che attua il processo di lettura dei dati dai sensori andando a leggere i dati contenuti nei DataBases che si interfacciano direttamente con i sensori inerenti il modulo dell'illuminazione pubblica. Tale classe è di tipo Thread, in quanto ciclicamente legge i dati da uno o più DataBases dedicati all'illuminazione (es. *LightingDB*) e, se rileva nuovi dati rispetto a quelli già immagazzinati su *SmartTownDB*, li inserisce nella tabella *Measurement* descritta precedentemente.

La classe possiede diversi parametri: i primi sono relativi alla locazione del DataBase server della piattaforma, ossia quello centralizzato; quindi ci sono i dati relativi al DataBases relativi all'illuminazione pubblica; ovviamente è necessario definire anche i relativi *DataBaseManager* per entrambi i DataBases.

È necessario aggiungere anche aggiungere le variabili per l'aggiornamento del sistema e il flag per l'avvio e il termine del thread. I parametri sono quindi i seguenti:

1. *Private String serverDbName;*
2. *Private String serverPath;*

3. *Private String serverUser;*
4. *Private String serverPassword;*
5. *Private String lightingDbName;*
6. *Private String lightingDbPath;*
7. *Private String lightingDbUser;*
8. *Private String lightingDbPassword;*
9. *Private DataBaseManager serverDbManager;*
10. *Private DataBaseManager lightingDbManager;*
11. *Private String propertiesPath*
12. *Private int updateMinutes;*
13. *Private int idleTime;*
14. *Private boolean flag.*

La classe possiede diversi metodi privati che vengono eseguiti solo all'interno della classe. I metodi vengono elencati di seguito, con le loro funzionalità:

1. ***init()*:** tale metodo svolge una serie di inizializzazioni sulle variabili private della classe *LightingDBReader*, leggendo dal file delle proprietà i diversi valori delle variabili;
2. ***sensorTableUpdat****e()*: questo metodo tiene aggiornata la tabella *SCADA*, inserendo per ogni sensore presente all'interno della tabella l'ultimo valore acquisito.; l'ultimo valore viene calcolato facendo il join, e quindi unendo, la tabella *SCADA* e il DataBase *Historian*; poi per ogni sensore viene prelevato e copiato il valore con timestamp più recente;
3. ***updateMeasurement()*:** metodo che svolge il trasferimento dei dati dalla tabella di appoggio *Measurement* alla tabella *Historian*; una volta eseguito il passaggio dei dati il contenuto della tabella *Measurement* viene cancellato.

L'ultimo metodo ha richiesto la definizione come syncronized (*private syncronized void updateMeasurement()*) per evitare inserimenti o cancellazioni di dati in maniera errata a causa di chiamate contemporanee dello stesso metodo.

Accanto ai metodi privati, esistono diversi metodi pubblici, che sono analoghi a quelli mostrati nella classe *AlarmsManager*:

- ***run()*:** il metodo avvia di fatto il thread; serve essenzialmente per avviare l'intero processo di acquisizione dei dati;
- ***setFlag()*:** metodo che consente di settare un flag a 1 che indica l'avvio del thread *LightingDBManager*;
- ***unsetFlag()*:** metodo che consente di settare il medesimo flag a 0, indicando il termine del thread *LightingDBManager*.

Tra i futuri aggiornamenti c'è quello di aggiungere una *LightingDBActuator* classe, per consentire la scrittura su DataBase dall'esterno in modo da poter anche mandare comandi agli attuatori presenti nel sistema reale.

Modulo per la Rete di Edifici

È stato implementato un modulo *BuildingDBReader* per la interconnessione con la base di dati *BuildingDB*. Tale modulo acquisisce dati dai DataBases che si interfacciano con i sensori dedicati all'acquisizione dei dati sui consumi energetici degli edifici e li immagazzina in *SmartTownDB*. Il modulo *BuildingActuator* è ancora in fase di implementazione. I dettagli implementativi del modulo per la rete di edifici è analogo a quello per l'illuminazione, per completezza viene fornita la descrizione.

Dettagli Implementativi del Modulo per Edifici

La classe ***BuildingDBReader*** implementa la lettura dei dati dai DataBases specifici degli apparati presenti nei diversi edifici e li copia all'interno della piattaforma Smart Town. Anche questa classe come la precedente è di tipo Thread, prevedendo al modalità di funzionamento ciclica grazie a delle variabili che rappresentano gli intervallo temporali.

Questa classe funziona ed ha la medesima struttura della precedente. Per completezza e uniformità si descrive comunque la struttura di questa classe. I parametri previsti sono quelli relative alla tabella *SCADA*, ossia al server che è centralizzato, della piattaforma Smart Town; della tabella che si interfaccia direttamente con i sensori e gli attuatori della rete di edifici; ovviamente vanno elencati i *DataBaseManager* per entrambe le tabelle e i tempi di ciclo per l'aggiornamento dei dati, ossia:

- *Private String serverDbName;*
- *Private String serverPath;*
- *Private String serverUser;*
- *Private String serverPassword;*
- *Private String buildingDbName;*
- *Private String buildingDbPath;*
- *Private String buildingDbUser;*
- *Private String buildingDbPassword;*
- *Private DataBaseManager serverDbManager;*
- *Private DataBaseManager buildingDbManager;*
- *Private String propertiesPath*
- *Private int updateMinutes;*
- *Private int idleTime;*
- *Private boolean flag.*

La classe possiede diversi metodi privati che vengono eseguiti solo all'interno della classe. Questi metodi sono uguali a quelli presenti nella classe *LightingDBReader*. I metodi vengono elencati di seguito, con le loro funzionalità:

1. ***init()*:** metodo che svolge una serie di inizializzazioni sulle variabili private della classe *BuildingDBReader*, leggendo dal file delle proprietà i diversi valori delle variabili;
2. ***sensorTableUpdate()*:** metodo che tiene aggiornata la tabella *SCADA*, inserendo per ogni sensore presente all'interno della tabella l'ultimo valore acquisito;
3. ***updateMeasurement()*:** metodo che svolge il trasferimento dei dati dalla tabella di appoggio *Measurement* alla tabella *Historian*; una volta eseguito il passaggio dei dati il contenuto della tabella *Measurement* viene cancellato.

L'ultimo metodo ha richiesto la definizione come syncronized (*private syncronized void updateMeasurement()*) per evitare inserimenti o cancellazioni di dati in maniera errata a causa di chiamate contemporanee dello stesso metodo.
Accanto ai metodi privati, esistono diversi metodi pubblici, che sono analoghi a quelli mostrati nella classe *AlarmsManager*:

- ***run()*:** il metodo avvia di fatto il thread; serve essenzialmente per avviare l'intero processo di acquisizione dei dati;
- ***setFlag()*:** metodo che consente di settare un flag a 1 che indica l'avvio del thread BuildingDBReader;
- ***unsetFlag()*:** metodo che consente di settare il medesimo flag a 0, indicando il termine del thread *BuildingDBReader*.

Tra i futuri aggiornamenti c'è quello di aggiungere una *BuildingDBActuator* classe, per consentire la scrittura su DataBase dall'esterno in modo da poter anche mandare comandi agli attuatori presenti nel sistema reale.

Modulo per la Mobilità

E' stato implementato un modulo *MobilityDBReader* per la interconnessione con la base di Dati *MobilityDB*. Tale modulo acquisisce dati da DataBase che si interfacciano direttamente con i sensori dedicati all'analisi del traffico orientata alla mobilità e li immagazzina in *SmartTownDB*. Il modulo *MobilityActuator* è ancora da implementare.

Dettagli Implementativi del Modulo di Mobilità

La classe *MobilityDBReader* implementa la lettura dei dati dai DataBase specifici degli apparati presenti nei diversi moduli mobili come ad esempio le navette della Casaccia – **ENEA** e li copia all'interno della piattaforma Smart Town. Anche questa classe come le precedenti è di tipo Thread, prevedendo al modalità di funzionamento ciclica grazie a delle variabili che rappresentano gli intervallo temporali.

Questa classe funziona ed ha la medesima struttura della precedente. Per completezza e uniformità si descrive comunque la struttura di questa classe. I parametri previsti sono quelli relative alla tabella *SCADA*, ossia al server che è centralizzato, della piattaforma Smart Town; della tabella che si interfaccia direttamente con i sensori e gli attuatori dei veicoli; ovviamente vanno elencati i *DataBaseManager* per entrambe le tabelle e i tempi di ciclo per l'aggiornamento dei dati, ossia:

- *Private String serverDbName;*
- *Private String serverPath;*
- *Private String serverUser;*
- *Private String serverPassword;*
- *private String mobilityDbName;*
- *private String mobilityDbPath;*
- *private String mobilityDbUser;*
- *private String mobilityDbPassword;*
- *Private DataBaseManager serverDbManager;*
- *Private DataBaseManager mobilityDbManager;*
- *Private String propertiesPath*
- *Private int updateMinutes;*
- *Private int idleTime;*
- *Private boolean flag.*

La classe possiede diversi metodi privati che vengono eseguiti solo all'interno della classe.

Questi metodi sono uguali a quelli presenti nelle classi *LightingDBReader* e *BuildingDBReader*.
I metodi vengono elencati di seguito, con le loro funzionalità:

1. ***init()*:** metodo che svolge una serie di inizializzazioni sulle variabili private della classe *MobilityDBReader*, leggendo dal file delle proprietà i diversi valori delle variabili;
2. ***sensorTableUpdate()*:** metodo che tiene aggiornata la tabella *SCADA*, inserendo per ogni sensore presente all'interno della tabella l'ultimo valore acquisito;
3. ***updateMeasurement()*:** metodo che svolge il trasferimento dei dati dalla tabella di appoggio *Measurement* alla tabella *Historian*; una volta eseguito il passaggio dei dati il contenuto della tabella *Measurement* viene cancellato.

L'ultimo metodo ha richiesto la definizione come syncronized (*private syncronized void updateMeasurement()*) per evitare inserimenti o cancellazioni di dati in maniera errata a causa di chiamate contemporanee dello stesso metodo.

Accanto ai metodi privati, esistono diversi metodi pubblici, che sono analoghi a quelli mostrati nella classe *AlarmsManager*:

- ***run()*:** il metodo avvia di fatto il thread; serve essenzialmente per avviare l'intero processo di acquisizione dei dati;
- ***setFlag()*:** metodo che consente di settare un flag a 1 che indica l'avvio del thread *MobilityDBReader*;
- ***unsetFlag()*:** metodo che consente di settare il medesimo flag a 0, indicando il termine del thread *MobilityDBReader*.

Come già affermato per le altre classi di tipo Thread, si intende proseguire lo sviluppo della piattaforma aggiungendo il Thread per l'attuazione di diversi comandi sugli apparecchi presenti sui diversi autoveicoli.

SPERIMENTAZIONE

E' stato possibile testare quasi la totalità delle funzionalità previste dal sistema Smart-Town. In particolare il funzionamento dei moduli di acquisizione dati, di gestione degli allarmi, l'interfaccia ed il logging sono stati testati su dati creati ad hoc in fase di simulazione.
Le funzionalità di attuazione ancora non sono state testate.

Al momento si prevedono varie fasi di sperimentazione sul sistema SmartTown, che elenchiamo brevemente nella lista seguente:

- Sviluppo di un sistema di Autenticazione Sicuro
- Sviluppo di un sistema di Gestione della Sessione Utenti
- Sviluppo di un sistema di Gestione Amministrativo
- Sviluppo di Web Services con varie tecnologie a disposizione
- Porting del DataBase su applicazioni a carattere Enterprise
- Interoperabilità fra MFC (Microsoft Foundation Class) e Java
- Sviluppo di Viste e di Stored Procedure
- Sviluppo di nuove pagine di Front-End per le Web Application
- Sviluppo di un Sistema di Analisi della Configurazione
- Porting su altri sistemi Operativi (Linux, FreeBSD)
- Interoperabilità con NET Web Services
- Altre funzionalità applicative (costruzione di pagine JSP/Servlet specifiche)
- Impiego di altre tecnologie a supporto applicativo
- Sviluppo di un portale a contenuto Georeferito (Open GIS)

Stiamo valutando anche la possibilità di rendere il sistema ridondante, in caso di guasti o problemi sul server. La ridondanza del sistema ci ha portato anche a valutare l'eventualità di istallare la piattaforma su **Cloud**. La possibilità di installazione su Cloud consentirebbe la fruibilità dei dati in maniera più semplice, e questi dati sarebbero sempre disponibili. Come svantaggio si avrebbe quello della sicurezza e della confidenzialità dei dati caricati all'interno della piattaforma, in quanto uno dei nostri scopi nel costruire questa piattaforma

è stato anche quello della sicurezza e della disponibilità dei dati delle diverse infrastrutture che si possono collegare alla piattaforma.

La nostra applicazione, come qualsiasi applicazione del campo dell'informatica, dovrà rispettare i seguenti tre criteri, riguardanti gli aspetti di sicurezza:

1. Confidenzialità delle informazioni: i dati devono essere accessibili solo a coloro autorizzati ad avere l'accesso al sistema;
2. Integrità dei dati: le informazioni devono mantenere sempre la medesima struttura completa;
3. Disponibilità dei dati: le informazioni devono essere sempre disponibili per chi accede al sistema.

CONCLUSIONI

La piattaforma Smart Town consente l'integrazione di diverse tecnologie per rendere disponibili agli scienziati, agli operatori e agli utilizzatori un gran numero di dati provenienti da una Smart Town, che potrebbe essere il centro di ricerca ENEA in Casaccia, o una qualsiasi realtà urbana.

La struttura logica e fisica del sistema è stata pensata per poter gestire grandi quantità di dati, come si può ragionevolmente pensare, possono essere ottenuti dalle tecnologie fisiche presenti in una città. Al momento i test che abbiamo effettuato, su dati creati appositamente per testare il sistema, hanno portato alla consapevolezza che il sistema è molto veloce e resistente a possibili errori o a bugs.

Il sistema è stato creato per rendere disponibili in maniera sicura e affidabile i dati prodotti. La possibilità è quella di creare routines in grado di analizzare i dati eterogenei e generare o testare le metriche ideate da scienziati e ricercatori. Non è esclusa la possibilità di generare routines in grado di generare migliori algoritmi di controllo di specifici sistemi, grazie all'utilizzo di dati generati da altri sistemi o da dati generati da algoritmi di fusione.

Sono state presentate sommariamente le funzionalità modulari deployate e le eventuali sperimentazioni che suddetta piattaforma offre in relazione alla crescente esigenza di Building Automation, che non escludono sviluppi anche commerciali in diversi settori (Efficiency Strategy, Logistica, Metering & Reporting, Audit Commerciale, Audit Tecnico). L'elevata personalizzazione dei Web Services e la forte concorrenza in questo settore, crea una sinergia che vale la pena poter approfondire.

Crediamo che, accanto alla disponibilità del servizio fornito dalla piattaforma, è necessario anche rendere il servizio sicuro rispetto a possibili errori degli operatori o a intrusioni per acquisire i dati sensibili acquisiti e generati. È necessario quindi pensare a strategie di ridondanza del servizio, tramite DataBase e procedure di backup, ma anche eventuale distribuzione della piattaforma.

Last Page Left Blank