Protezione Civile

Valeriy Litkovskyy, Alessio Spagnolo, Christian Premoli, Gabriele Bianchi
June 9, 2020

Introduzione

In questo documento discuteremo le tecniche implementative utilizzate. Per evitare di ripeterci, ometteremo le informazioni già riportate nella documentazione di UML (soprattutto nella descrizione del Class Diagram).

Fasi di lavoro

L'implementazione del software java è l'ultima parte della realizzazione del progetto. In questa fase creeremo il programma, che andrà a rispettare tutti i modelli realizzati nelle fasi precedenti. Abbiamo realizzato innanzitutto il database, utilizzando SQlite. Dopodiché abbiamo creato i package con le relative classi, al cui interno sono stati inseriti i metodi, implementandoli. Infine sono state integrate la struttura client - server con RMI e la grafica con Java Swing.

Implementazione

L'implementazione del nostro software è stata suddivisa in quattro package per delimitare le componenti dell'applicazione. Questi package sono: Source, Server, User e Common. Source, Server e User sono separati e indipendenti tra loro, infatti ognuno ha un proprio main e funzionano in modo indipendente utilizzando classi presenti in Common. Abbiamo anche un package Test adibito al collaudo del software.

Source

Dal momento che all'interno del package, durante l'esecuzione, ci sono 2 timer che lavorano contemporaneamente (1 in source ed 1 in TimedForecast) e che entrambi accedono all'attributo forecast24H di TimedForecast, potrebbero nascere problemi di conflitto dei dati.

Per gestire i problemi di concorrenza, abbiamo utilizzato un Semaphore.

Server

Anche in questo caso, visto che più client possono contemporaneamente accedere attraverso RMI, al DB, è stato utilizzato un Semaphore per gestire la concorrenza. Come DB abbiamo utilizzato SQLite connettendoci attraverso JDBC (con il driver SQLite). Nel metodo storeEvents abbiamo utilizzato la funzionalità di batch di JDBC per raggruppare più events in una sola query ed inviarla al DB.

Database

Per contenere i dati degli eventi è stata usata una base di dati SQLite per la sua facilità di utilizzo e di integrazione.

Note sui tipi di dati scelti:

Per i CAP è stato utilizzato INTEGER invece di TEXT perché sappiamo la natura dei nostri dati. TEXT coinvolgerebbe un utilizzo di espressioni regolari complicati (es. `/[0-9]{5}/`) per garantire l'integrità della base di dati, i quali non sono supportati nativamente da SQLite. Siccome i CAP Italiani sono tutti numerici è molto più facile verificare che i numeri si trovano nel range [0, 99999] con un semplice `CHECK(`cap` BETWEEN 0 AND 99999)` e aggiungere gli zeri davanti alla selezione dei dati. Inoltre, questo approccio utilizza meno memoria (il che comunque non fa parte delle nostre priorità).

Per il tempo è stato utilizzato INTEGER che rappresenta Unix Epoch. Questa scelta permette di semplificare la validazione dei dati.

Per lo stato dell'evento è stato utilizzato un TEXT. È stato scelto TEXT perché SQLite non supporta gli ENUM. È possibile utilizzare gli INTEGER per gli stati avendo una conversione fatta tramite un CASE statement che converte TEXT in INTEGER e viceversa. Tale soluzione sarebbe però complicata da implementare. Oppure, si potrebbe usare una tabella a parte che contiene tutti i possibili valori dello stato, che comporterebbe l'utilizzo degli ASSERT e dei TRIGGER. Considerando tutti i compromessi, il tipo TEXT richiede molte meno risorse per l'implementazione e il mantenimento (ciò fa parte delle nostre priorità).

Per il tipo di evento è stato usato un semplice TEXT NOT NULL perché non conosciamo i possibili valori in anticipo.

User

User è il package che si occupa di gestire i dati dell'utente e di inviargli le informazioni.

All'interno troviamo la classe Caps che implementa soltanto metodi statici utili per filtrare i CAP partendo da provincia e comune. Questa classe contiene una struttura dati che mappa province e comuni con i rispettivi CAP. Tale struttura viene creata a partire da un file (provincia_comune_cap.csv) che contiene tutti i CAP italiani.

Altra classe che troviamo nel package User è TimedNotificator. Questa si occupa, attraverso i timer, di inviare richieste continue (ogni 5 sec) al server per richiedere i nuovi eventi riguardanti i CAP preferiti dell'utente (per nuovi eventi si intendono gli eventi che hanno come attributo data una data successiva all'istante corrente). Per farlo utilizza il metodo user.getEvents(SearchFilter). Una volta ottenuti, li sottrae a quelli del Set notifiedEvents e il risultato viene inviato all'interfaccia EventsNotificationShower.

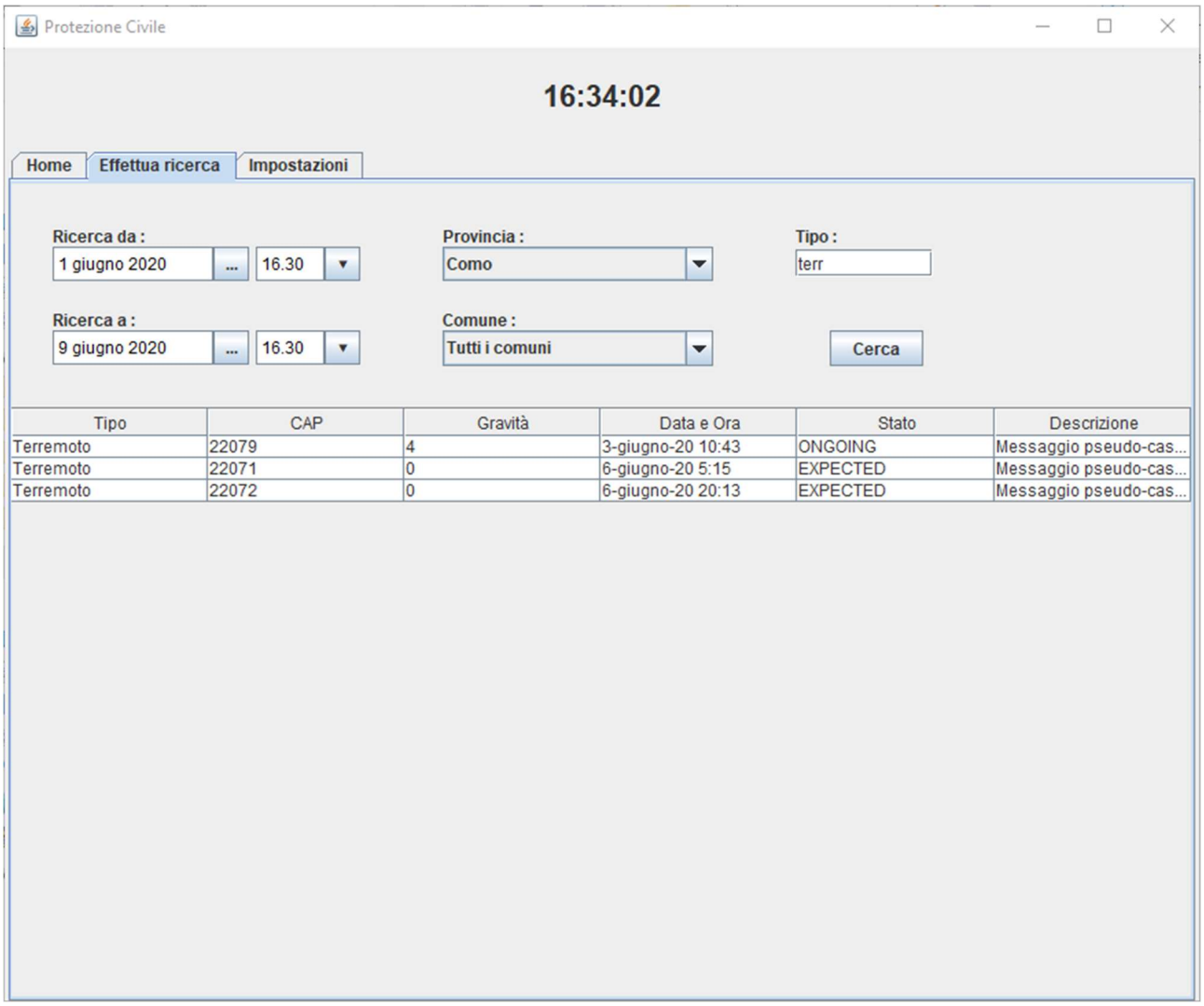
Se l'interfaccia riesce ad inviarli all'utente senza lanciare eccezioni, allora vengono inseriti in notifiedEvents e non verranno visualizzati nuovamente.

GUI

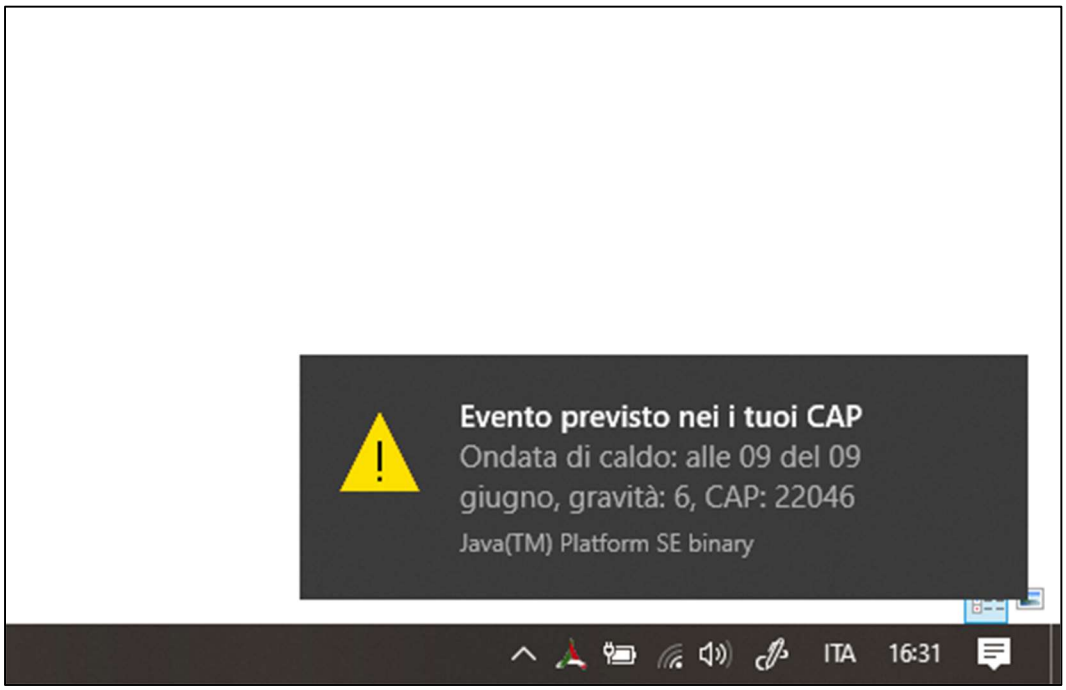
La gui è stata costruita con la libreria swing e progettata in maniera modulare. Infatti, all'interno di gui.component, si trovano tutti i componenti che compongono il Frame(FrameCivilProtectionUser). Questa soluzione ci offre un vantaggio in termini di riutilizzo e modificabilità.

Ad esempio, è possibile modificare la classe PnlEventsTable per cambiare l'aspetto di tutte le tabelle che mostrano gli eventi nella gui.

Per implementare la funzionalità di notifica è stato utilizzato il System.Tray, che invia notifiche al SO attraverso il metodo trayIcon.sendMessage. Tale soluzione è multipiattaforma ma, in ogni caso, controlliamo se il sistema in cui è stato avviato il software supporta la funzionalità. In questo modo, in caso non la supportasse, le notifiche verrebbero inviate sul terminale con System.out.println. La classe FrameCivilProtectionUser implementa l'interfaccia EventsNotificationShower.

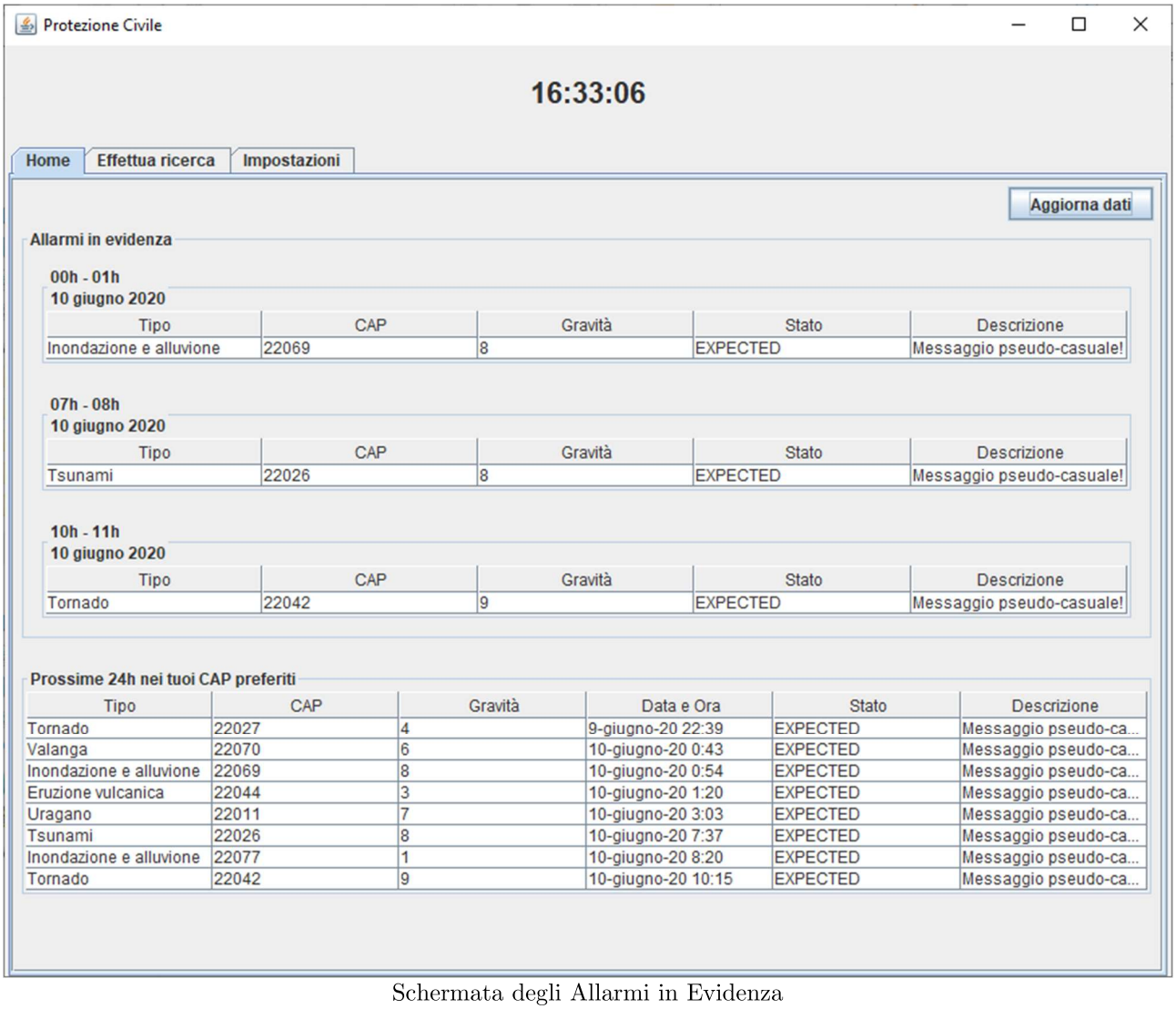


Schermata di Ricerca

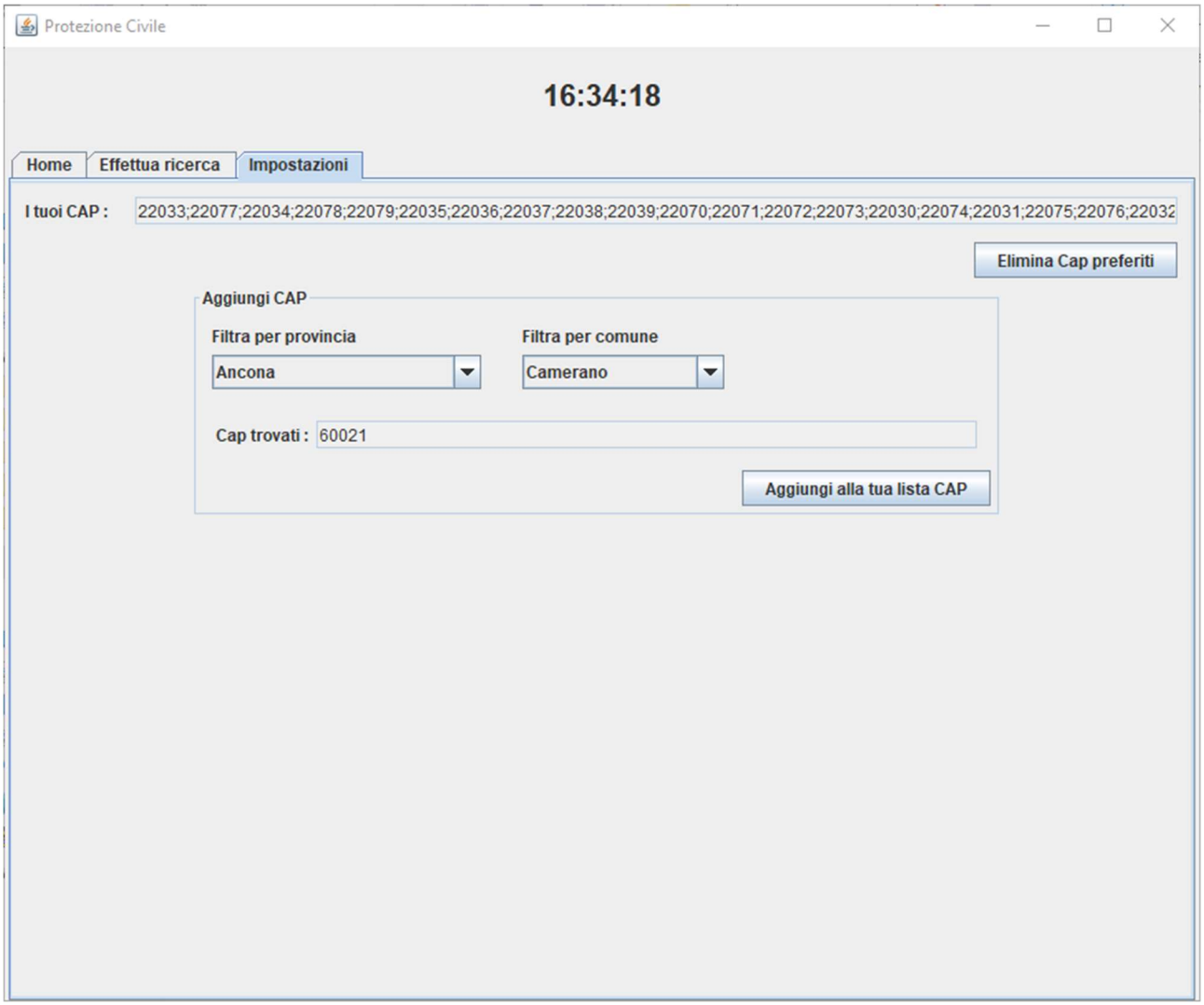


Esempio di notifica in Windows

Esempio di notifica in GNU/Linux



Schermata degli Allarmi in Evidenza



Schermata delle impostazioni

Common

La classe common è una classe da cui tutte le altre dipendono che contiene funzionalità utili all'intero sistema come la classe Event.

In quest'ultima, al fine di utilizzarla in un SortedSet, è stata implementata l'interfaccia Comparable che ordina gli eventi per: Data, Gravità, etc...

Abbiamo anche effettuato l'override dei metodi equals e hashCode perché quelli predefiniti non calcolavano i valori degli attributi non permettendo l'utilizzo degli HashSet.

Test

Per l'integration testing abbiamo usato Junit. Questi test coprono una parte del codice.