

Protezione Civile

Valeriy Litkovskyy, Alessio Spagnolo, Christian Premoli, Gabriele Bianchi

June 9, 2020

Introduzione

Fasi di lavoro

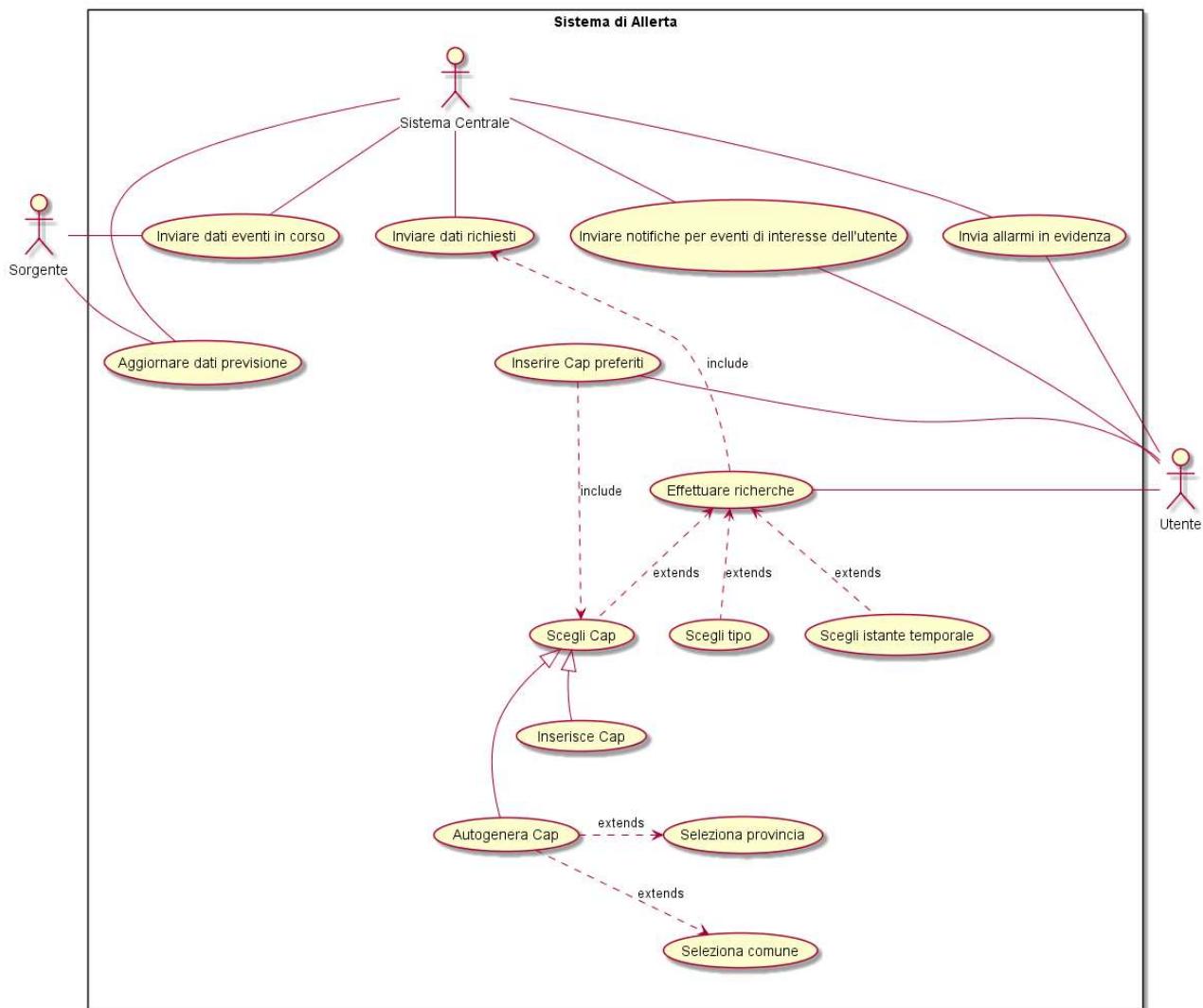
Per modellizzare il design abbiamo utilizzato il linguaggio UML, che mette a disposizione diversi diagrammi che permettono di delineare il comportamento che dovrà avere il programma. Abbiamo immaginato le varie funzionalità che avremmo dovuto implementare, e le abbiamo rappresentate nei diagrammi.

Abbiamo ritenuto opportuno realizzare: uno Use Case Diagram, un Class Diagram, un Package Diagram, un Object Diagram, un Deployment Diagram, due Activity Diagrams, due Sequence Diagrams, un Communication Diagrams e due State Diagram.

Per la creazione dei diagrammi abbiamo utilizzato PlantUML, un linguaggio di marcatura apposito.

Design

UseCase Diagram



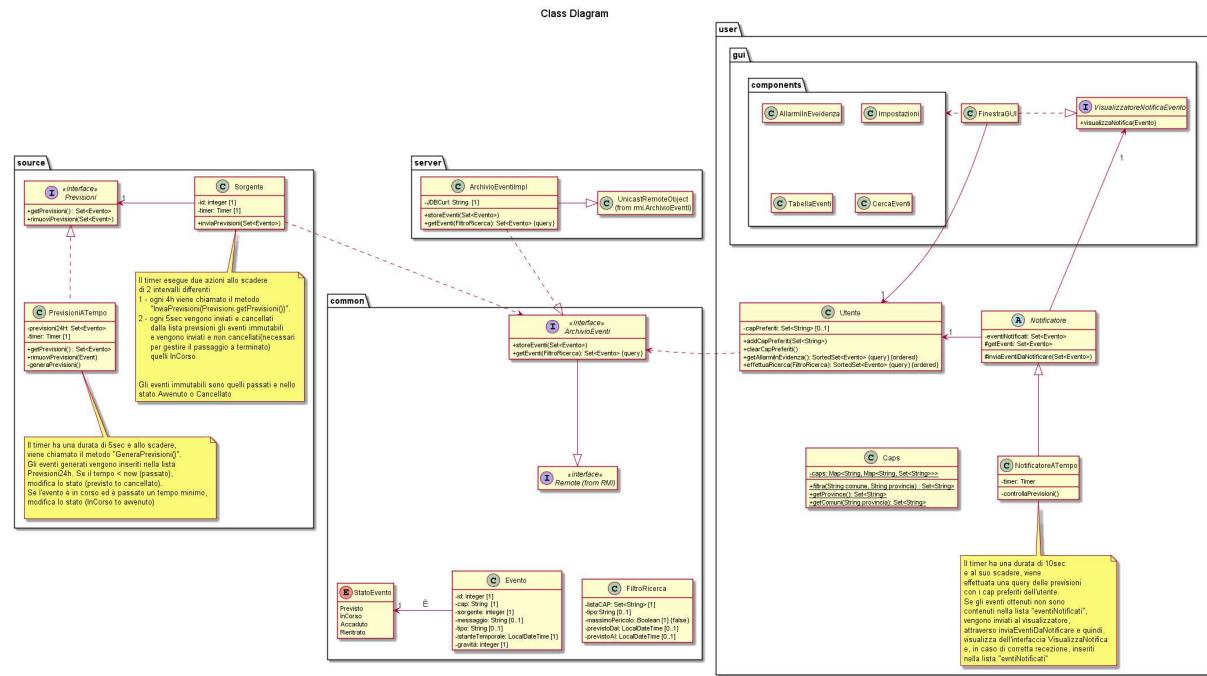
In questo diagramma vengono descritti i comportamenti dei tre attori del sistema di allerta: **Sorgente**, **Sistema Centrale** e **Utente**.

Sorgente si occupa di aggiornare le previsioni ed inviare al sistema centrale le informazioni relative agli eventi in corso.

Da parte dell'**Utente** avverrà l'inserimento dei CAP preferiti, questo ne implica obbligatoriamente la scelta, che avviene tramite inserimento esplicito oppure per autogenerazione in seguito all'inserimento di provincia e comune. Un'ulteriore azione che può svolgere l'**Utente** è effettuare ricerche. Questa funzione viene svolta scegliendo il tipo di evento e/o l'istante temporale in cui esso avviene.

Le ricerche implicano un'azione obbligatoria del **Sistema Centrale**, cioè l'invio dei dati richiesti in base ai parametri di studio selezionati dall'utente. Il Sistema Centrale possiede altri due casi d'uso: l'invio di allarmi in evidenza e l'invio di notifiche per gli eventi di interesse, cioè quelli che avvengono nelle località aventi un CAP selezionato dall'utente. Come si vede dal diagramma, il Sistema centrale è in realtà un attore interno al nostro sistema.

Class Diagram



Nel Class Diagram è possibile vedere la struttura del progetto e le classi utilizzate. Vediamo che il sistema è diviso in quattro package: **Source**, **Server**, **User** e **Common**. Quest'ultimo è un package che contiene tutte le classi usate da più di uno degli altri package. Per semplicità, sono stati omessi dai grafici i getter e setter che non danno informazioni aggiuntive e i dettagli delle classi della GUI.

Vediamo più nel dettaglio le varie classi:

Package Common:

- **ArchivioEventi**: è l'interfaccia utilizzata da tutti gli attori per la comunicazione attraverso RMI. Definisce due metodi necessari alla gestione del DB localizzato sul server.
- **Evento**: Oggetto che contiene tutte le informazioni (così come saranno sul DB) del tipo di dato Evento.
- **StatoEvento**: Enum con i possibili stati dell'Evento.
- **FiltroRicerca**: Classe che contiene le informazioni per effettuare una ricerca di eventi nel DB.

Package Source:

- **Sorgente**: Classe che si occupa di comunicare con il Server, attraverso RMI, al fine di inviare gli eventi generati dall'interfaccia **Previsioni**.
- **Previsioni**: Interfaccia che definisce i metodi, utilizzati dal Sorgente, per comunicare con il generatore di previsioni.
- **PrevisioniATempo**: Classe che implementa **Previsioni** che, attraverso un timer, stabilisce ogni quanto tempo generare delle nuove previsioni per le prossime 24 ore, da salvare in **previsioni24H**. Sarà possibile, attraverso il metodo `getPrevisioni`, per la classe Sorgente, leggere gli eventi salvati in **Previsioni24H**.
- È stata utilizzata un'interfaccia per rendere il sistema più modulare e poter creare delle classi che implementano **Previsioni** con un principio diverso da quello del timer.

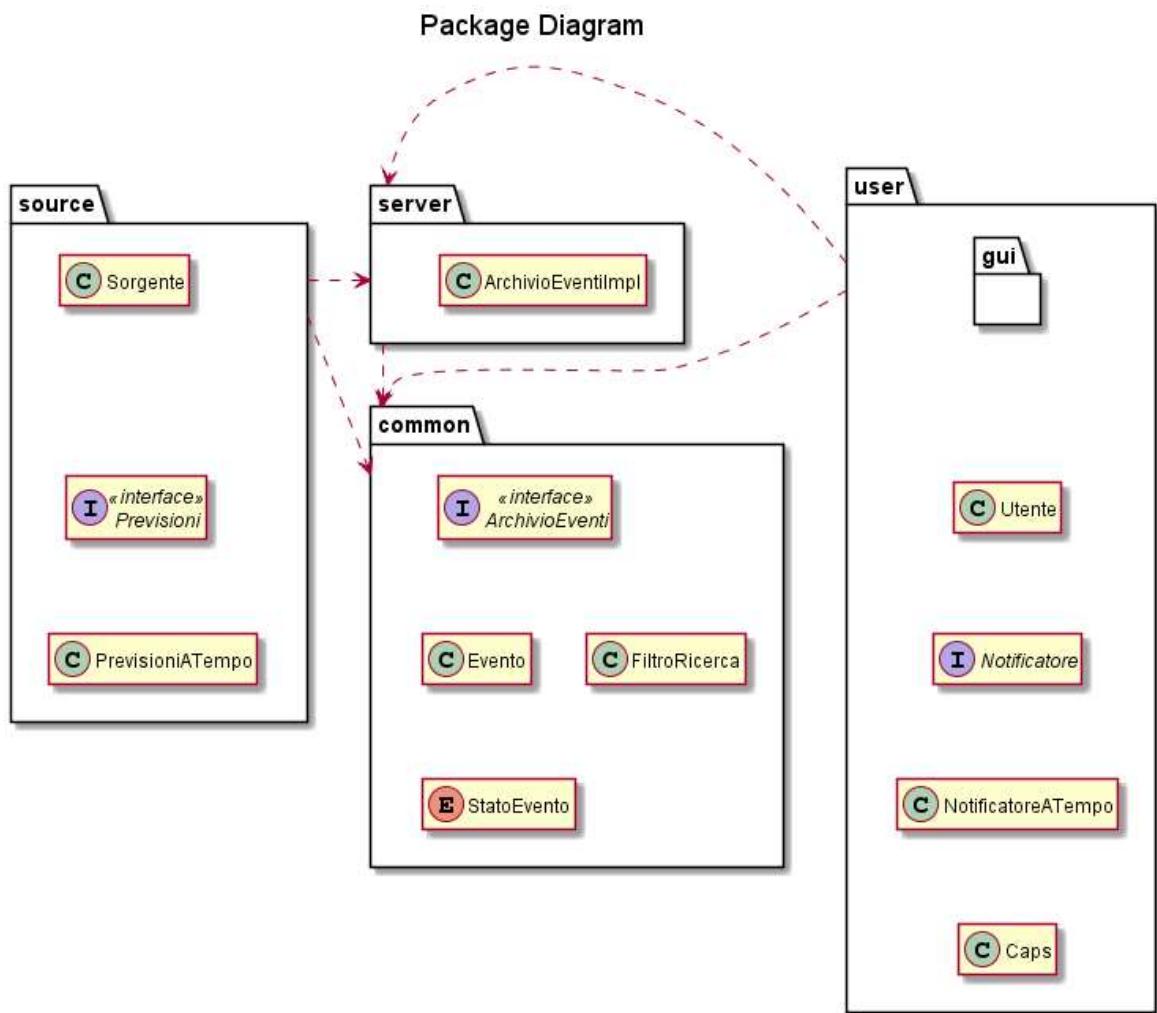
Package Server:

- ArchivioEventiImpl: Classe che si occupa di implementare l'interfaccia ArchivioEventi.

Package User:

- Utente: classe che contiene i CAP preferiti dell'utente (da utilizzare per le notifiche e gli eventi in home page) e i metodi per ottenere gli eventi dal server attraverso RMI.
- Notificatore: Classe astratta che si occupa di ottenere i nuovi eventi ed inviarli all'Interfaccia VisualizzatoreNotificaEvento che li mostrerà all'utente. Gli eventi notificati con successo vengono inseriti nel Set eventiNotificati e non verranno più notificati.
- NotificatoreATempo: Classe che estende Notificatore ed implementa un timer che controllerà i nuovi eventi del DB ogni 10sec.
- Caps: Classe che contiene metodi statici, utilizzata per filtrare i CAP da una coppia di provincia, comune.

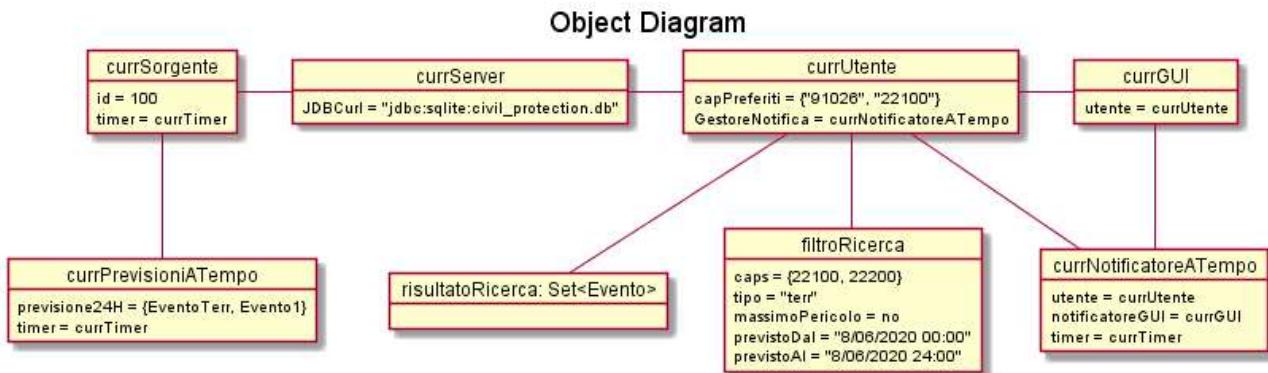
Package Diagram



Il Package Diagram ci mostra le dipendenze fra i vari package. Sappiamo che modificando una classe all'interno del common potrebbe avere ripercussioni in tutti gli altri package dal momento che dipendono da lui (source, server, user).

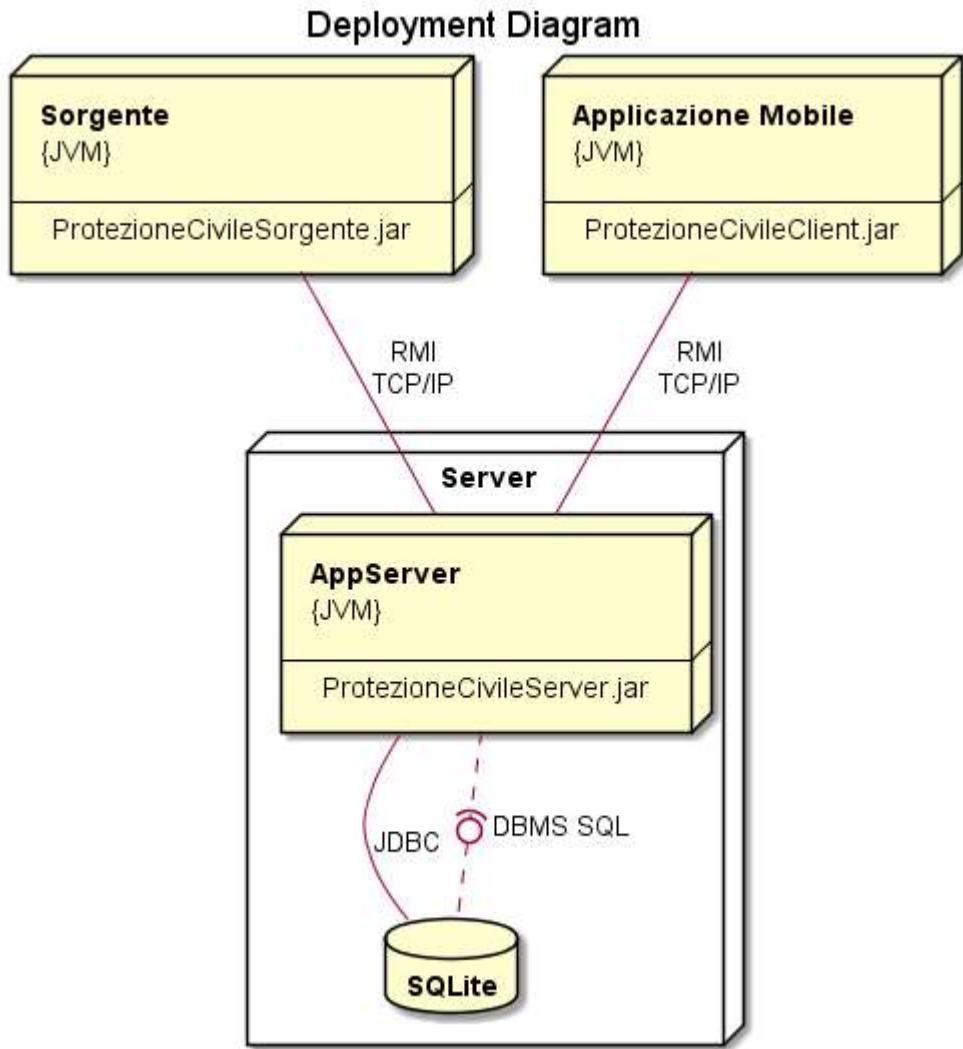
source e user dipendono da server perché un'implementazione diversa ad esempio delle query potrebbe modificare il comportamento di questi due package.

Object Diagram



L'Object Diagram mostra un momento statico dell'intera applicazione in cui viene mostrato un Sorgente (currSorgente) che attraverso currPrevisioniATempo invia previsione24H al currServer, che si occuperà di registrarle nel DB. Contemporaneamente currUser richiede degli eventi filtrandoli attraverso filtroRicerca e gli viene restituito un risultatoRicerca. Inoltre currNotificatoreATempo invia richieste al currServer utilizzando i capPreferiti dell'utente. Infine il currGUI si occupa di mostrare tutte le informazioni all'utente.

Deployment Diagram



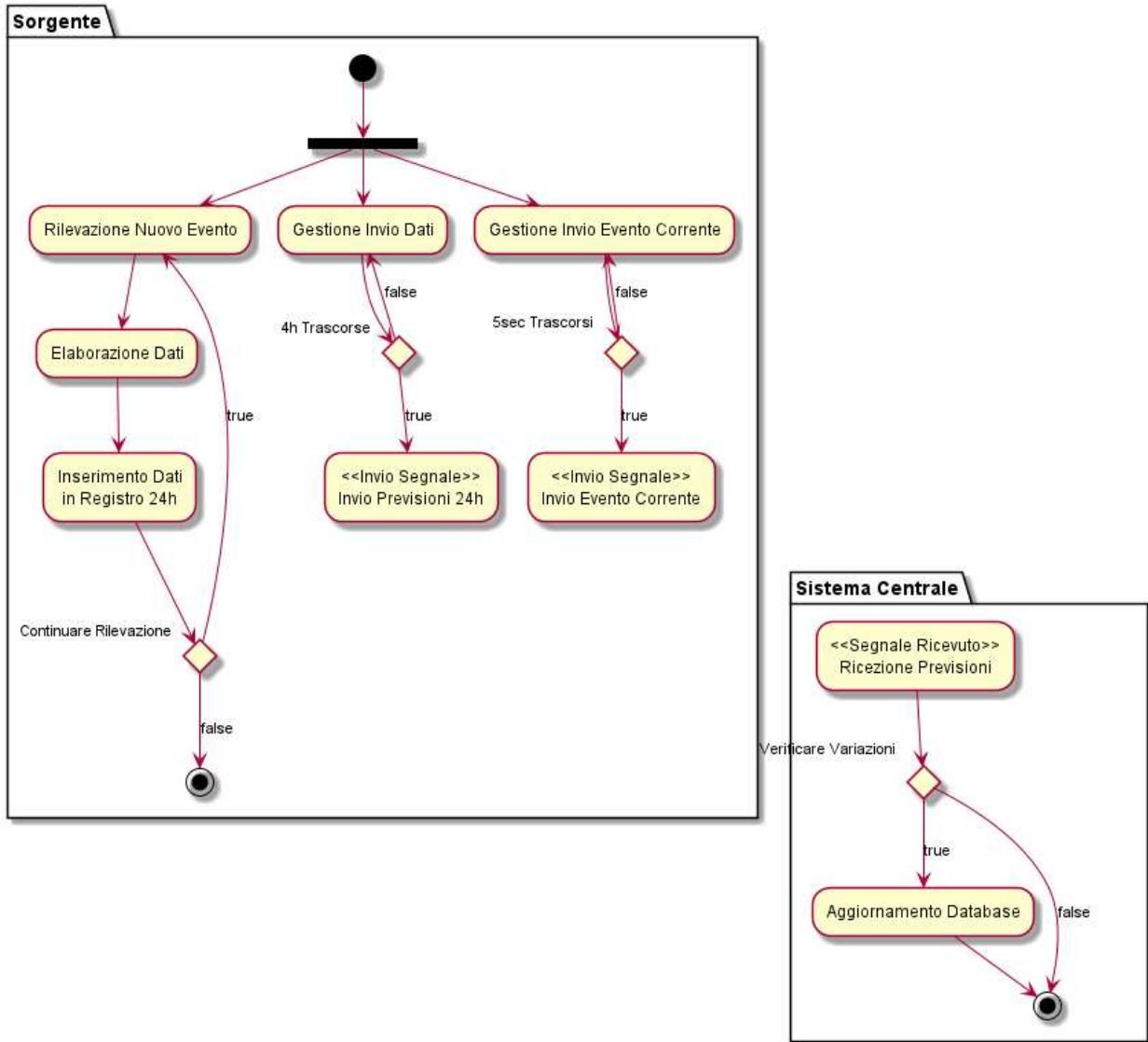
Il Deployment Diagram mostra la distribuzione fisica del sistema. Possiamo vedere come il sistema distribuito sia composto da tre software separati:

- `ProtezioneCivileSource.jar` per il Source
- `ProtezioneCivileServer.jar` per il Server
- `ProtezioneCivileUser.jar` per lo User

Source e User comunicano con il server attraverso RMI.

Il Server comunica con un DB SQL (nel nostro caso SQLite) attraverso JDBC.

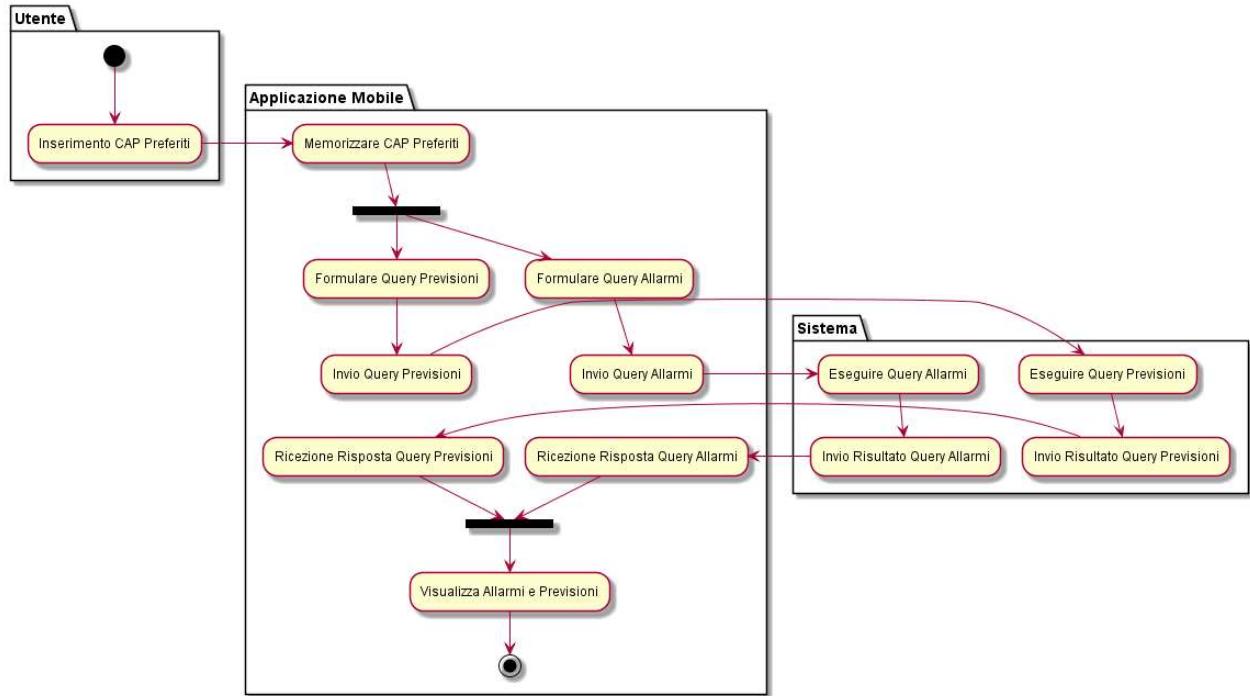
Activity Diagram - Sorgente



In questo diagramma viene rappresentata la sequenza di attività tra il **Sorgente** ed il **Sistema Centrale**.

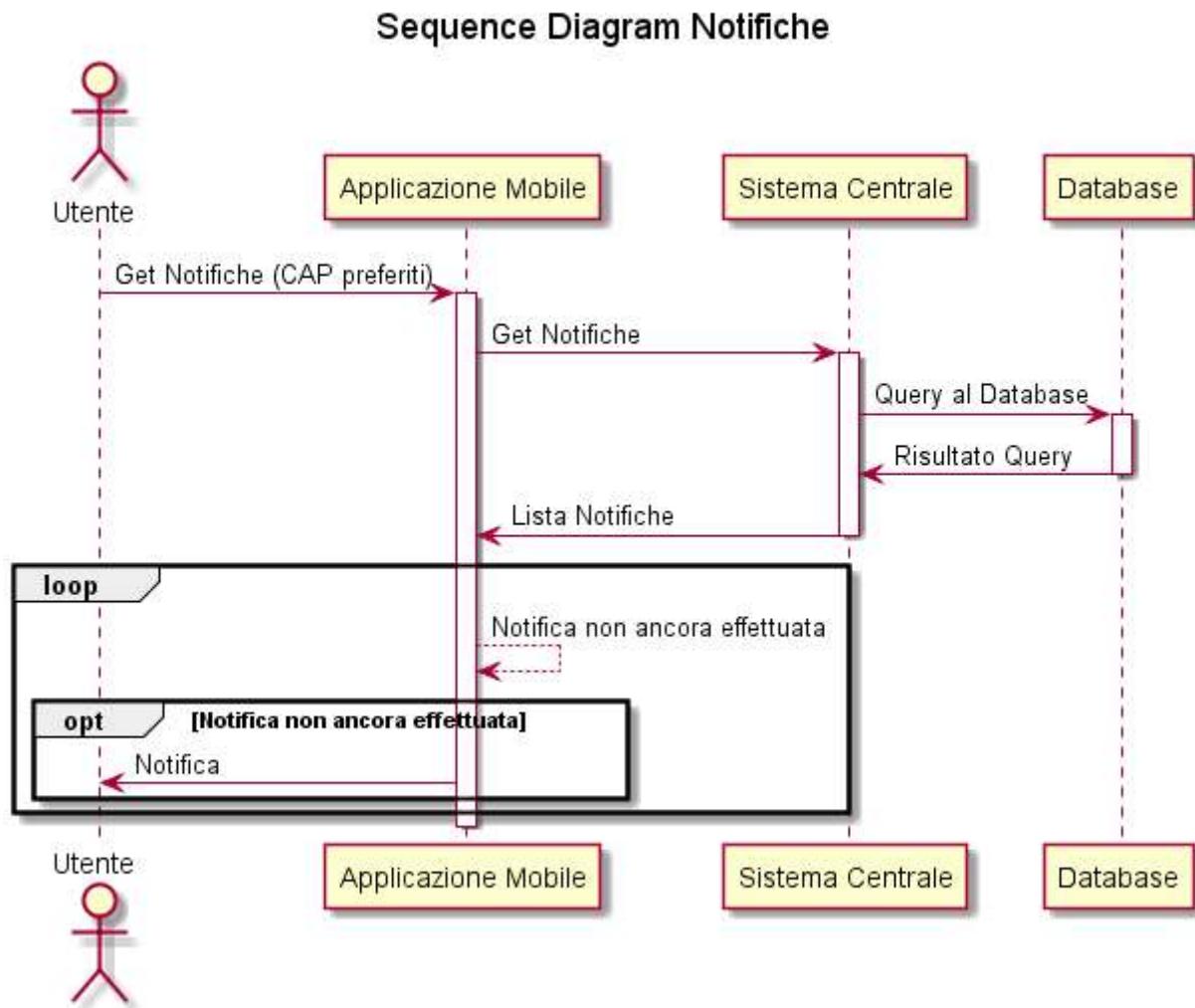
Nella sezione riguardante il **Sorgente**, dopo lo stato iniziale è presente una fork per rappresentare l'esecuzione simultanea di tre sequenze di attività. Nella prima vengono descritti i passi per la rilevazione e l'acquisizione di un nuovo evento, e viene effettuato un controllo per capire se la Sorgente vuole continuare queste operazioni. Nelle sequenze di attività parallele viene effettuato un controllo per sapere se è trascorso il tempo necessario per poter inviare i dati delle previsioni oppure quelli relativi ad un evento corrente. Nel primo caso devono essere trascorse 4 ore, mentre nel secondo 5 secondi. Se uno dei due controlli ha avuto esito positivo, vengono inviati i dati al **Sistema Centrale**, il quale si limiterà a verificare la presenza di variazioni nel database e, in caso positivo, quest'ultimo verrà aggiornato.

Activity Diagram - App Mobile



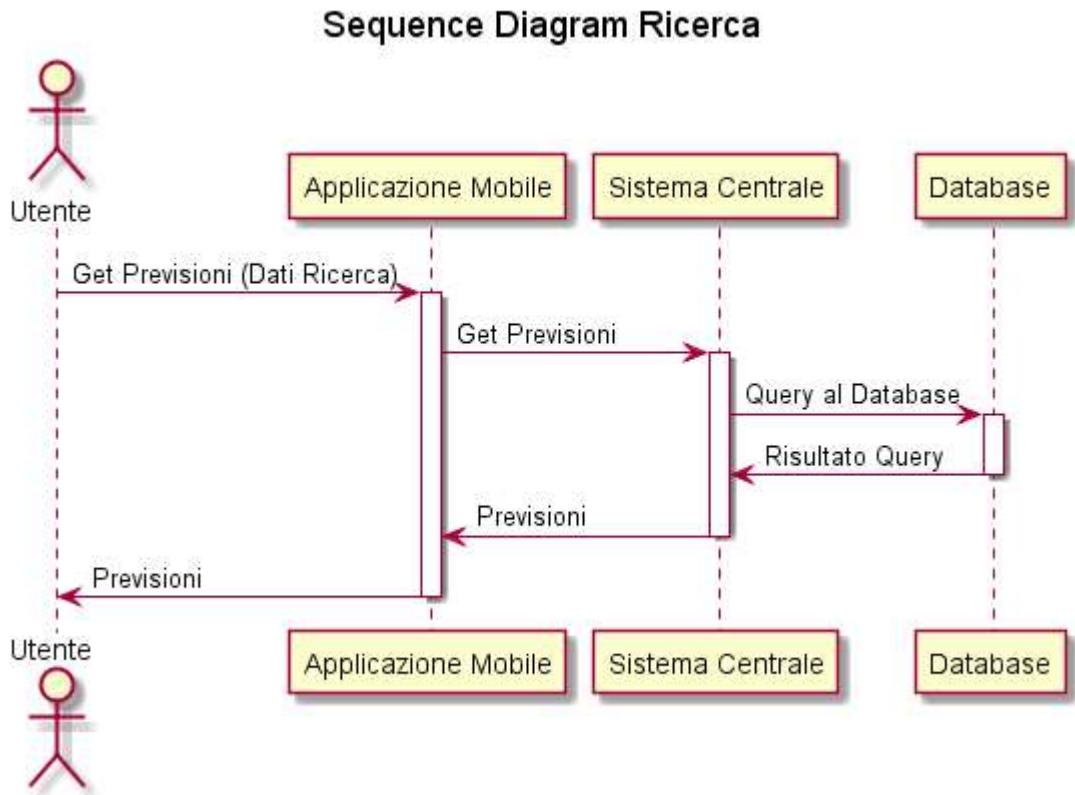
In questo Activity Diagram vengono descritte le attività principali che svolgono Utente, Applicazione Mobile e Sistema Centrale per comunicare tra loro. Dopo lo stato di avvio l'Utente inserisce i CAP preferiti nell'Applicazione Mobile per memorizzarli. In base ai dati inseriti, il software formula le query per ricavare le previsioni e gli allarmi, dopodiché le invia al Sistema Centrale. Qui vengono eseguite le interrogazioni al database, e il risultato viene restituito all'applicazione per essere mostrato all'utente in un elenco.

Sequence Diagram - Notifiche



In questo Sequence Diagram è descritto il comportamento di invio di una notifica. L'Utente imposterà i CAP preferiti nell'App Mobile, che andrà a richiedere le notifiche al Sistema Centrale. Questo invierà una Query al Database, che risponderà con l'elenco degli eventi da notificare che verrà inviato all'App Mobile. L'App dunque andrà ciclicamente a controllare di non aver già inviato quegli eventi come notifica, e in caso positivo li manderà all'utente.

Sequence Diagram - Ricerca



Il secondo Sequence Diagram riguarda la gestione delle ricerche effettuate dall'Utente. Questo andrà a selezionare dei parametri di ricerca nell'App Mobile, che verranno richiesti al Sistema Centrale. Questo invierà una Query al Database, che risponderà con l'elenco degli eventi da notificare. Il Sistema Centrale manderà poi le previsioni all'App Mobile, che le farà visualizzare all'Utente.

Communication Diagram - Ricerca

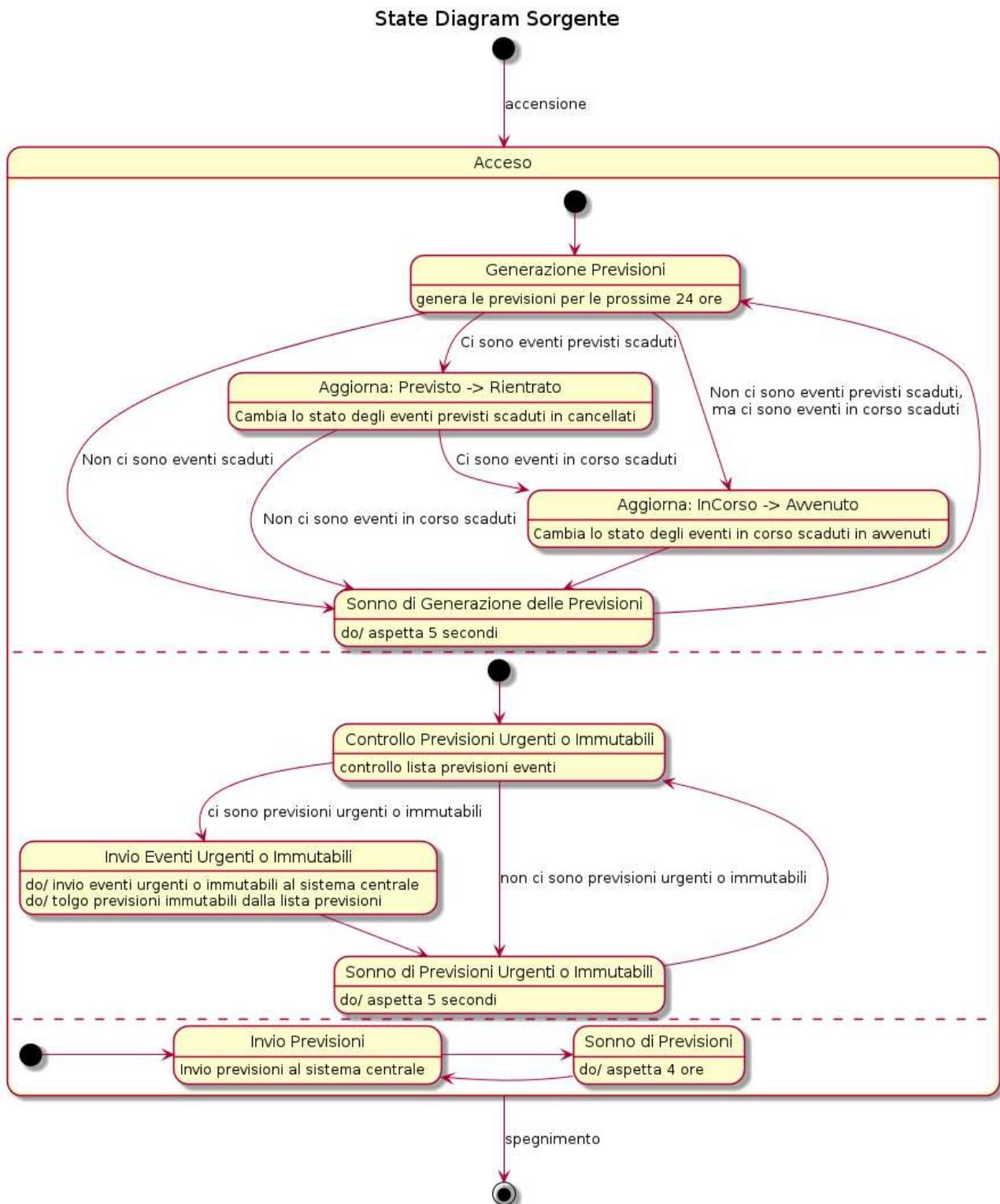


In questo diagramma è descritta la cooperazione che avviene tra le istanze del sistema allo scopo di effettuare una ricerca di un evento nel Database.

Le **Sorgenti** invieranno gli aggiornamenti sulle previsioni delle prossime 24h al **Sistema Centrale**, che con queste aggiorneranno il Database.

L'**Utente**, dunque, quando effettuerà una ricerca, darà all'**App Mobile** i dati necessari, con i quali invierà una Query al **Sistema Centrale**. Questo darà in risposta all'**App Mobile** l'elenco degli eventi che si trovano nei parametri di ricerca, che verranno dunque visualizzati dall'**Utente**.

State Diagram - Sorgente



All'accensione la sorgente passa nello stato **Acceso**. Dopo di che si avviano tre stati paralleli. Da qualsiasi sottostato di questi stati si può passare nello stato **Spento**.

Stato Invio Previsioni:

Invia previsioni nuove al sistema centrale e passa nello stato di **Sonno 4 ore**. Dopo aver aspettato per 4 ore, ricomincia con l'invio delle previsioni.

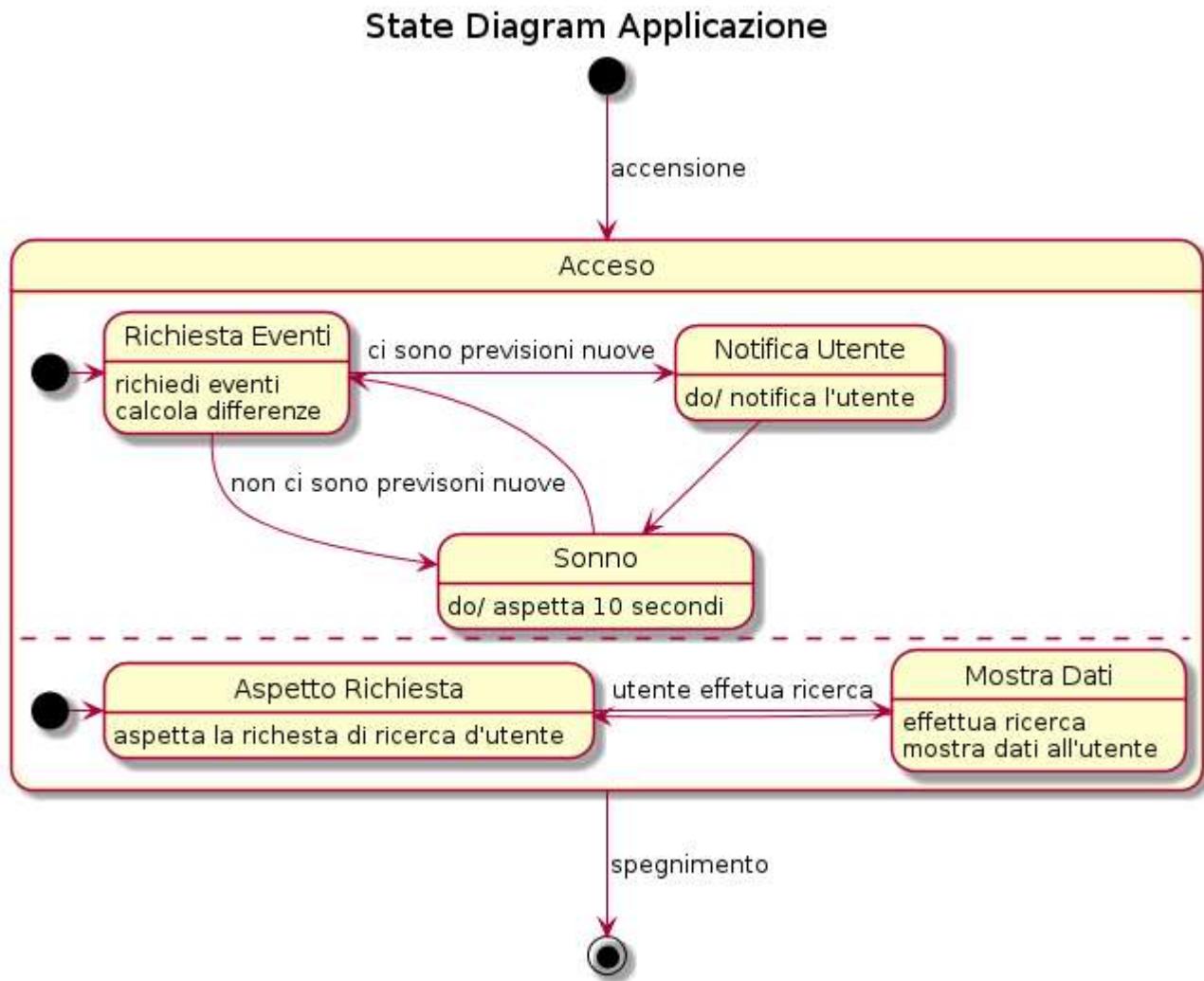
Stato Invio Previsioni Urgenti o Immutabili:

Questo stato parte con un controllo della lista delle previsioni. Tramite questo controllo si vogliono trovare delle previsioni urgenti o immutabili che devono essere subito inviate al sistema centrale. Le previsioni sono considerate *Urgenti* se hanno un livello di gravità maggiore o uguale a 8. Le previsioni sono *Immutabili* se sono nello stato di **Accaduto** o **Rientrato**. Se sono state trovate previsioni *Urgenti* o *Immutabili* si passano nello stato di **Invio Previsioni**. Durante questo stato verranno inviate le previsioni *Urgenti* e *Immutabili* al sistema centrale, dopo di che si tolgono gli eventi *Immutabili* dalla lista delle previsioni. Conseguentivamente si passa nello stato di **Sonno 5 secondi**. Se non sono state trovate previsioni *Urgenti* o *Immutabili* si passa direttamente nello stato di **Sonno 5 secondi**. Dopo aver aspettato per 5 secondi, si ricomincia con il controllo delle previsioni.

Stato Generazione Previsioni:

Si comincia con lo stato di **Generazione Previsioni**, durante il quale si generano e vengono aggiunte nuove previsioni nella lista previsioni. Se nella lista vengono trovate previsioni dove l'istante per cui è prevista precede di 30 minuti il tempo corrente e il suo stato è **In Corso**. In questo caso lo stato passa da **In Corso** ad **Avvenuto**. Se invece vengono trovate previsioni il cui istante è precedente al tempo corrente e il cui stato è **Previsto**, allora viene modificato in **Rientrato**. Conseguentivamente si passa nello stato **Sonno 5 secondi**. Dopo aver aspettato per 5 secondi, si ricomincia con la generazione delle previsioni.

State Diagram - Applicazione



All'accensione l'applicazione passa nello stato **Acceso**, dopo di che si avviano due stati paralleli. Da qualsiasi sottostato di questi si può passare nello stato **Spento**.

Stato Richiesta Eventi

In questo stato vengono richieste tutte le previsioni future che riguardano i CAP preferiti d'utente. Dopo di che si notifica l'utente con i possibili aggiornamenti delle varie previsioni. Consecutivamente si passa nello stato di **Sonno 10 secondi**. Dopo aver aspettato 10 secondi, si ricomincia con la richiesta previsioni.

Stato Ricerca Previsioni

Si aspetta finché l'utente effettui una ricerca di eventi. Quando viene effettuata una ricerca, vengono richiesti i dati necessari al sistema centrale e vengono mostrati gli eventi trovati all'utente. Dopo di che si passa di nuovo nello stato di **Aspetto Richiesta**.