UNIVERSITÀ DEGLI STUDI DI MESSINA

DIPARTIMENTO DI INGEGNERIA

Corso Di Laurea Triennale In Ingegneria Elettronica E Informatica

---

APPLICAZIONI PER AMBIENTI INTELLIGENTI BASATI SU MONGODB E SUL PROTOCOLLO COAP

Tesi di Laurea di:
Marco Ieni

Relatore:
Chiar.mo Prof. Massimo Villari

Correlatrice:
Chiar.ma Ing. Maria Fazio

Anno Accademico 2014/2015

Indice

Lista delle figure

Introduzione

L'obiettivo della tesi è la realizzazione di un sistema di monitoraggio e gestione di un ambiente intelligente. Gli utenti del sistema possono accedere ai servizi personalizzati e dinamici, che integrano i dati ambientali (provenienti da sensori) con le esigenze degli utenti, modificando le caratteristiche dell'ambiente stesso (tramite gli attuatori). Il sistema sviluppato sfrutta il protocollo CoAP come protocollo di comunicazione tra i dispositivi embedded (sensori ed attuatori) e immagazzina l'ingente quantità di dati che viene continuamente prodotta dai sensori per il monitoraggio ambientale nel Cloud, mediante il DBMS MongoDB.

Il lavoro della tesi è stato portato avanti in modo da fornire un'analisi ed un'implementazione delle tecnologie considerate, creando un sistema flessibile e funzionale per varie applicazioni.

Il servizio finale consente all'utente di:

- tenere sotto controllo i propri sensori, visualizzandone le misure effettuate nel tempo (anche applicando filtri come tag, gruppi o posizione);
- comandare i propri attuatori, stabilendo regole di funzionamento in base all'orario e alla locazione dell'interruttore attivato, accendendo solo determinate luci oppure caricando da remoto un proprio programma in python da eseguire ottenendo, così, dei servizi personalizzati;
- raggruppare i dispositivi in gruppi, in modo tale da avere una gestione più semplificata;
- rendere pubblici i propri dispositivi, in modo che tutti gli utenti registrati nel sistema possano accedervi;
- creare gruppi di utenti che hanno accesso solo a determinati gruppi di dispositivi. Per esemplificare, si pensi ad un'università in cui tutte le porte siano considerate degli attuatori collegati a uno strumento capace di identificare un utente (lettore di impronte digitali, tastierino numerico, card reader, ecc.). Si potrebbe abilitare a un gruppo di tirocinanti l'apertura di tutte le porte che conducono al laboratorio dove devono recarsi, in modo tale da farli accedere solamente a determinate zone e salvare l'orario di entrata e di uscita dal laboratorio di ogni tirocinante;
- ottenere informazioni in base alla posizione dei dispositivi. Ad esempio, è possibile ottenere una media della temperatura di un determinato luogo pesata in base alla precisione di ogni dispositivo a cui si ha accesso nelle vicinanze di un determinato punto, identificato da latitudine e longitudine.

Il presente lavoro si articola in quattro capitoli:

Capitolo 1 - Analisi delle tecnologie adoperate

Nel primo capitolo vengono analizzati nel dettaglio:

- il protocollo CoAP, approfondendo anche il motivo per cui esso è di fondamentale importanza per la comunicazione dei sistemi embedded;
- analisi di soluzioni cloud-based, esaminandone vantaggi e svantaggi rispetto all'approccio centralizzato.
- MongoDB, descrivendo tutte le principali peculiarità di questa tipologia di database;

Capitolo 2 - Scelta dell'implementazione CoAP

Nel secondo capitolo si riporta tutto il percorso che ha portato alla scelta dell'implementazione del CoAP da utilizzare, analizzando svantaggi, vantaggi e codici creati per ogni implementazione studiata.

Capitolo 3 - Il sistema sviluppato

Nel terzo capitolo viene presentato il sistema sviluppato, analizzando lo scenario di riferimento e i vincoli che esso impone, lo schema del sistema e le sue funzionalità (anche mediante l'ausilio dei sequence diagram).

Capitolo 4 - Dettagli implementativi

Nel quarto capitolo vengono illustrati alcuni screenshot della shell, che mostrano il funzionamento del sistema e i dettagli implementativi, come le query e le parti di codice più importanti e il diagramma delle classi principali.

Capitolo 5 - Sperimentazione

Nel quinto capitolo viene mostrato un particolare scenario applicativo utilizzato per testare il sistema sviluppato. In particolare, la sperimentazione è stata fatta ipotizzando un campus univeristario, ed implementando un servizio Smart Building Activity Based. Tale servizio permette di controllare i consumi elettrici legati al riscaldamento dell'edificio del campus.

Sono inoltre presenti le conclusioni e la bibliografia.

Capitolo 1. Analisi delle tecnologie adoperate

Le tecnologie più innovative impiegate nel progetto sono il protocollo CoAP, MongoDB e il cloud. Esse sono strettamente collegate tra loro e l'impiego di una, spesso, implica anche quello di un'altra.

Il CoAP, ad esempio, è utilizzato nell'ambito dell'IoT, l'internet delle cose, un nuovo trend che vede protagonisti gli oggetti che comunicano con la rete per inviare e ricevere dati. Il problema è che gli oggetti collegati ad internet saranno moltissimi e, di conseguenza, la mole di dati prodotta sarà così grande da non potere essere gestita da un solo server, per quanto potente esso possa essere. Si sente dunque l'esigenza di passare da un approccio verticale, in cui si potenzia sempre di più un server centrale, a un approccio orizzontale, in cui si aggiungono altre macchine a cooperare tra di loro, ovvero da un sistema centralizzato a uno distribuito.

Quest'ultimo sistema è detto Cloud e può essere sia a livello applicativo sia a livello dati. Per utilizzare il cloud a livello dati non possono essere utilizzate le tecnologie dei database relazionali, poiché questi presuppongono che tutte le tabelle del database siano contenute su una sola macchina, ma si devono impiegare i database non relazionali, in quanto questi ultimi consentono di distribuire i dati di uno stesso database su più server.

Un esempio di Database Management System non relazionale è MongoDB, poiché si allontana dalla struttura tradizionale basata sulle tabelle dei database relazionali in favore di documenti in stile JSON con schema dinamico rendendo, così, l'integrazione di dati di alcuni tipi di applicazioni più facile e veloce. Tra i vari DBMS non relazionali si è scelto MongoDB perché è un software libero, open souce, stabile e ben documentato. Inoltre è utilizzato come backend da un alto numero di grandi siti web e società di servizi come eBay, Foursquare, SourceForge e il New York Times [1]. Dopo aver spiegato il perché della scelta di queste tecnologie, di seguito verranno approfondite una per una.

1.1. CoAP

> Il Constrained Application Protocol (CoAP) è un protocollo di trasferimento web specializzato per l'uso con nodi vincolati e reti vincolate nel contesto dell'Internet delle cose.

[1] fonte: https://it.wikipedia.org/wiki/MongoDB

— http://coap.technology/

Il protocollo è di tipo RESTful e di livello applicativo, mentre la definizione e la descrizione di “nodi vincolati” e “reti vincolate” si può trovare nell’RFC 7228 [2].

1.1.1. Nodi vincolati

Un nodo vincolato è un nodo dove alcune delle caratteristiche che molto spesso sono date per scontate per i classici nodi di internet non sono presenti, spesso a causa di vincoli di costo e/o fisici come la dimensione, il peso e la potenza e l’energia disponibili.

In primo luogo vi sono limiti stretti su potenza, memoria e capacità di elaborazione delle risorse che rendono l’ottimizzazione di energia, codice e larghezza di banda una parte importante dei requisiti di progettazione. Inoltre, servizi di secondo livello come connettività completa, broadcast e multicast talvolta possono anche mancare.

Un altro vincolo può essere quello della mancanza di una interfaccia utente e della scarsa accessibilità in distribuzione che può comportare, per esempio, l’impossibilità di aggiornare il software.

Anche se questa non è una definizione rigorosa, definisce chiaramente il distacco tra i nodi vincolati e altri sistemi più potenti come i server tradizionali, computer desktop, laptop o potenti dispositivi mobile come tablet e smartphone.

L’uso di nodi vincolati spesso porta anche a vincoli sulle reti stesse, ma ci possono anche essere vincoli sulle reti che sono ampiamente indipendenti da quelli dei nodi. Per questo motivo si distinguono i “nodi vincolati” dalle “reti vincolate”.

1.1.2. Reti vincolate

Le reti vincolate sono definite come reti in cui alcune delle caratteristiche date per scontato con gli strati di collegamento di uso comune nelle reti dell’internet odierna non sono disponibili e, quindi, vi sono dei vincoli di progettazione. Questi vincoli possono includere:

- bassa bitrate;
- perdita di pacchetti e alta variabilità della perdita di pacchetti;
- caratteristiche di collegamento molto asimmetriche;

[2] http://tools.ietf.org/html/rfc7228

- severe penalità per l'utilizzo di pacchetti di grandi dimensioni (ad esempio a causa di un alto tasso di perdita di pacchetti);
- limiti di raggiungibilità nel tempo, poichè un numero considerevole di dispositivi potrebbe spegnersi in qualsiasi momento, per poi svegliarsi e comunicare per un breve periodo di tempo;
- mancanza di servizi avanzati come il multicast.

1.1.3. Il protocollo

Il CoAP, dunque, è stato creato per risolvere e nascondere al programmatore quasi tutte le problematiche elencate sia per i nodi che per le reti vincolate e presenta le seguenti caratteristiche:

- è basato sul modello REST quindi, come per l'HTTP, i server rendono disponibili le risorse attraverso una URL e i client accedono a queste risorse usando metodi come GET, PUT, POST e DELETE;
- riesce a utilizzare risorse minime, sia sul dispositivo che sulla rete, perché invece di uno stack di trasporto complesso come il TCP, lavora con l'UDP, che non è orientato alla connessione e non implementa i controlli del flusso e della congestione, più adatti per scambiare messaggi di grosse dimensioni. Inoltre presenta un header fissato a quattro byte e una codifica compatta di opzioni, che consentono ai piccoli messaggi di non causare frammentazione;
- molti server possono operare senza stato;
- chiavi RSA a 3072 bit come scelta predefinita dei parametri DTLS. Questo aspetto è molto importante, infatti una delle cose che impedisce di più all'IoT di diffondersi è la questione della sicurezza. A tal proposito il CoAP offre una protezione avanzata senza per questo compromettere il funzionamento persino sui nodi meno potenti;
- collegamento semplice tra HTTP e CoAP mediante proxy cross-protocol, dal momento che condividono lo stesso modello REST: un web client che utilizza HTTP può anche non accorgersi che sta accedendo a una risorsa di un sensore, in quanto il server CoAP locale può occuparsi di inoltrare ai nodi della sottorete le richieste in CoAP, traducendole da quelle HTTP originali;
- servizi di discovery già integrati nel protocollo, in quanto è già presente un modo per scoprire le proprietà dei nodi della rete;
- come HTTP, il CoAP può trasportare diversi tipi di payload e può anche identificare che tipo di payload si sta usando, compresi formati come XML, JSON, CBOR e molti altri che sono già integrati di base;

- ottimizzato per funzionare anche su controllori con 10KB di RAM e 100KB di spazio di codice. Per capire l'importanza e la potenzialità di questa caratteristica basta pensare che si prevede che l'Internet of Things avrà bisogno di miliardi di nodi. Ovviamente meno questi nodi saranno costosi, più grande sarà il risparmio complessivo, quindi avere un protocollo che funziona bene anche su dispositivi con prestazioni estremamente basse è essenziale;
- osservazione di risorse con il modello publish/subscribe, ovvero un server notifica i client che si sono registrati a una specifica risorsa quando questa cambia;
- resource discovery già integrata e supporto a trasmissione di grandi quantità di dati, suddividendo i dati in blocchi nel mittente e ricomponendoli al livello applicativo in ricezione (questa feature è chiamata block-wise transfer);
- supporto nativo al multicast;
- affidabilità della comunicazione. Nonostante il CoAP sia basato su UDP, infatti, è implementato un meccanismo per garantire che i messaggi siano stati ricevuti dal destinatario. In particolare, per un determinato intervallo di tempo, quando si invia un pacchetto si aspetta un ACK, che può contenere anche un eventuale payload e, se questo non arriva, viene ritrasmesso il messaggio; al termine dell'intervallo il pacchetto si considera scaduto e, quindi, non ricevuto dal destinatario.

Tutte queste caratteristiche rendono il CoAP il protocollo ideale per impieghi che rientrano nelle architetture dell'IoT.

1.2. Cloud

Per approfondire la tecnologia del Cloud sono stati letti e analizzati vari articoli di Joe Wienman, il fondatore di Cloudonomics, un rigoroso approccio analitico e multidisciplinare che coinvolge economia, economia comportamentale, statistica, matematica, teoria della complessità computazionale, simulazione e teoria dei sistemi per caratterizzare il talvolta poco intuitivo business multi-dimensionale e i benefici dell'esperienza utente del cloud computing e di altri modelli di business on-demand e pay-per-use [3].

1.2.1. Che cos'è

> In informatica con il termine inglese cloud computing (in italiano "nuvola informatica") si indica un paradigma di erogazione di risorse

[3] fonte: http://www.joeweinman.com/bio.htm

> informatiche, come l'archiviazione, l'elaborazione o la trasmissione di dati, caratterizzato dalla disponibilità on demand attraverso Internet a partire da un insieme di risorse preesistenti e configurabili.
>
> — Peter Mell, Timothy Grance *The NIST Definition of Cloud Computing. NIST, Special Publication 800-145, Settembre 2011.*

Le risorse non vengono pienamente configurate e messe in opera dal fornitore apposta per l'utente, ma gli sono assegnate, rapidamente e convenientemente, grazie a procedure automatizzate, a partire da un insieme di risorse condivise con altri utenti, lasciando all'utente parte dell'onere della configurazione. Quando l'utente rilascia la risorsa, essa viene similmente riconfigurata nello stato iniziale e rimessa a disposizione nel pool condiviso delle risorse, con altrettanta velocità ed economia per il fornitore [4].

Alcuni esempi che mostrano le potenzialità del cloud sono:

- Messaggiare tramite smartphone;
- Condividere foto e aggiornamenti su Facebook o Twitter;
- Guardare un video di Youtube su una TV;
- Condividere file su Dropbox.

Ma oltre a queste applicazioni che coinvolgono l'uomo, ve ne sono molte altre che coinvolgono le macchine (IoT), in cui una varietà di endpoints sono connessi al cloud:

- Telecamere di videosorveglianza che mandano video in diretta a un archivio di video basato sul cloud per una maggiore sicurezza;
- Device che monitorano continuamente alcuni parametri dei pazienti, come la pressione sanguigna, i battiti cardiaci, che possono essere mandati all'ospedale più vicino in caso di valori al di fuori del normale;
- Case intelligenti con dispositivi in grado di spegnere ed accendere a distanza dispositivi ad alto consumo come condizionatori, lavatrici, asciugatrici e lavastoviglie, in modo da ridurre al minimo i consumi;
- Fattorie che monitorano lo stato dell'ambiente, come la temperatura, le precipitazioni, per assicurare che il terreno riceva esattamente l'irrigazione necessaria [5].

[4] https://it.wikipedia.org/wiki/Cloud_computing

[5] Joe Weinman, "Defining a cloud", http://www.techradar.com/news/internet/cloud-services/defining-a-cloud-1209348

1.2.2. Caratteristiche

Un'analisi delle caratteristiche principali di un servizio cloud-based viene effettuata nell'articolo di Joe Weinman "Cloudonomics: A Rigorous Approach to Cloud Benefit Quantification" [6]. Esse sono:

Infrastruttura *C*omune

Le risorse sono standardizzate, condivise e allocate dinamicamente.

La penalità del costo del servizio segue la funzione $\frac{1}{\sqrt{m}}$, dove m sta per il numero di richieste indipendenti che riceve il sistema. Questo vuol dire che più richieste ci sono, più il costo che si paga per mantenere il servizio è "giustificato", perché lo si sta sfruttando. Se invece ci sono poche richieste rispetto a quelle supportate, il costo che si sta pagando per gestire più richieste è, ovviamente, uno spreco.

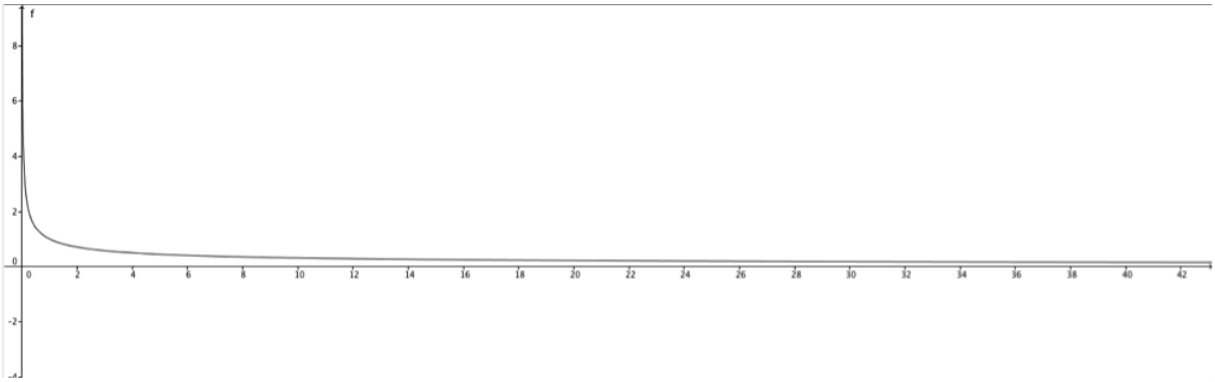


Figura 1.1. Andamento della penalità del costo del servizio

Come si nota dal grafico la funzione è decrescente, quindi più m è grande, più la penalità del costo si abbassa, ma se m aumenta troppo bisogna aumentare la capacità del sistema e, quindi, investire altri soldi in esso perchè, se la capacità del sistema sarà troppo piccola rispetto alle richieste ricevute, quest'ultimo soffrità di un'alta latenza e verrà a mancare la reattività dell'applicazione del consumatore.

Indipendenza dalla *L*ocazione

La reattività del servizio non tiene conto della posizione dell'utente, perchè i server possono essere sparsi ovunque, con benefici derivanti dalla riduzione della latenza e la valorizzazione dell'esperienza dell'utente.

[6] Ottobre 2011, https://www.csiac.org/sites/default/files/journal_files/stn14_4.pdf

La latenza è fortemente, ma non perfettamente, correlata con la distanza. La ragione di questa imperfezione ha a che fare con le anomalie a livello fisico della rete, come gli hop dei router, il tempo di conversione tra ottico ed elettronico e viceversa, le congestioni, i collegamenti antiquati, ecc.

Su un piano, sia per il caso peggiore sia per quello atteso, la latenza è proporzionale al raggio del cerchio centrato sul nodo. Di conseguenza, l'area coperta è proporzionale al raggio e al numero di nodi (Weinman, 2011).

In particolare, per n nodi su un piano, l'area A coperta dipende dal raggio r, correlato con la latenza/distanza, e una costante di proporzionalità k che dipende dalla strategia di copertura. Si deduce, dunque, che $A = kn\pi r^2$. Pertanto, se l'area A è una costante, si ricava che $r \propto \frac{1}{\sqrt{n}}$ dove n sta per il numero di nodi aggiustato di un fattore $\frac{1-\cos(2\beta)}{1-\cos(\beta)}$.
Quindi, come si deduce dall'andamento della funzione, se si hanno pochi nodi e se ne aggiungono altri, si ottiene un miglioramento notevole nella latenza mentre, se di base si hanno già molti nodi, il miglioramento sarà esiguo.

Si faccia un esempio trascurando il fattore di aggiustamento e supponendo di avere un servizio che viene offerto a 100000 utenti, in cui ogni richiesta da parte di un utente impieghi un tempo di elaborazione di 1ms e che 20200 di questi utenti facciano richiesta contemporaneamente. Se si avesse solamente un nodo, il sistema sarebbe pronto ad accettare nuove richieste dopo 20,2 secondi. Se aggiungessimo un altro nodo 10100 utenti verrebbero diretti verso il primo nodo e altri 10100 verso l'altro, quindi da 20,2 secondi un ipotetico 202001° utente che faccia richiesta qualche istante dopo arriverebbe ad aspettare 10,1 secondi, ottenendo un miglioramento del 50%. Si ponga, invece, il caso in cui il sistema sia formato da 100 nodi. Per lo stesso numero di richieste ogni nodo ne gestirebbe 202 e, quindi, il sistema sarebbe pronto a gestire nuove richieste dopo 202ms. Se si aggiungesse 1 nodo al sistema, ogni nodo gestirebbe 200 nodi e quindi il sistema sarebbe pronto dopo 200ms piuttosto che 202ms. Come si nota dai risultati, il miglioramento, che nel primo caso era di ben 10,1 secondi, ora è solo di 2ms, cioè appena dello 0,99%.

Per quanto riguarda il fattore di aggiustamento esso serve a tenere in considerazione la distanza tra nodo e utente, che ovviamente più è elevata, più va a pesare sulla latenza. β è l'angolo formato dal nodo e dall'utente, con vertice preso al centro della terra. Questo coefficiente è molto significativo quando i nodi non sono dislocati in maniera omogenea sul pianeta, mentre se ci sono molti nodi ben disseminati sulla superficie terrestre β sarà molto piccolo, perché ogni utente avrà un nodo vicino ad esso e quindi il fattore tenderà ad 1.

Accessibilità *O*line

Un servizio cloud-based è più flessibile rispetto ad uno centralizzato.

Quando si esamina la connettività, il costo potrebbe essere semplice da calcolare: dollari per Gigabyte trasferiti o il costo di router o strutture ottiche, ma il valore è difficile da quantificare, dal momento che è una esternalità. Un buon approccio è di considerarne il costo marginale e usarlo per compensare i benefici connessi con il cloud puro o ibrido.

E' stato calcolato che una struttura ad hub rispetto ad una P2P ha un beneficio di $\frac{1}{n}$, dove n sta per il numero di connessioni. Ecco il grafico di questa funzione:

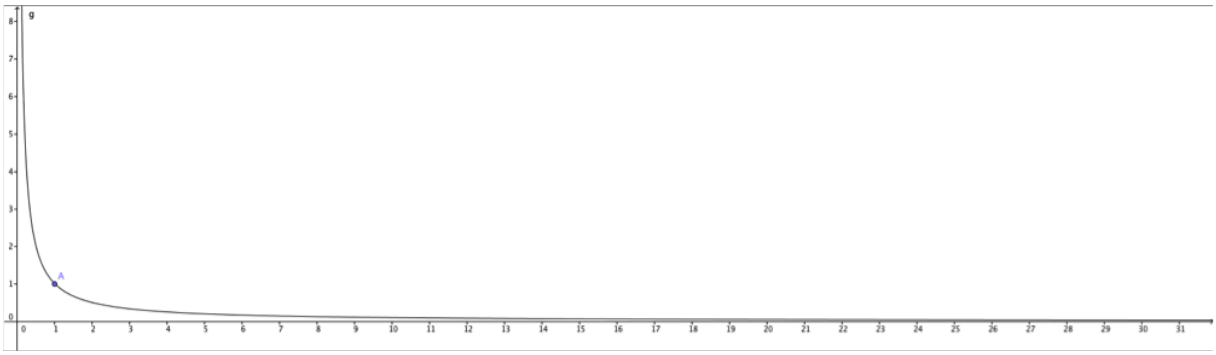


Figura 1.2. Andamento del beneficio di una struttura ad hub rispetto ad una P2P

Come si può notare notare, da un punto di vista prestazionale, il P2P conviene praticamente sempre, tranne nel caso in cui vi sia un solo nodo, ma per una maggiore facilità di gestione a volte si preferisce adoperare comunque una struttura centralizzata. Utilizzando politiche smart all'interno di una infrastruttura di rete (il che significa non utilizzare algoritmi semplici come FCFS o Round Robin per il bilanciamento del carico, ma utilizzare metodi più intelligenti) porta a un miglioramento di $\frac{1}{2} + \frac{1}{n+1} - \sqrt{\frac{\pi}{2} n}$ dove n sta per il numero di path. Ecco il grafico:

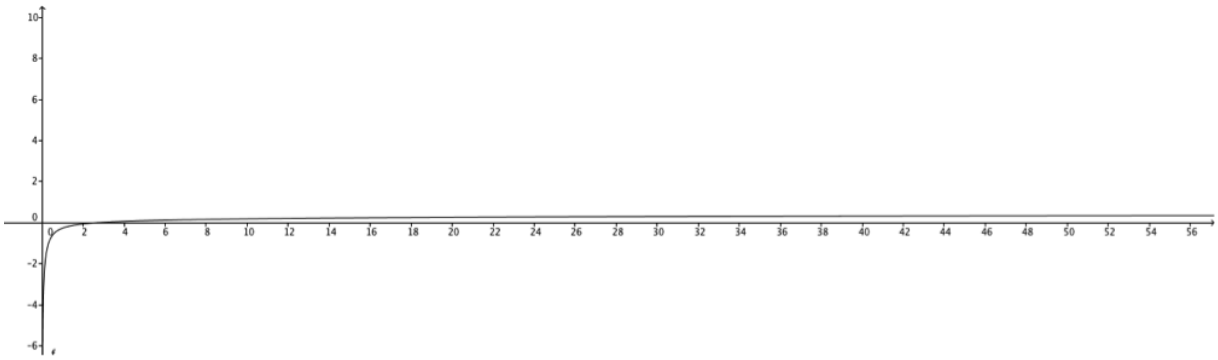


Figura 1.3. Andamento dei benefici che l'utilizzo di politiche smart porta all'interno di una infrastruttura di rete

Prezzi vantaggiosi (*U*tility pricing)

Con il cloud vi è la possibilità di pagare le risorse solo al momento dell'uso effettivo, con benefici apprezzabili soprattutto per gli ambienti con livelli di domanda variabile.

Il cloud, per definizione, è utile quando costa meno di realizzare un sistema privato o quando il carico di lavoro è molto variabile. Se il numero di richieste subisce molti sbalzi, infatti, significa che il sistema deve essere pronto a ricevere in ogni momento il picco più alto di traffico che potrebbe arrivare, mentre il cloud permette di richiedere le esatte risorse che servono in quel determinato momento, senza sprechi e pagando effettivamente solo per le risorse utilizzate.

Si consideri una richiesta di risorse variabile nel tempo e la si indichi con D(t) con $0 \le t \le T$. La richiesta di picco sarà dunque $P = \max\{D(t)\}$, mentre la domanda media sarà $A = \mu\{D(t)\}$. Sia il premio dell'utilità U il rapporto tra il costo per l'utilizzo di una unità del cloud e quello per l'utilizzo di un'unità dedicata. Ad esempio se si decide che si ha bisogno di una macchina, quest'ultima può essere comprata a rate (ad esempio per 300€ al mese) o affittata quando ci serve (ad esempio a 45€ al giorno). 300€ mensili al giorno vogliono dire circa 10€, quindi U = 4,5 (= 45€ / 10€). B sarà il costo di base per le risorse di proprietà.

Il prezzo per un utilizzo medio deve tenere in conto A, non P, quindi il costo totale per la richiesta di risorse D(t) sarà dato da A×U×B×T, che è il valore dell'integrale definito $\int_0^T U \times B \times D(t)dt$. Per una soluzione con risorse proprietarie non esiste il concetto di pay-per-use, quindi il prezzo pagato sarà sempre quello riguardante la richiesta di picco, non quella media, quindi il prezzo totale sarà dato da P×B×T. Per avere un costo minore delle risorse on-demand rispetto a quelle proprietarie si deve avere $A \times U \times B \times T < P \times B \times T$. La B e la T si semplificano, portando alla soluzione che il cloud

è più economico quando $\frac{P}{A} > U$. Si deduce, dunque, che se il costo di un'unità del cloud è U volte più grande di quello di una risorsa dedicata, ma il rapporto tra la richiesta di picco e quella media è superiore ad U, una soluzione di tipo cloud sarà sempre meno costosa.

Risorse on-*D*emand

Il valore dell'on-Demand consiste nell'avere sempre l'esatto numero di risorse di cui si ha bisogno in ogni momento e con il cloud si può raggiungere questo risultato. In base alla richiesta, infatti, si possono ottenere o rilasciare risorse dal cloud provider che si utilizza.

Si considerino i costi di penalizzazione associati all'insufficienza e all'eccesso di risorse. Se si indica con D la domanda e con R le risorse, il costo di penalizzazione è proporzionale a |D-R|. Se la domanda e le risorse sono tempo varianti, invece, il costo di penalizzazione è proporzionale a $\int |D(t) - R(t)| dt$.

Se la domanda è costante, basta il resourcing tradizionale mentre, se non è lineare o se non è prevedibile, allora sorgono dei problemi e le risorse on-demand diventano indispensabili.

Si dimostra ora che la penalità del costo di un servizio centralizzato aumenta esponenzialmente con il tempo se la funzione della domanda è esponenziale.

Data una richiesta uniformemente distribuita da 0 a P, in un tempo T, con una penalità del costo del servizio di c, la penalità totale per risorse fissate, anche se si imposta il livello in modo ottimale, sarà $\frac{1}{6} \times P \times T \times c$.

Quando la funzione della domanda è esponenziale, ovvero $D(t) = e^t$, e l'intervallo di risorse fisso è k, le risorse saranno $R(t) = e^{t-k}$, quindi la penalità del costo del servizio sarà dato da $D(t) - R(t) = e^t - e^{t-k} = e^t(1 - e^{-k})$.

Come è possibile notare, ogni intervallo di risorse fisso che cerca di soddisfare la domanda in base al suo livello corrente ha una penalità del costo del servizio che cresce esponenzialmente con il tempo, quindi in presenza di una domanda che con andamento esponenziale non conviene utilizzare un servizio centralizzato.

1.2.3. Perchè utilizzarlo

Il cloud ormai fa parte della vita di tutti i giorni: da consumatori, si spende parte del tempo su social network basati sul cloud e usando app scaricate dal cloud e, a livello

lavorativo, si usano svariati programmi basati sul cloud. I vantaggi legati al cloud e i motivi per cui le compagnie dovrebbero aggiornare i loro sistemi a questa tecnologia sono molteplici:

- riduzione dei costi;
- incremento della “business agility”;
- miglioramento dell’esperienza dell’utente, riducendo la latenza dei servizi interattivi;
- aumento delle capacità computazionali, offrendo la possibilità di far cooperare insieme un numero di risorse “vicino all’infinito”.

Bisogna però specificare che il cloud, come anche big data, social e IoT, diventa particolarmente efficace quando le compagnie lo utilizzano per ottenere vantaggi strategici. Vi sono quattro principali modi mediante i quali le compagnie utilizzano il cloud:

Information excellence

Le aziende sfruttano le informazioni per ottimizzare dinamicamente la produzione, i servizi e altri processi di business. L’analisi ha portato da sempre a un processo di miglioramento a lungo termine, a maggior ragione in questo momento, in cui i dati sono accessibili in tempo reale e possono essere sottoposti ad algoritmi di analisi che portano a una riduzione notevole dei costi;

Solution leadership

Creazione di prodotti e servizi differenziati e collegati alla rete mediante servizi basati sul cloud. Le macchine di ultima generazione o sistemi di monitoraggio di attività, sono alcuni degli esempi di questa nuova generazione di soluzioni capace di interfacciarsi ad internet;

Collective intimacy

Raccolta massiccia di dati degli utenti, in modo da estrapolarne preferenze e abitudini e per fornire, quindi, delle raccomandazioni individuali mirate a ciascun cliente. Un esempio sono raccomandazioni di film o libri, ma anche la generazione di terapie mediche per i pazienti basate sulle caratteristiche di ogni individuo e sui suoi sintomi;

Accelerated innovation

L’insieme di tutti quei processi che consentono di portare avanti l’innovazione ancora più velocemente. Alcuni esempi possono essere esperimenti a basso costo, crowdfunding e crowdsourcing. Il cloud, inoltre, permette di far interagire tra di loro i

dispositivi che fanno parte dell'IoT permettendo, ad esempio, l'ottimizzazione delle smart city in modo da avere una sincronizzazione di scuola bus, ambulanze, ecc. [7]

1.2.4. Aspetti economici

La crescita esplosiva del Cloud è dovuta ai benefici finanziari e strategici e la tariffazione pay-per-use è un attributo di base. Qualcuno dice che la riduzione dei costi e la business agility sono i due più importanti benefici del Cloud, altri indicano l'economia di scala dai grandi fornitori come principali benefici del cloud e altri prevedono che tutta l'IT si sposterà nel cloud.

Quando si considerano i costi di un sistema, non bisogna solamente guardare al costo delle unità, ma anche al loro utilizzo. Se si usa solo il 33% di un sistema, significa che per ogni risorsa utilizzata ce ne sono 2 inutilizzate e il costo effettivo delle unità, dunque, triplica. Quando si considerano le unità, il prezzo è visibile nel loro costo, mentre quando si guarda all'utilizzo, un prezzo alto può essere causato da una scarsa gestione delle risorse, ma anche dall'inevitabile risultato di picchi di carichi di lavoro in presenza di capacità fissata.

Anche se il cloud teoricamente ottimizza i costi, ci sono alcune spese addizionali da tenere in conto:

- La ricerca per trovare il giusto provider, anche se intermediari come broker possono aiutare;
- Il costo per migrare le applicazioni a una architettura di tipo cloud se queste non sono state pensate fin dall'inizio per esso;
- Costi di tipo tecnologico, come infrastrutture di rete, trasferimento dei dati o tool di gestione;
- I costi di un'organizzazione che revisiona il servizio offerto dal provider.

Avere la giusta quantità di risorse per soddisfare le richieste ha un chiaro beneficio economico. Se vi sono troppe risorse, infatti, significa che parte del capitale usato per esse poteva essere investito diversamente o risparmiato mentre, se ve ne sono poche, l'applicazione funzionerà lentamente o non funzionerà del tutto. Se non si ha la possibilità di fare una stima di questa quantità è consigliabile affidarsi (anche parzialmente) al cloud pubblico che, mediante risorse su richiesta, garantisce sempre la giusta quantità di risorse.

[7] Joe Weinman, "What's the real reason to do cloud, again?", 3 ottobre 2014. http://www.cloudcomputing-news.net/news/2014/oct/03/whats-the-real-reason-to-do-cloud-again/

Tuttavia, se l'andamento delle richieste è completamente piatto, non c'è un vero motivo per affidarsi a servizi on-demand, se non per non rimanere impreparati alla possibilità che questo andamento possa diventare esponenziale in futuro. In tal caso, infatti, affidarsi al cloud pubblico è vitale, poiché altrimenti alcune di queste richieste non saranno soddisfatte e, quindi, vi sarà una perdita economica.

Le infrastutture del cloud, tipicamente, sono geograficamente disperse per garantire una riduzione della latenza. Quest'ultima garantisce dei benefici in termini di business, infatti, tempi di caricamento minori implicano che più clienti possono visitare più pagine web e fare più acquisti o che gli impiegati possono essere più produttivi [8].

1.2.5. Cloud ibrido

La prospettiva economica di cloud privato o pubblico può essere semplificata nel seguente modo: il primo possiede risorse dedicate che sono proprietà dell'azienda che le utilizza e che richiedono un costo fisso indipendentemente da quante di queste risorse si stanno utilizzando, mentre il secondo ha delle risorse pronte all'uso, condivise anche con altre aziende e che vengono "affittate" all'occorrenza, quindi vi è un costo variabile in base a quanto le si utilizza. Per il cloud privato, dunque, vi sono delle risorse fisse, che sono quelle che l'azienda ha comprato e di cui si occupa di fare manutenzione, mentre per il cloud pubblico è possibile richiedere quante risorse servono, quindi si avrà una capacità variabile.

Qualsiasi mix di questi due approcci puri per andare incontro ai bisogni di una o più applicazioni può essere definito come architettura ibrida (Hybrid architecture).

Guardando anche ad altri settori si può notare che, sia il modello su cui si basa il cloud pubblico (a breve termine, su richiesta e con pagamento solo al momento dell'uso) sia quello su cui si basa il cloud privato (proprietà privata), esistono da moltissimo tempo ed hanno un grande successo a livello di business.

L'uomo, quindi, ha tratto i suoi vantaggi sia dall'uno sia dall'altro, dimostrando di non poter fare a meno di nessuno dei due: si immagini di non poter pagare un mutuo, di utilizzare solamente i mezzi pubblici, che tutte le luci degli appartamenti funzionassero solo ad energia solare, o ancora che non si potrebbero affittare stanze negli alberghi.

Quello che si fa in informatica, in questo caso, è quello che fa l'uomo tutti i giorni, ovvero prendere il meglio da entrambi i modelli pubblico e privato optando, quindi, per l'approccio ibrido.

[8] Joe Weinman, The Nuances of Cloud Economics (le sfumature del Cloud Economics), http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=7057586

In moltissimi settori gli approcci ibridi sono prevalenti, tra cui l'economia, l'ottimizzazione delle prestazioni o il comportamento umano.

In generale, in presenza di richieste variabili, le risorse ibride rappresentano un mix ottimale di un costo inferiore per le risorse dedicate che eseguono la parte di richieste costanti e per le risorse pay-per-use per la parte delle richieste variabili. Per richieste costanti si intendono quelle richieste che sono sempre presenti e che non dipendono da fattori esterni, mentre per richieste variabili si intendono quelle richieste che variano in modo imprevedibile.

L'approccio ottimale nella maggior parte dei casi, dunque, è utilizzare il cloud privato per le richieste che si presentano con costanza e che, quindi, sono prevedibili e utilizzare il cloud pubblico solamente nel caso di superamento della soglia di capacità del cloud privato, ovvero in presenza di richieste variabili.

Qualora i costi di uno dei due approcci dovessero abbassarsi, il cloud ibrido consentirebbe di espandersi verso l'approccio più economico, quindi vi è una flessibilità non indifferente. Investire tutti i propri investimenti in una propria infrastruttura, adottando quindi esclusivamente il modello del cloud privato, potrebbe rivelarsi una mossa totalmente errata se un domani i cloud provider abbassassero drasticamente il costo del cloud pubblico. D'altra parte, essere bloccato nel cloud pubblico potrebbe essere un problema nel caso in cui il proprio provider dovesse avere cali di perfomance, per esempio.

Lo spostamento di applicazioni da un ambiente privato al cloud pubblico può significare anche costi di migrazione, quindi è meglio adottare l'approccio ibrido fin da subito.

Una sottile differenza tra cloud pubblico e privato che viene spesso trascurata è quella delle prestazioni. Mentre il cloud pubblico offre numerose opzioni e configurazioni standard, il controllo finale sulle architetture fisiche si ha con le risorse di proprietà, che possono essere configurate con elementi infrastrutturali come memoria, core e svariate altre scelte che possono soddisfare esattamente le esigenze dell'applicazione. Un'architettura con elementi più costosi e adatti all'applicazione può offrire un migliore rapporto tra prestazioni e prezzo, ma si va incontro a una grossa perdita in termini di flessibilità.

Di seguito si analizzano gli scenari e le architetture in cui il cloud ibrido può essere impiegato. Un ambiente di cloud computing ibrido comprende tipicamente risorse private, uno o più servizi di cloud pubblico ed i "load balancers", bilanciatori di carico che instradano gli utenti alle risorse private o pubbliche.

In casi semplici, come la visualizzazione di pagine web statiche, i load balancers possono instradare gli utenti indifferentemente verso risorse private o pubbliche, altre volte il cloud può fungere da front-end e il data center aziendale da back-end o viceversa. Un esempio in cui il cloud funge da back-end e le risorse private da front-end potrebbe essere quello in cui gli ordini commerciali e le transazioni di un servizio di vendita vengono gestite all'interno del cloud privato, mentre il cloud pubblico si occupa del content delivery (distribuzione di contenuti) per un catalogo web-based e di contenere il database del catalogo.

Oltre al cloud nello spazio si può anche pensare al cloud nel tempo, anche se è una visione meno tradizionale. Vi sono quattro ibridi generici nel tempo:

- Lo sviluppo e il test dell'applicazione viene fatto nel cloud pubblico mentre, una volta che viene realizzata, l'applicazione viene eseguita all'interno del cloud privato aziendale;
- Nello scenario inverso al precedente, lo sviluppo e il test dell'applicazione viene fatto nel cloud privato o anche in un computer desktop o in una workstation mentre, quando l'applicazione è ultimata, viene spostata nel cloud pubblico;
- L'applicazione viene sia sviluppata che eseguita su cloud privato, ma viene spostata nel cloud pubblico a causa di strategie economiche o di altra natura;
- Nello scenario inverso al precedente, una società, magari una startup, basa la sua IT (tecnologia dell'informazione) inizialmente sul cloud pubblico e, una volta che raggiunge un certo potere economico decide di investire in un proprio data center, migrando le sue applicazioni nel cloud privato.

Molte combinazioni di tutti questi scenari si verificano continuamente sulla base dei requisiti di business.

Ad esempio, in un approccio che oggi è estremamente prevalente, diversi cloud provider pubblici integrano applicazioni o componenti nel data center aziendale, ad esempio la validazione di una carta di credito, un SaaS per il CRM (Customer relationship management), per la condivisione di documenti o per lo storage.

Idealmente la scelta di cloud pubblico, privato o ibrido dovrebbe essere completamente separata da quella dell'architettura dell'applicazione allo stesso modo di come, quando si compra una macchina, la scelta di affittarla, pagare in contanti o tramite rate non dovrebbe influenzare sulla scelta del modello della stessa.

I requisiti per attuare questa strategia sono semplici. L'applicazione dovrebbe essere distribuibile facilmente sia nel caso di cloud privato, pubblico o entrambi, senza

riscrivere parti di codice e dovrebbero essere disponibili dei tool automatici di gestione per abilitare varie componenti dell'applicazione in modo da aumentare la scalabilità e spostare l'applicazione da cloud privato a pubblico e viceversa e per fornire dati chiave come le performance dell'applicazione e le differenze di costo, in modo da ottimizzare intelligentemente i due ambienti.

Dietro questi requisiti di alto livello vi sono tecnologie specifiche di migrazione di macchine virtuali, dischi rigidi virtuali, configurazione di gestione di reti private virtuali, sicurezza, ecc. [9]

1.3. MongoDB

> MongoDB è un database open-source, basato sui documenti e progettato per la facilità di sviluppo e scalabilità.
>
> — https://docs.MongoDB.org/manual/

MongoDB (da "hu**mongo**us", enorme) è il più popolare DBMS non relazionale, ovvero di tipo NoSQL. Esso, dunque, non presenta la struttura tradizionale basata su tabelle dei database relazionali, ma adotta dei documenti in stile JSON con schema dinamico, rendendo l'integrazione di dati di alcuni tipi di applicazioni più facile e veloce [1]. Basti pensare che il JSON è uno dei formati più utilizzati per l'invio di strutture su internet per cui, la maggior parte delle volte, è immediato prendere le informazioni da questi file e inserirli nel database.

1.3.1. I documenti

I documenti sono gli analoghi delle strutture dei linguaggi di programmazione che associano chiavi a valori. Strutture del genere sono ad esempio i dizionari, le hash table, ecc. Formalmente i documenti sono di tipo BSON, che è una rappresentazione binaria di JSON con informazioni sui tipi aggiuntive. Nei documenti, il valore di un campo può essere qualsiasi tipo di dato BSON, compresi anche altri documenti, array e array di documenti.

I documenti sono salvati nelle collection, che sono un gruppo di documenti correlati che hanno un set di indirizzi comuni. Se si vuole fare un confronto con la logica relazionale, le collection sono le analoghe delle tabelle, mentre i documenti sono analoghi alle righe,

[9] fonte: Joe Weinman, "Hybrid Cloud: The Best of Both Worlds", http://blogs.technet.com/b/server-cloud/archive/2015/02/04/hybrid-cloud-the-best-of-both-worlds-an-introduction-to-the-benefits-of-hybrid-cloud.aspx

con la differenza che una collection non ha bisogno che i suoi documenti abbiano la stessa struttura.

Mentre in SQL si è abituati a incrociare i dati di più tabelle, in MongoDB le query selezionano documenti da una singola collection. Questi ultimi hanno sempre un campo chiamato "_id", che deve essere unico e che funge da chiave primaria. Quando non viene specificato dall'utente, esso viene creato automaticamente. Gli sviluppatori di MongoDB consigliano di farlo generare automaticamente lasciando il campo vuoto se non vi sono esigenze particolari.

L'ordine dei documenti ritornati da una query non è definito se non si specifica una sort. Questo comportamento è simile al non ordinamento verticale di una tabella SQL. Se si vuole fare un paragone tra i database non relazionali e quelli relazionali, in genere i database relazionali hanno più funzionalità di quelli non relazionali, però quelli non relazionali hanno più performance e scalabilità orizzontale, in quanto il database è distribuito su più server.

Rispetto alle tabelle, il modello Json da dei vantaggi in termini di agilità e flessibilità. Se nelle prime, infatti, c'è una definizione di uno schema rigido che può essere modificato con grande sforzo e modificando anche i preesistenti dati, in MongoDB vi sono i documenti che hanno uno schema dinamico, in cui la definizione della struttura della tabella viene specificata al momento dell'inserimento dei dati, quindi una modifica di una parte di questi dati si fa semplicemente mediante una update e non ha bisogno di stravolgere il resto della collection. Il modello dei dati, dunque, può evolvere facilmente senza il bisogno di modificare la base di dati ogni volta che si decide anche solo di aggiungere un campo ad una tabella. In questo modo ci si può adattare più velocemente ai cambiamenti dell'applicazione, in quanto la struttura dei dati non viene fatta per i dati in se stessi, ma per l'uso che se ne fa, ovvero in base allo schema di accesso ad essi.

Un altro vantaggio dei documenti sulle tabelle è la maggiore vicinanza con l'attuale orientamento della programmazione: gli oggetti. Salvare un oggetto in MongoDB è molto più semplice che salvarlo in delle tabelle, perché tutti i suoi attributi, che siano stringhe, array o persino altri oggetti, possono essere mappati direttamente in un documento JSON.

1.3.2. Scalabilità

Per la scalabilità si utilizza l'approccio dello sharding, ovvero il processo di salvare i dati distribuendoli su più server. Prima, con il modello relazionale, lo sharding era a

livello applicativo, quindi si dividevano i dati all'interno dei vari database, ma tutte le operazioni, sia di scrittura sia di lettura, erano guidate dalle applicazioni (normalmente si faceva un hashing oppure un range di una chiave di dati). Con i database non relazionali, invece, lo sharding è trasparente ed automatico. Quest'ultimo, in MongoDB, è realizzato mediante uno strato intermedio (posto tra l'applicazione e i vari server) che si occupa di distribuire le query. Esso funziona come una sorta di proxy e gira su dei query router, che possono sia essere messi su delle macchine a parte sia girare sulla macchina su cui risiede l'applicativo e, come si evince dall'immagine seguente, l'applicazione comunica con loro attraverso dei driver.

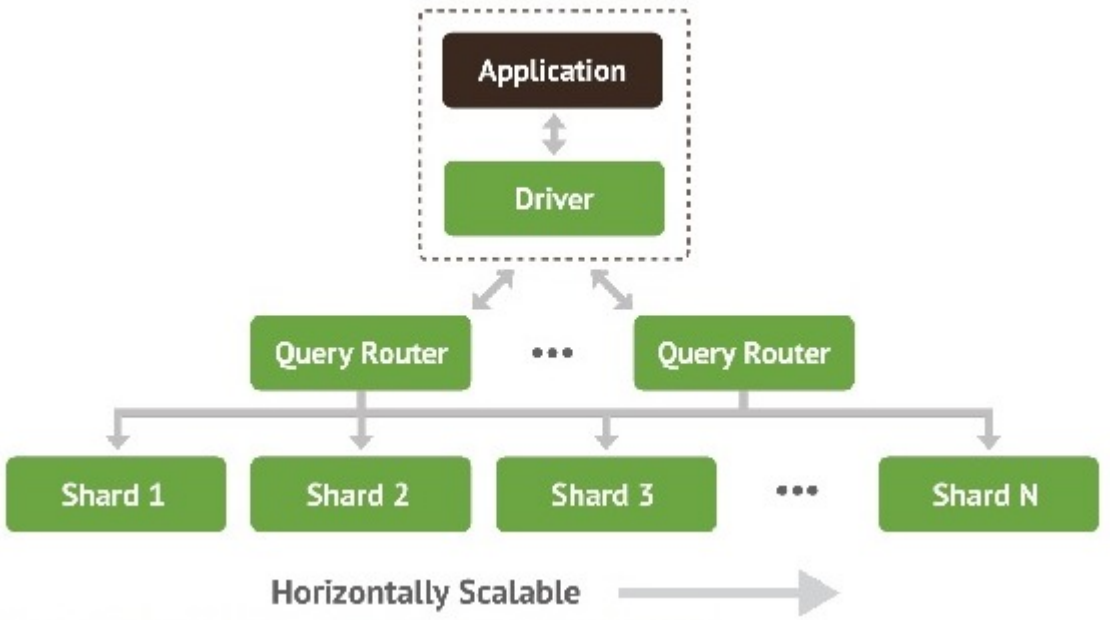


Figura 1.4. L'infrastruttura di MongoDB

I Query router fanno la ricerca in modo intelligente quindi, in base alla richiesta che si vuole fare, i query router cercheranno di capire dove sono i dati a cui si vuole accedere e faranno la richiesta solo agli shard interessati. Il vantaggio della scalabilità orizzontale è quello di poter aggiungere o togliere il numero di server (anche detto shard) a seconda delle necessità. Questo approccio viene detto scale out, mentre quello che si fa con i database relazionali è lo scale up, ovvero l'upgrade dei server esistenti per aumentarne le prestazioni.

Uno dei vantaggi di MongoDB è il bilanciamento automatico dei dati, poichè il carico viene distribuito automaticamente in modo da non appesantire troppo alcuni server in particolare. Quando si aggiunge un altro server, quindi, questo non rimane vuoto, poichè i dati vengono ridistribuiti in modo da cercare un carico uniformato tra tutti i server e in modo che il nuovo server entri in funzione fin da subito. Se è vuoto, infatti, sul server non si possono fare operazioni di lettura, ma solo di scrittura, quindi è lecito

considerare che questo nuovo server non verrà utilizzato molto di frequente e, quindi, l'upgrade tecnologico non sarà stato molto efficace. Cosa molto importante è che il bilanciamento automatico dei dati viene fatto senza fermare il servizio principale.

1.3.3. Disponibilità del servizio e affidabilità

MongoDB garantisce l'alta affidabilità, ovvero assicura la disponibilità del servizio durante varie tipologie di failure. Per quanto riguarda il disaster recovery (che si verifica ad esempio quando viene aggiunto un nuovo shard), lo sviluppatore ha a disposizione due parametri:

RTO (recovery time objective)
: Periodo di tempo massimo accettabile per eseguire lo switch a un altro datacenter

RPO (recovery point objective)
: Massima quantità di dati che è possibile perdere durante l'operazione di disaster recovery

In base al valore di questi parametri, MongoDB si comporterà di conseguenza, scegliendo che priorità dare all'operazione di disaster recovery rispetto al servizio normale. La manutenzione, invece, che può essere rappresentata ad esempio da upgrade di hardware o software, può essere ordinaria o straordinaria e viene fatta senza interrompere il servizio.

L'affidabilità viene garantita attraverso il replica set, che si compone di un primary e più copie del primary, dette secondary, che vengono aggiornate mediante una replicazione asincrona. Normalmente l'applicazione legge e scrive dal primary e i secondary vengono usati per il backup e per arbitration, ovvero quel meccanismo attraverso il quale, nel caso in cui alcuni server subiscono danni (failure), MongoDB automaticamente eleggerà un nuovo primary tra i secondary sopravvissuti. Siccome l'elezione prevede che esista almeno il 50%+1 dei sopravvissuti, il numero dei server deve essere dispari quindi, se si vogliono usare meccanismi di backup, bisogna usare almeno 3 server. È anche possibile usare un server solo come arbitration, tenendolo vuoto.

La disponibilità del servizio è garantita anche quando vi sono le operazioni di manutenzione a cui si è accennato in precedenza, poichè in queste occasioni, mettendo il primary come secondary per poi spegnerlo, il servizio può essere garantito ugualmente, in quanto le operazioni di lettura e scrittura possono essere fatte sul nuovo primary temporaneo. Al termine delle operazioni di manutenzione, è possibile ripristinare il primary originario, aggiornando tutte le modifiche fatte.

La replicazione asincrona comporta che, in caso di failure del primary, alcuni dati potrebbero andare perduti. Questo accade quando il failure avviene dopo che questi ultimi vengono scritti e prima che si completi la replica su almeno un secondary. Se lo sviluppatore ritiene che certi dati della sua applicazione sono così importanti da non poter accettare questa eventualità, quando si fa una scrittura sul primary si può specificare su quanti server si vuole che questa scrittura sia fatta prima che l'applicazione riceva la conferma di operazione effettuata con successo. Ad esempio si può indicare che la conferma venga data quando la maggioranza dei nodi ha scritto il dato e in questo modo si è sicuri che in caso di failure del primary i dati risiederanno almeno sulla maggior parte dei secondary. La replicazione asincrona comporta il vantaggio che i secondary possono essere adattati a vari tipi di utilizzo:

- si possono usare semplicemente come backup per garantire l'alta disponibilità del servizio;
- possono essere distribuiti più data center per gestire il disaster recovery;
- possono essere usati per la gestione della manutenzione;
- possono essere distribuiti sul territorio, creando un meccanismo di caching per diminuire la latenza. Questo tipo di utilizzo è importante soprattutto per i servizi che sono principalmente in lettura e hanno bassa scrittura. Mettendo i secondary più vicini (in termini di network) agli utenti, infatti, questi ultimi leggeranno i dati dal secondary a loro più vicino, quindi riceveranno una risposta in maniera più rapida e il primary sarà meno carico.

Ad aumentare ancora di più la disponibilità del servizio si ha che il sistema operativo di MongoDB, di default, è di tipo Journaling (disabilitabile a richiesta), quindi anche in caso di crash della macchina non si perde nessun dato anche se non si è fatto un flush sui data file, perché vi è il file log Journaling del file system che permette di fare la recovery dei dati.

1.3.4. Performance

Si è già detto che MongoDB, rispetto ai database relazionali è caratterizzato da un aumento delle performance. In particolare, questo miglioramento è dato dal fatto che mettere insieme nello stesso documento dati che, in un modello SQL, apparterrebbero a tabelle diverse elimina la necessità di fare join tra tabelle e andare a cercare i dati sparpagliati all'interno del disco (disk seek). I dati che appartengono a uno stesso documento, infatti, saranno scritti in modo contiguo sul disco, proprio per il fatto di appartenere allo stesso documento, quindi le operazioni di lettura e scrittura, nel

modello NoSQL, diventano operazioni sequenziali. Per ovviare alla mancanza della possibilità dell'SQL di riunire i dati che si erano precedentemente spezzati in più tabelle mediante una join, in MongoDB vi è il meccanismo dell'embedding, che consiste nell'aggiungere sottodocumenti nei documenti master.

1.3.5. Durabilità dell'informazione

La Durabilità rappresenta la "D" delle "ACID", le proprietà logiche che devono avere le transazioni nel database, e rappresenta la garanzia che i dati scritti resteranno nel database in modo permanente. Più i dati sono durabili, più i tempi di risposta sono lunghi, quindi bisogna trovare un compromesso tra durabilità e tempi di risposta in base all'importanza dei dati che si stanno salvando. La caratteristica della durabilità è pienamente personalizzabile in MongoDB, poichè il livello di durabilità può essere impostato mediante diversi metodi di scrittura:

Unacknowledged
: È il metodo più veloce di scrivere dati su MongoDB e ed è basato sul modello asincrono, in quanto consiste solo nel mandare la scrittura al demone mongod, senza aspettare la sua risposta. Dopo che si scrive, quindi, non si sa se la scrittura è andata a buon fine o meno. Questa modalità è utile quando i dati sono veramente poco critici, ad esempio nel caso di logging aggressivo in cui se si perde qualche attività non è rilevante oppure nell'operazione di raccolta di statistiche.

Acknowledged
: È il metodo abilitato di default e consiste nell'aspettare la risposta dopo la scrittura, che viene mandata solo dopo che il dato è stato scritto in memoria. Un'eventuale risposta positiva permette al driver di accorgersi di molti errori, come quelli di network o dovuti a chiavi duplicate, ma non garantisce che i dati siano stati salvati su disco. Infatti, se sul server di MongoDB si verifica un crash dopo aver mandato l'ack ma prima di avere trasferito i dati dalla memoria al disco, questi ultimi vengono persi, senza che chi ha effettuato la scrittura se ne accorga.

Journalled
: È il metodo che assicura che i dati siano stati scritti su disco in quanto, in questa modalità, il server risponde positivamente solo dopo che l'operazione viene fatta sul journal. MongoDB, infatti, utilizza il journaling del file system di default e quindi, anche se si verifica un crash nel server prima che il dato possa essere scritto su disco, se si ha ricevuto una risposta affermativa da MongoDB si ha la sicurezza che il dato si trova almeno sul journal e che quindi, al prossimo riavvio, i dati verranno recuperati dal journal e scritti definitivamente sul disco. All'interno dello statement

vi è il parametro j che sta per journaling che, quando vale true, attiva il journaling mentre, quando vale false, lo disattiva. Questo parametro j si può passare statement per statement, quindi si può abilitare o disabilitare il journaling a seconda del tipo di dato da trattare. Per quanto riguarda i tempi di risposta, di default il contenuto del journal viene scritto su disco ogni 100 ms, mentre se si ha una write con l'opzione del journaling attivata, i 100 ms si riducono a 30, per abbassare i tempi di risposta. Da questo dato si deduce che se si specifica j:true per ogni write, il massimo throughput sarà di 1000/30 = 33.3 write al secondo.

Fsynced

In questa modalità, MongoDB da una risposta positiva alla write solo dopo che il dato è stato scritto sul disco.

Replica acknowledged

È un metodo che, a differenza degli altri, si applica a più server. All'interno del replica set, infatti, si può stabilire su quanti server i dati hanno bisogno di essere scritti prima di considerare la scrittura andata a buon fine e, se il numero di server è superiore a uno, vuol dire che si sta utilizzando il metodo di scrittura "Replica acknowledged". In questa modalità, infatti, non basterà scrivere i dati sul primary, ma la risposta positiva sarà data solo dopo aver scritto i dati anche su un certo numero di secondary decisi dallo sviluppatore. Con gli altri metodi, le replicazioni dal primary ai secondary avvengono in maniera asincrona, in modo tale che il primary non risenta troppo di avere molti secondary, mentre con questo metodo bisogna aspettare anche la scrittura su un certo numero di secondary, perciò i tempi di risposta aumentano di molto. Tuttavia, utilizzare questo metodo in certi contesti è indispensabile. In caso di failure del primary, infatti, se vi erano dei dati che erano sul primary e che non erano ancora stati replicati sui secondary, questi dati andranno persi con gli altri metodi, mentre con il Replica acknowledged si ha la sicurezza che questi siano già stati scritti, quindi si risolve il problema della replicazione asincrona. Il parametro da passare in questo caso sarà getLastError w:n, dove n può essere:

- il numero di nodi su cui voglio che il dato venga scritto prima di avere risposta;
- "majority", indicando l'intenzione di aspettare che il dato sia scritto sulla maggioranza dei secondary, in modo tale che nel caso in cui si aggiungessero altri nodi non si debba manualmente aggiornare un numero statico;
- "all" per indicare che il dato deve essere scritto su tutti i secondary. Nei casi di distribuzione geografica spinta questo comportamento può avere un impatto notevole sulla velocità e la latenza di risposta [10].

[10] fonte http://blog.mongodirector.com/understanding-durability-write-safety-in-MongoDB/

1.3.6. Indicizzazione

Gli indici sono il più grande fattore di miglioramento delle performance di un DB e, ovviamente, MongoDB non ne è sprovvisto. Essi, in MongoDB, sono gestiti mediante B-Tree e non binary tree, in quanto questi ultimi hanno il difetto di essere troppo profondi.

Nella struttura B-Tree ogni nodo dell'albero può contenere molteplici valori, che saranno sempre minori dei valori contenuti nel nodo successivo e maggiori di quelli del nodo precedente. Proprio per la proprietà dei B-tree di avere un'altezza dell'albero ridotta, viene fatto un numero di confronti molto basso, in modo da ottimizzare al massimo le operazioni con gli indici. In particolare, l'aggiunta di un nuovo elemento avviene sempre con una complessità computazionale pari a log(n).

Esistono anche gli indici composti, ovvero indici che usano più di un campo, anche tra tipi di campi diversi, come un campo normale e un array. A partire dagli indici composti, MongoDB crea anche degli indici parziali, quindi se si fa una query su un solo campo su cui era stato creato un indice composto verrà sfruttato un indice parziale, creato a partire da quello composto, per velocizzare l'operazione.

Si è già detto che in MongoDB i documenti possono avere struttura diversa all'interno della stessa collection. Nel caso in cui si abbiano dei documenti con un campo specifico con cardinalità molto bassa rispetto al numero dei documenti, creando un indice su questo campo il comportamento di default è di aggiungere comunque tutti i documenti all'interno dell'indice e, in quelli in cui il campo non esiste, esso viene aggiunto e messo a NULL. Questo porta a una dimensione dell'indice notevole o comunque non necessaria. Per questo motivo, in MongoDB esiste lo sparse index, che si può abilitare mediante l'opzione "sparse: true", all'interno dell'istruzione per creare l'indice. Esso fa in modo che nell'indice vi siano solo i documenti che hanno quel campo, quindi è particolarmente consigliato per tutti i campi con cardinalità molto bassa, perché porta a una notevole riduzione dell'indice.

Ovviamente, come per i database relazionali, si possono creare anche degli indici unici tramite il parametro unique: true, in modo tale da garantire consistenza alla base di dati. Inoltre, esistono molti indici particolari, che permettono di fare operazioni specifiche sui campi, come ad esempio:

Indici geografici

Permettono di definire le coordinate e il tipo di oggetto, come ad esempio punto, cerchio o poligono, che può essere specificato mediante un array di coppie di coordinate. Per creare un indice di questo tipo basterà specificare la dicitura

"2dsphere" come tipo dell'indice al momento della sua definizione. Questo indice usa la notazione GeoJSON e vi sono degli operatori speciali di MongoDB che rendono particolarmente facili le operazioni con questi tipi di campi, come ad esempio "$near", che trova tutti i documenti vicini a delle certe coordinate, "$maxDistance", che serve a specificare la massima distanza rispetto a un punto e "$within", che serve a trovare tutti i documenti all'interno di un poligono.

indici testuali

Permettono di effettuare lo stemming, ovvero il processo di riduzione della forma flessa di una parola alla sua forma radice, o dare un peso per i campi indicizzati, quindi ad esempio si può mettere un peso maggiore al titolo di un articolo rispetto al corpo. Per creare un indice di questo tipo basterà specificare la dicitura "text" come tipo dell'indice al momento della sua definizione.

1.3.7. Ordinamento

Quando si crea un indice, si può specificare se ordinare gli elementi dell'indice in ordine ascendente o discendente, in modo tale che il risultato della query sarà già ordinato. Questa istruzione, inoltre, funziona anche sui campi di tipo array o su campi di sottodocumenti.

L'ordinamento ha poca importanza sugli indici singoli, perché con i B-tree posso sia leggere dal basso che dall'alto quindi, anche se l'indice su un campo è in un verso, presentare i dati in ordine inverso ha lo stesso costo. Il discorso cambia negli indici composti, dove bisogna specificare l'ordine per ogni campo e, in questa situazione, ordinare in maniera differente ha un costo notevole, in quanto bisogna eseguire delle operazioni di sort che possono essere anche molto pesanti. Si possono, dunque, creare indici non solo quando bisogna ottimizzare le operazioni su un determinato campo, ma anche per evitare operazioni di ordinamento frequenti.

Quando si fanno le query solo su campi di cui esiste un indice, per ottenere il risultato non c'è bisogno di leggere i documenti, ma basta ritornare i valori che sono all'interno dell'indice per risolvere la query, quindi questa sarà molto più rapida rispetto a una query eseguita su campi di cui non esiste un indice. Questi tipi di query si chiamano "query su covered index" o "query index only" e permettono di aumentare a dismisura le performance del sistema perché, di fatto, evitano di eseguire letture su disco.

Capitolo 2. Scelta dell'implementazione del CoAP

In questo capitolo si riporta tutto il percorso che ha portato alla scelta dell'implementazione del CoAP da utilizzare, analizzando svantaggi, vantaggi e codici creati per ogni implementazione studiata.

Inizialmente si è cercato un implementazione del CoAP in Java, ma in seguito a vari problemi incontrati durante lo sviluppo, è stato necessario cambiare direzione e orientarsi verso il CoAPthon, un'implementazione basata su python (come si evince dal nome), tramite la quale si è riusciti a raggiungere gli scopi prefissati.

2.1. jCoAP

La prima implementazione java del CoAP che si è testata è stata jCoAP, un' implementazione stabile ed abbastanza semplice da utilizzare. Per cominciare si è esaminato il codice del jCoAP [1] e, prendendo spunto da alcuni programmi di esempio(creati da Christian Lerche), si è riusciti ad ottenere una corretta comunicazione che rispecchiasse il paradigma client-server.

In prima fase, infatti, piuttosto che preoccuparsi del progetto nella sua interezza, ci si è concentrati sulla realizzazione di un client e un server che comunicassero con il protocollo CoAP.

A titolo di esempio si è considerata la situazione in cui ci fosse un sensore di temperatura lato client che trasmette valori ad un server.

2.1.1. Server

Di seguito si riportano i 3 file che costituiscono il server.

MyCoapServer

```
import java.nio.ByteBuffer;
import org.ws4d.coap.Constants;
import org.ws4d.coap.connection.BasicCoapChannelManager;
import org.ws4d.coap.interfaces.CoapChannelManager;
import org.ws4d.coap.interfaces.CoapMessage;
```

[1] http://code.google.com/p/jcoap/

```
import org.ws4d.coap.interfaces.CoapRequest;
import org.ws4d.coap.interfaces.CoapServer;
import org.ws4d.coap.interfaces.CoapServerChannel;
import org.ws4d.coap.messages.CoapMediaType;
import org.ws4d.coap.messages.CoapResponseCode;

// Resta in ascolto sulla porta PORT e ogni volta che gli arriva un messaggio avvia un thread
 che si occupa di gestire i dati(un double) contenuti in esso

public class MyCoapServer implements CoapServer {
    private final int PORT = Constants.COAP_DEFAULT_PORT;
    static int counter = 0;

    public void work(MyCoapServer server) {
        System.out.println("Start CoAP Server on port " + PORT);
        CoapChannelManager channelManager = BasicCoapChannelManager.getInstance();
        channelManager.createServerListener(server, PORT);
    }

 @Override
 public CoapServer onAccept(CoapRequest request) {
  System.out.println("Accept connection...");
  return this;
 }

 @Override
 public void onRequest(CoapServerChannel channel, CoapRequest request) {
  System.out.println("Received message: " + request.toString());
  GestoreDato gd= new GestoreDato(ByteBuffer.wrap(request.getPayload()).getDouble()); //creo
l'oggetto gd che si occuperà di fare le operazioni necessarie sul dato
  gd.start();
  CoapMessage response = channel.createResponse(request, CoapResponseCode.Content_205);
  response.setContentType(CoapMediaType.text_plain);
  channel.sendMessage(response);
 }

 @Override
 public void onSeparateResponseFailed(CoapServerChannel channel) {
  System.out.println("Separate response transmission failed.");

 }
}
```

GestoreDato

```
public class GestoreDato extends Thread {

 private double temperatura;

 GestoreDato(double temperatura){
  setTemperatura(temperatura);
 }

    void setTemperatura(double temperatura){
        this.temperatura=temperatura;
```

```
    }

 double getTemperatura(){
  return temperatura;
 }

 @Override
 public void run(){
  System.out.println(getTemperatura());
 }
}
```

DemoServer

```
//programma che lancia il server
public class DemoServer {

 public static void main(String[] args) {
  MyCoapServer server=new MyCoapServer();
  server.work(server);
 }

}
```

2.1.2. Client

Si riportano ora i 3 programmi del client

MyCoapClient

```
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.nio.ByteBuffer;

import org.ws4d.coap.Constants;
import org.ws4d.coap.connection.BasicCoapChannelManager;
import org.ws4d.coap.interfaces.CoapChannelManager;
import org.ws4d.coap.interfaces.CoapClient;
import org.ws4d.coap.interfaces.CoapClientChannel;
import org.ws4d.coap.interfaces.CoapRequest;
import org.ws4d.coap.interfaces.CoapResponse;
import org.ws4d.coap.messages.CoapRequestCode;



// Manda al server un double rilevato da Sensore s ogni SLEEP secondi

public class MyCoapClient implements CoapClient {

 private final int SLEEP=5000; //millisecondi che bisogna attendere tra una lettura del
 sensore e un'altra
    private String server_address;
```

```
    private final int PORT = Constants.COAP_DEFAULT_PORT;
    static int counter = 0;
    CoapChannelManager channelManager;
    CoapClientChannel clientChannel;

    void setServer_address(String address){
     server_address=address;
    }

    public void work( MyCoapClient client ) {
        System.out.println("Start CoAP Client");
        client.channelManager = BasicCoapChannelManager.getInstance();
        client.runTestClient();
    }

    public void runTestClient(){
     Sensore s=new Sensore();
 byte[] bytes = new byte[8]; //bytes da spedire al server, sono 8 perché devo mandare un
double
     try {
  while(true){
   clientChannel = channelManager.connect(this, InetAddress.getByName(server_address),
PORT);
   CoapRequest coapRequest = clientChannel.createRequest(true, CoapRequestCode.POST);
      ByteBuffer.wrap(bytes).putDouble(s.rilevaTemperatura()); //trasformo il double che
rappresenta la temperatura in byte[]
   coapRequest.setPayload(bytes);
   clientChannel.sendMessage(coapRequest);
   System.out.println("Message sent");
   Thread.sleep(SLEEP);
  }

 } catch (UnknownHostException e) {
  e.printStackTrace();
 } catch (InterruptedException e) {
  e.printStackTrace();
 }
   }

 @Override
 public void onConnectionFailed(CoapClientChannel channel, boolean notReachable, boolean
resetByServer) {
  System.out.println("Connection Failed");
 }

 @Override
 public void onResponse(CoapClientChannel channel, CoapResponse response) {
  System.out.println("Received response");
  System.out.println(response.toString());
 }
}
```

DemoClient

```
public class DemoClient {
```

```
  public static void main(String[] args) {
   MyCoapClient client=new MyCoapClient();
   if (args.length != 0)
    client.setServer_address(args[0]);
   else
    client.setServer_address("127.0.0.1");
   client.work(client);
  }

}
```

Sensore

```
public class Sensore {
 public double rilevaTemperatura(){
  return Math.random()*100-50; //ritorna un double tra -50 e 50
 }
}
```

2.1.3. Motivazioni dell'abbandono

Una volta ottenuto il risultato precedente si è provato ad estendere il lavoro svolto, cercando di implementare uno dei requisiti fondamentali del progetto: il multicast. Dopo svariati tentativi ci si è resi conto che il jCoAP (almeno fino all'ultima versione disponibile al momento in cui si scrive, che è quella del 12 giugno 2012) non implementa il multicast. Nel codice, infatti, non era presente alcuna istruzione che permettesse l'istanza di socket di tipo multicast o meccasmi di join agli indirizzi IP riservati a questo fine.

2.2. Californium

Come conseguenza al fatto che il jCoAP non soddisfava i requisiti del progetto si è stati obbligati a cercare una nuova implementazione del protocollo, così si è studiato il Californium [2], l'implementazione più completa(e anche più complessa) del CoAP in java.

Anche in questo caso ci è posti subito l'obiettivo di realizzare client e server, cosa che si è rivelata più complicata del previsto, vista la grandezza del core del californium.

Successivamente, si è cercato di simulare una parte del comportamento del nostro sistema, non includendo il multicast. Si è creato(basandosi su codici di esempio),

[2] http://people.inf.ethz.ch/mkovatsc/californium.php

dunque, l'interruttore, il server e la lampadina. L'interruttore manda segnali al server e quest'ultimo li rimanda alla lampadina. Di seguito ne riportiamo il codice.

2.2.1. Interruttore

Client

```
public class Interruttore {

    public static void main(String[] args) {
        Scanner interruttore= new Scanner(System.in);
        String segnale;
        while(true){
         Request post = new Request(Code.PUT);
         post.setURI("coap://127.0.0.1:4000/Listener");
         segnale=interruttore.nextLine();
         switch(segnale){
         case "0":
          post.setPayload("0");
          break;
         case "1":
          post.setPayload("1");
          break;
         default:
          System.out.println("immettere 1 o 0");
          continue;
         }
         post.getOptions()
            .setContentFormat(MediaTypeRegistry.TEXT_PLAIN)
            .setAccept(MediaTypeRegistry.TEXT_PLAIN)
            .setIfNoneMatch(true);
            try {
       String response = post.send().waitForResponse().getPayloadString();
       System.out.println(response);
      } catch (InterruptedException e) {
       e.printStackTrace();
      }

        }

    }

}
```

2.2.2. Server

Server

```
package step1;

import static org.eclipse.californium.core.coap.CoAP.ResponseCode.BAD_REQUEST;
```

```
import static org.eclipse.californium.core.coap.CoAP.ResponseCode.CHANGED;

import org.eclipse.californium.core.CoapResource;
import org.eclipse.californium.core.CoapServer;
import org.eclipse.californium.core.coap.MediaTypeRegistry;
import org.eclipse.californium.core.coap.Request;
import org.eclipse.californium.core.coap.CoAP.Code;
import org.eclipse.californium.core.server.resources.CoapExchange;

public class Server {

 static Selector selector;

 public static void main(String[] args) {
        // binds on UDP port 5683
        CoapServer server = new CoapServer(4000);
        server.add(new HelloResource());
        server.start();
    }

    public static class HelloResource extends CoapResource {
        public HelloResource() {
            // resource identifier
            super("Listener");
            // set display name
            getAttributes().setTitle("Hello-World Resource");
        }

        @Override
        public void handleGET(CoapExchange exchange) {
            exchange.respond("Respond");
        }

        @Override
        public void handlePUT(CoapExchange exchange) {
            byte[] payload = exchange.getRequestPayload();
            String destination;
            try {
             System.out.println("Richiesta PUT, SERVER");
                String value = new String(payload, "UTF-8");
                System.out.println(value);
                exchange.respond(CHANGED, value);
                //destination=selector.groupSelect();
                Request post = new Request(Code.PUT);
                post.setURI("coap://127.0.0.1:5683");
                post.setMulticast(true);
                post.setPayload(value);
                post.getOptions()
                .setContentFormat(MediaTypeRegistry.TEXT_PLAIN)
                .setAccept(MediaTypeRegistry.TEXT_PLAIN)
                .setIfNoneMatch(true);
                try {
           String response = post.send().waitForResponse().getPayloadString();
          } catch (InterruptedException e) {
           e.printStackTrace();
          }
            } catch (Exception e) {
```

```
                e.printStackTrace();
                exchange.respond(BAD_REQUEST, "Invalid String");
            }
        }
    }

}
```

Selector

```
package step1;

import java.net.InetAddress;
import java.net.UnknownHostException;

//decide a quale sottogruppo mandare l'input
public class Selector {

 String group;

 String groupSelect(){
  setGroup("coap://localhost/Ricevitore");
  return group;
 }

 void setGroup(String s){
  group=s;
 }
}
```

2.2.3. Lampadina

Server

```
package step1;

import static org.eclipse.californium.core.coap.CoAP.ResponseCode.BAD_REQUEST;
import static org.eclipse.californium.core.coap.CoAP.ResponseCode.CHANGED;

import java.io.IOException;
import java.net.InetSocketAddress;

import org.eclipse.californium.core.CoapResource;
import org.eclipse.californium.core.CoapServer;
import org.eclipse.californium.core.network.CoAPEndpoint;
import org.eclipse.californium.core.network.Endpoint;
import org.eclipse.californium.core.network.config.NetworkConfig;
import org.eclipse.californium.core.server.resources.CoapExchange;

public class LServer {
```

```
public static void main(String[] args) throws IOException {
       // binds on UDP port 5683
 //InetAddress addr = InetAddress.getByName("224.0.1.187");

 //MulticastSocket s=new MulticastSocket(5683);
 //s.joinGroup(addr);
 InetSocketAddress a=new InetSocketAddress("127.0.0.1",5683);
       CoapServer server = new CoapServer();
       Endpoint endpoint = new CoAPEndpoint(a, NetworkConfig.getStandard());
       server.addEndpoint(endpoint);
       server.add(new HelloResource());
       server.start();
   }

   public static class HelloResource extends CoapResource {
       public HelloResource() {
           // resource identifier
           super("Ricevitore");
           // set display name
           getAttributes().setTitle("Hello-World Resource");
       }

       @Override
       public void handleGET(CoapExchange exchange) {
           exchange.respond("Respond");
       }

       @Override
       public void handlePUT(CoapExchange exchange) {
           byte[] payload = exchange.getRequestPayload();
           try {
            System.out.println("Richiesta PUT, LAMPADINA");
               String value = new String(payload, "UTF-8");
               System.out.println(value);
               exchange.respond(CHANGED, value);
           } catch (Exception e) {
               e.printStackTrace();
               exchange.respond(BAD_REQUEST, "Invalid String");
           }
       }
   }

}
```

2.2.4. Motivazioni dell'abbandono

Una volta arrivati a questo punto si è provato, come in precedenza, ad aggiungere il multicast, senza però alcun successo. Dopo un'attenta analisi del codice e svariati tentativi, infatti, abbiamo capito che neanche nel californium(almeno alla versione disponibile al momento in cui si scrive, ovvero la release 1.0.0-M2 [3]) è stata

[3] http://github.com/eclipse/californium/tree/1.0.0-M2

implementata questa feature. Siamo arrivati alla conclusione quindi che non esistono implementazioni del CoAP in Java che supportano il multicast.

2.3. CoAPthon

A questo punto ci si è informati su tutte le implementazioni del CoAP, non solo su quelle realizzate in Java, e alla fine si è scelto di studiare il CoaPthon, perché tra le caratteristiche presentava in bella vista il supporto al multicast.

Una difficoltà che si è avuta inizialmente è stata quella di imparare un nuovo linguaggio di programmazione in quanto questa implementazione è realizzata in python, un linguaggio che non era stato studiato fino ad ora dall'autore della tesi. Fortunatamente esso si presenta fin da subito abbastanza user-friendly e non si riscontrano particolari ostacoli nell'impararne gli aspetti fondamentali.

2.3.1. CoAPclient

All'interno del codice del CoAPthon subito si nota il file coapclient.py, un programma che quando viene eseguito permette di mandare richieste CoAP inserendo da terminale le informazioni necessarie, come l'operazione da compiere(GET, PUT, ecc.), il percorso della richiesta e il payload. Se ne apprezzano dunque l'intuitività e il meccanismo di black-box e si cerca un modo per adattarlo agli scopi del progetto.

Semplicemente vengono modificati gli argomenti presi in ingresso dal terminale, aggiungendo l'indirizzo del destinatario mediante l'opzione "-A". In questo modo, per mandare una richiesta all'interno di un programma basterà chiedere al sistema operativo di lanciare una stringa sulla linea di comando. Avendo a disposizione questo strumento si è creato client e server partendo dalla base dei codici di esempio creati da Giacomo Tanganelli(lo sviluppatore principale del CoAPthon).

2.3.2. Errori riscontrati e risoluzione

Una volta installate tutte le librerie necessarie si è riscontrato un errore all'interno del codice del file client/coap_protocol.py:

```
logfile = DailyLogFile("CoAPthon_client.log", home + "/.coapthon/")
```

e del file server/coap_protocol.py:

```
logfile = DailyLogFile("CoAPthon_server.log", home + "/.coapthon/")
```

Queste istruzioni non permettono il corretto funzionamento su sistemi operativi windows(il sistema sul quale si stava lavorando in quel periodo), in quanto il percorso indicato non è lecito perché era stato pensato per linux. Pertanto si è modificato le due righe nel seguente modo:

```
logfile = DailyLogFile("CoAPthon_client.log", "/coapthon/")
```

e

```
logfile = DailyLogFile("CoAPthon_client.log", "/coapthon/")
```

Come si può notare è stato eliminato il punto dal percorso. In questo modo la cartella creata non sarà nascosta, ma almeno il programma sarà multipiattaforma. Una volta risolti questi problemi si è cercato di inserire lo scambio di richieste in multicast all'interno del programma. Per farlo abbiamo utilizzato le seguenti istruzioni, che servono ad abilitare il server all'ascolto di richieste multicast.

```
server = CoAPServer(group, 5683)
reactor.listenMulticast(5683, server, listenMultiple=True)
```

dove il costruttore di CoAPServer è:

```
def __init__(self, host, port, multicast=False):
```

e "group" indica l'indirizzo IP multicast al quale il server presta ascolto.

Dopo varie prove si è riusciti a legare diversi server ad un unico indirizzo IP multicast e a mandare richieste di multicast dal client al gruppo di server. Questi ultimi ricevono il messaggio del client e rispondo correttamente, mentre il client nonostante riceva tutte le risposte alza un'eccezione:

```
("Can't stop reactor that isn't running.") twisted.internet.error.ReactorNotRunning: Can't
 stop reactor that isn't running.)
```

Dopo svariati tentativi per cercare di risolvere questo problema si è deciso di contattare tramite e-mail il programmatore del CoAPthon: Giacomo Tanganelli. Costui si è rivelata fin da subito una persona disponibilissima e ha spiegato che con più server in esecuzione, il client solleva questa eccezione in ricezione perché si aspetta solo un ack, in quanto manda la richiesta ad un solo indirizzo IP, ma siccome l'indirizzo è di tipo multicast il messaggio verrà ricevuto da più macchine, che risponderanno ognuna con il proprio ack. Il client, perciò, si ritroverà risposte in eccesso e per questo lancerà quella

eccezione. Dai test eseguiti, comunque, non risulta che questo problema intacchi la stabilità e il corretto funzionamento del sistema.

Al momento della scrittura del programma l'ultima versione disponibile del coapthon era il coapthon 2, ma successivamente è uscita la versione 3, in cui questo errore potrebbe essere stato risolto.

2.3.3. Motivazioni della scelta

Concludendo, dunque, si è deciso di adottare il CoAPthon come implementazione del CoAP all'interno del progetto per i seguenti motivi:

- in seguito a vari test effettuati si è dimostrato molto stabile;
- il coapclient ha una struttura black-box che rende semplicissimo inviare un messaggio in CoAP, cosa non riscontrata in nessun'altra delle implementazioni provate fino ad ora;
- è presente il supporto al multicast;
- il progetto è open-source e ben mantenuto;

Capitolo 3. Il sistema sviluppato

Dopo aver visto le basi teoriche delle tecnologie su cui si fonda il progetto, si illustra di seguito il sistema sviluppato, ponendo particolare attenzione allo scenario di riferimento, ai vincoli che esso impone, agli obiettivi del progetto, alla soluzione, ai requisiti, alle funzionalità e ad alcuni casi d'uso.

3.1. Scenario di riferimento

Lo scenario di riferimento è quello di un insieme di reti vincolate, contenenti i dispositivi, che si interfacciano con la rete internet standard e gli utenti che vogliono accedere ai dispositivi, come mostrato nella figura di seguito.

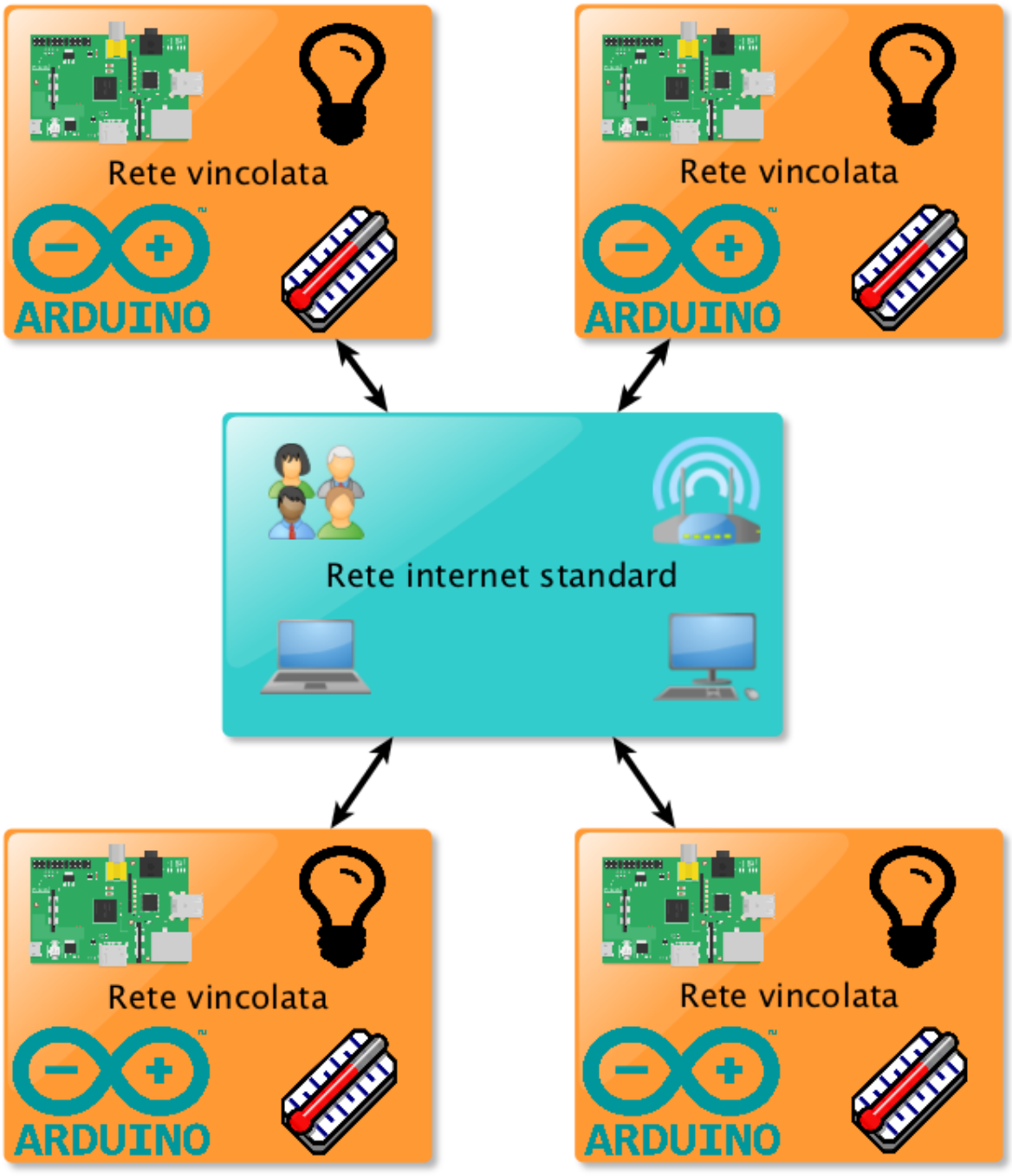


Figura 3.1. Rappresentazione dello scenario

I dispositivi

Sono sensori e attuatori collegati a un sistema embedded, che gli permette di comunicare con il mondo esterno. Si considerano tutti come dei nodi vincolati.

Le reti vincolate

Sono rappresentate da qualunque ambiente in cui vi siano dei nodi vincolati.

Come si è visto nella sezione 1.1.1, infatti, l'uso dei nodi vincolati spesso porta anche a vincoli sulle reti stesse, quindi, anche se una rete non ha vincoli, ma al

suo interno contiene nodi vincolati, essa può essere considerata ugualmente come una rete vincolata.
Alcuni esempi potrebbero essere:

- smart home con le luci che si accendono in base all'ora del giorno e alla locazione dell'interruttore attivato;
- università in cui si registrano gli ingressi e le uscite degli studenti dai laboratori e le cui porte si aprono solo se gli studenti sono abilitati a passare;
- terreni agricoli con dei sensori che misurano la presenza di determinati insetticidi.

Gli utenti
Persone che, mediante un client, vogliono accedere ai dispositivi attraverso la rete Internet o una rete intranet.

3.1.1. Vincoli progettuali

Visto lo scenario di riferimento, è possibile elencare i vari vincoli progettuali che esso impone:

- la presenza di nodi vincolati, per tutte le problematiche descritte nel capitolo 1.1.1, impone l'utilizzo del protocollo CoAP, poiché quest'ultimo ottimizza l'utilizzo di questi tipi di dispositivi;
- la presenza di più utenti che vogliono personalizzare il servizio, registrando i propri dispositivi o i propri servizi, porta all'esigenza di salvare le credenziali di accesso per ogni utente;
- Il vincolo precedente implica l'impiego di una comunicazione sicura, in quanto non è possibile far viaggiare su internet i dati personali degli utenti in chiaro. Inoltre non tutti gli utenti vogliono condividere le informazioni riguardanti i propri dispositivi, quindi deve essere possibile criptare anche queste ultime.

3.2. Obiettivi

Il progetto nasce per realizzare un sistema che permetta agli utenti di comandare gli attuatori mediante regole e servizi specificati da loro stessi e di leggere le misurazioni prodotte dai sensori. Tali obiettivi si traducono nei seguenti obiettivi tecnologici:

- utillizare CoAP come protocollo di comunicazione nelle reti vincolate;

- gestire l'aggregazione di più attuatori all'interno di gruppi multicast;
- far accedere gli utenti ai dispositivi mediante il protocollo HTTP;
- salvare i dati dell'applicazione all'interno di un database NoSQL, mantenendo i dati personali degli utenti in un database SQL;
- gestire la creazione di gruppi di utenti e dispositivi.

3.3. La soluzione

3.3.1. Schema

S'illustra di seguito lo schema completo della soluzione sviluppata, spiegando quali sono stati i componenti che a mano a mano sono stati aggiunti e per quale motivo.

Schema iniziale

Il primo elemento che si è pensato di aggiungere allo scenario è un server remoto che salvasse tutti i dati che servono all'applicazione, le misurazioni effettuate dai sensori e che fasse gestire il controllo degli attuatori in modo semplificato. Gli utenti, dunque, possono leggere le misurazioni dei sensori e controllare gli attuatori mediante un'interfaccia offerta da questo server.

I sensori e gli attuatori, tuttavia, non possono interfacciarsi direttamente con il server remoto, in quanto è necessaria un'altra entità che gestisca ogni rete vincolata. Si è scelto, dunque, di aggiungere in ogni rete vincolata un server di frontiera, che per semplicità chiameremo "server CoAP". Il nome deriva dal fatto che questo server comunica mediante il protocollo CoAP con i dispositivi della rete vincolata, in modo da limitare gli effetti dei vincoli.

Lo schema della soluzione sviluppata, dunque, prevede che i dispositivi comunichino con il server CoAP della propria rete vincolata e che quest'ultimo comunichi con il server remoto in modo tale da mandare le misure dei sensori e fornire un'interfaccia per comandare i gruppi multicast di attuatori presenti nella propria sottorete.

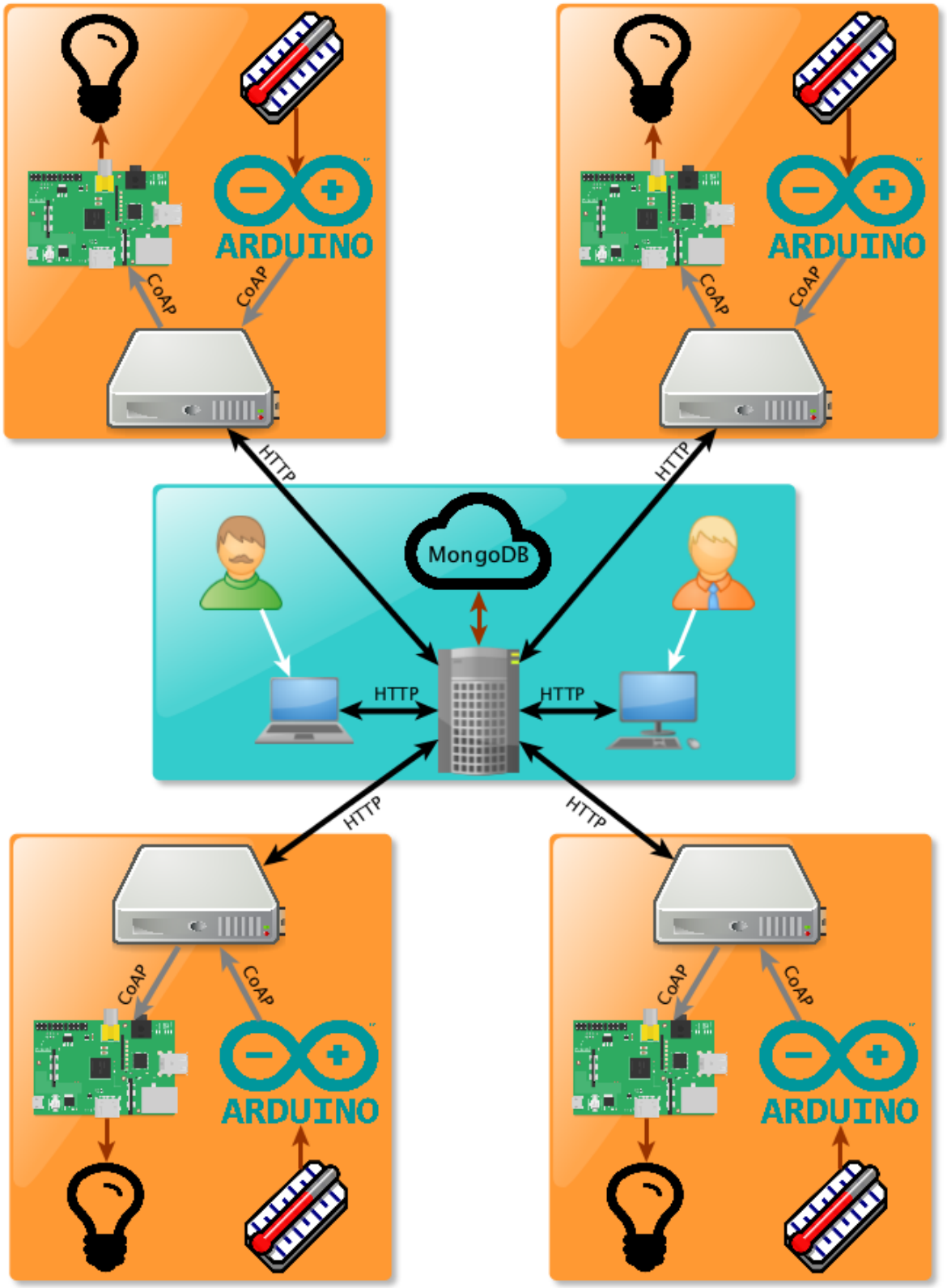


Figura 3.2. Schema della prima soluzione sviluppata

Schema completo

Lo schema appena illustrato ha il problema che il server remoto è troppo sovraccarico, in quanto deve ricevere sia le richieste dei server CoAP che quelle degli utenti.

Si è pensato dunque di creare due server distinti e indipendenti: uno che ricevesse le richieste degli utenti e gestisse le richieste in uscita verso i server CoAP e uno che, invece, ricevesse le richieste in entrata da questi ultimi. Facendo così, se si hanno a disposizione due macchine, ogni server può girare su una macchina diversa, consentendo di avere un bilanciamento del carico di lavoro.

Questa organizzazione logica separa la necessità degli utenti di controllare gli attuatori e, quindi, di far partire richieste verso i server CoAP dalla gestione dei dati mandati dai sensori che vanno opportunamente processati ed immagazzinati, cioè le richieste in entrata dei server CoAP.

Lo schema completo del sistema, dunque, è il seguente.

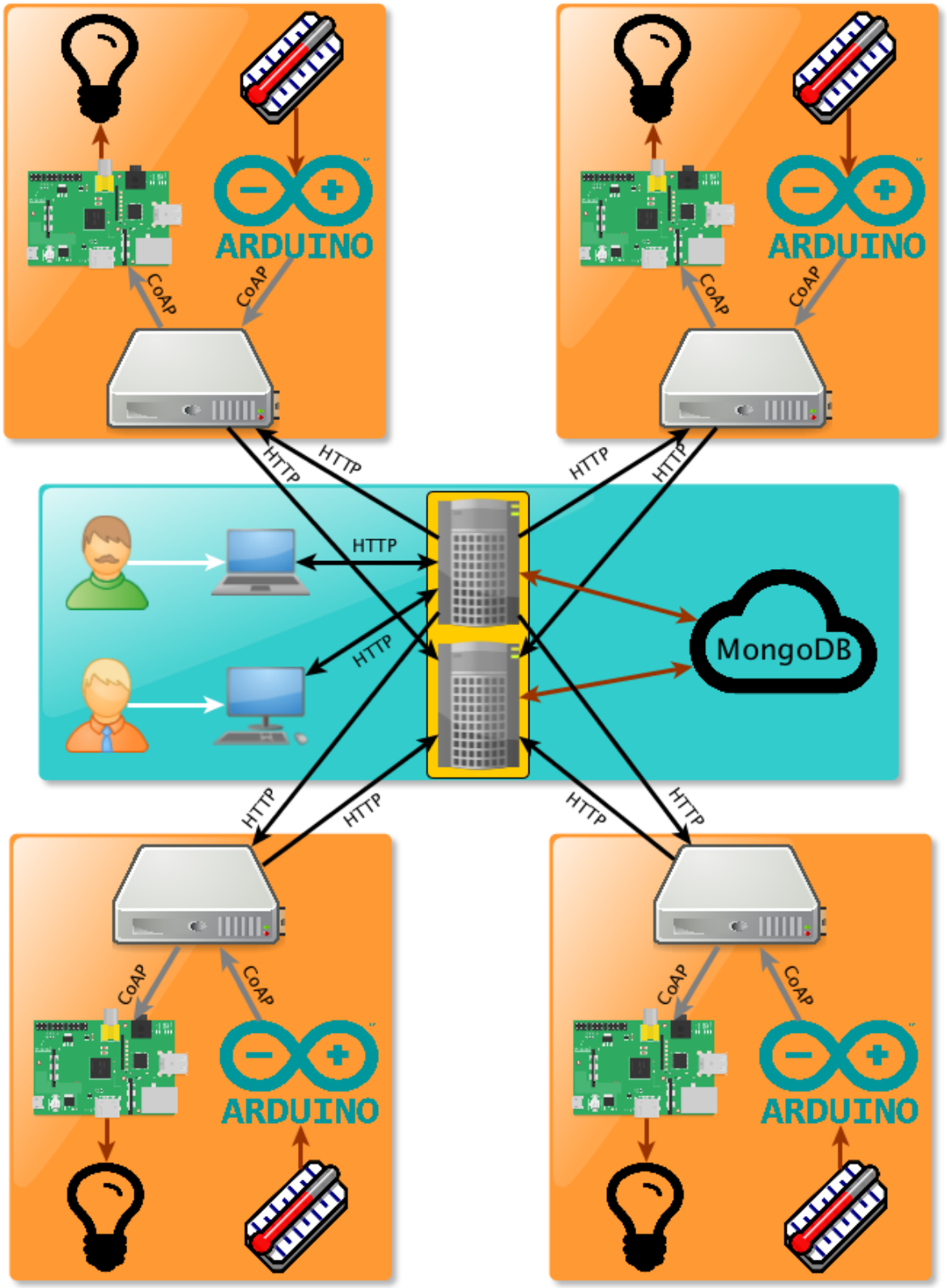


Figura 3.3. Schema della soluzione sviluppata

Come si può notare non ci sono frecce tra i due server perché non hanno bisogno di comunicare tra di loro, in quanto sono stati progettati in modo da essere indipendenti

l'uno dall'altro, per evitare di aumentare la latenza del sistema. In ogni caso da un punto di vista logico il server viene considerato unico e sempre sotto il nome di "server remoto".

3.3.2. Gestione della sicurezza

Per gestire la sicurezza vengono usati gli algoritmi di crittografia RSA e AES. Le chiavi dell'algoritmo RSA possono essere a 1024 o a 2048 bit, mentre quelle dell'AES possono essere a 128 oppure, scaricando la Java Cryptography Extension (JCE), a 256 bit.

Il server espone la chiave pubblica RSA a tutti, infatti, dal client viene richiesta automaticamente quando se ne ha la necessità, come ad esempio quando bisogna registrarsi al sistema. L'amministratore del server può decidere in qualsiasi momento di cambiare la chiave pubblica, in quanto vi sono meccanismi, spiegati meglio nel capitolo successivo, che permettono di rilevare quando un client ha bloccato la chiave, ovvero quando la sta usando per mandare dei dati al server; quindi la chiave pubblica verrà cambiata solo quando tutti client avranno finito di eseguire le loro operazioni o quando la sessione con tutti i client sarà scaduta, escludendo così il pericolo che server e client utilizzino contemporaneamente chiavi diverse.

Se non esiste alcuna coppia di chiavi RSA, essa viene creata all'avvio, sia nel server sia nel client. La chiave AES, invece, è creata dal server al momento della registrazione dell'utente e viene inviata al client al momento del login criptandola con la sua chiave pubblica, in modo tale che solo il client, con la sua chiave privata, potrà decriptarla. Siccome il client è stato realizzato con lo scopo di far connettere l'utente da quanti client volesse (non contemporaneamente) al momento del login se non è presente la chiave AES, ad esempio perché ci si sta connettendo per la prima volta da un client diverso da quello con cui ci si è registrati, essa viene richiesta e, se il login avviene con successo, il server manda la chiave, criptandola con la chiave pubblica del nuovo client, che viene mandata da quest'ultimo al momento del login.

Con questo sistema l'unico inconveniente è che non si può cambiare la chiave AES, perché il client ogni volta che trova una chiave AES nel suo file system riguardante quel determinato utente assume che sia corretta e la utilizza. Per fare in modo che si potesse cambiare la chiave AES si potevano scegliere altre due strade:

- il server tiene traccia di tutti i client dai quali l'utente si era collegato, notificandoli se la chiave AES era stata modificata;
- il server manda a ogni nuova sessione la chiave AES e non la cambia fin quando la sessione non è conclusa, a meno di mettersi d'accordo con il client.

Questi due metodi sono abbastanza dispendiosi in termini di risorse, perché aggiungono complessità all'operazione di login, che è l'operazione che si esegue con più frequenza, quindi si è preferito scegliere la soluzione di non poter cambiare la password AES.

Le operazioni legate all'utente in se, quindi login, logout, eliminazione dell'account e registrazione, sono effettuate con l'algoritmo RSA, mentre tutte le altre operazioni, come l'inserimento di un nuovo dispositivo, la ricezione di misurazioni, eccetera, vengono criptate con la chiave simmetrica AES.

3.3.3. Gerarchia dei gruppi

Una caratteristica fondamentale del sistema è l'organizzazione in gerarchie dei gruppi di utenti e dispositivi.

Gerarchia dei gruppi di utenti

La gerarchia dei gruppi di utenti si traduce in un'ereditarietà dei privilegi rispetto al gruppo padre. Questo significa che un gruppo figlio ha tutti i privilegi del gruppo padre, ma può anche aggiungerne di nuovi. Quando questo avviene, i nuovi privilegi saranno trasmessi anche ai suoi discendenti, ma non ai suoi antenati.

Un esempio potrebbe essere quello di un'università in cui si ha un gruppo di studenti che sono abilitati ad aprire le porte che conducono alle aule. Se da questo gruppo viene creato un sottogruppo di tirocinanti che, oltre a seguire le lezioni di mattina, nel pomeriggio deve recarsi presso alcuni laboratori, gli utenti appartenenti al gruppo di tirocinanti saranno già abilitati ad aprire le porte che conducono alle aule e basterà aggiungere solo i permessi che i tirocinanti hanno in più rispetto agli studenti normali per completare la registrazione del nuovo gruppo, quindi non bisognerà riscrivere ogni volta, per ogni gruppo, dei permessi che sono già stati descritti per altri gruppi.

Se si vuole far appartenere il dispositivo a più gruppi di utenti che non appartengono alla stessa gerarchia, è possibile specificare più gruppi per utente.

Gerarchia dei gruppi di dispositivi

Grazie all'organizzazione di tipo gerarchica, risulta molto più semplice accedere ai dispositivi per gli utenti. Accedendo a un gruppo, infatti, verranno selezionati sia i dispositivi che appartengono a quel gruppo sia quelli che appartengono ai gruppi figli. Specificando la gerarchia, dunque, si potrà ottenere la condizione più semplice da gestire, ovvero quella in cui un dispositivo può appartenere solamente ad un gruppo, senza rinunciare alla possibilità di accedere a un dispositivo da più gruppi.

Se non ci fosse il meccanismo di gerarchia, dunque, se si volesse far appartenere un dispositivo a più gruppi, si dovrebbe aggiungere quel dispositivo a tutti questi gruppi mentre, se i gruppi in questione sono tutti legati da un rapporto di discendenza, basterà aggiungere il dispositivo nell'ultimo discendente.

Un esempio di gerarchia per i gruppi di dispositivi potrebbe essere quella di un palazzo in cui vi è una lampadina all'interno di una stanza e si vuole accedere a questa lampadina sia quando si vogliono controllare tutti dispositivi del palazzo, sia quelli del piano dove si trova la stanza, sia quelli della stanza. In questo caso basterà creare un gruppo relativo al palazzo, un gruppo relativo al piano e un gruppo relativo alla stanza, estendendoli nell'ordine in cui sono stati elencati. In questo modo, quando si accederà al gruppo del palazzo, tutti i dispositivi che sono in un gruppo in cui vi è come antenato il gruppo del palazzo e i dispositivi che appartengono al gruppo del palazzo verranno coinvolti nel servizio, inclusa la lampadina interessata. Allo stesso modo, la lampadina sarà coinvolta quando si accederà sia al gruppo del piano sia a quello della stanza.

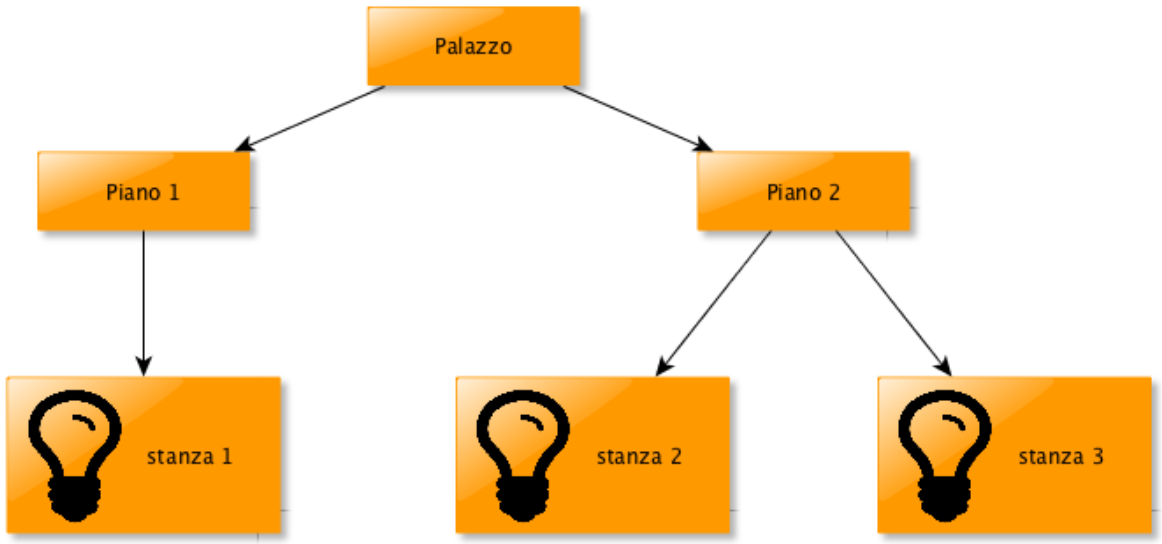


Figura 3.4. Schema dell'esempio

In ogni caso, se si volesse far appartenere il dispositivo a gruppi diversi che non sono legati tra di loro da rapporti di discendenza, ad esempio perché lo si vuole aggiungere a gruppi di altri utenti, è comunque possibile specificare più gruppi di appartenenza per dispositivo.

3.3.4. Gestione dei tag

All'interno del sistema si è pensato di introdurre un concetto utilizzato recentemente, ad esempio nei social network, che consente di ottimizzare la ricerca di servizi ed

informazioni, ovvero i tag. Mediante questo strumento, gli utenti possono etichettare qualsiasi dispositivo di loro proprietà, in modo tale da associarlo a un concetto particolare (es., un'attività, una funzionalità, una posizione,…). Il sistema, dunque, permette di eseguire l'accesso ai dispositivi mediante queste parole chiave.

Durante lo sviluppo, tuttavia, una delle questioni che ha dato più da pensare è stata l'utilizzo simile che si fa di tag e gruppi. La funzione principale di entrambi gli elementi, infatti, è quella di permettere l'accesso a più dispositivi contemporaneamente sotto un unico nome logico, quindi si è pensato alle possibilità di unire questi due concetti o di lasciali separati.

La prima opzione consiste nel far coincidere i tag con i gruppi. Questi ultimi, dunque, sono creati mediante i tag che, a questo punto, rappresentano semplicemente i nomi dei gruppi. Nello specifico, nel momento in cui si aggiunge un tag a un dispositivo, per gli attuatori significa mettersi in ascolto all'indirizzo IP multicast del gruppo il cui nome corrisponde al tag, mentre per i sensori significa aggiungere il gruppo, il cui nome corrisponde al tag, nel documento di quel sensore nel database MongoDB.

Questo approccio ha sia lati positivi che lati negativi:

Pro

- riduzione della complessità del progetto, in quanto tag e gruppi vengono gestiti come un'unica entità e non come due separate;
- quando si fa un accesso mediante tag a degli attuatori, si sa già a quali dispositivi mandare la richiesta in quanto, poiché un tag ha una corrispondenza 1 a 1 con un indirizzo IP multicast, non c'è bisogno di cercare quali attuatori hanno quel determinato tag, dunque si riduce la latenza.

Contro

- se vi sono molti tag, un attuatore deve associarsi a molti indirizzi multicast perché ogni tag, di fatto, corrisponde a un gruppo;
- snaturamento del concetto di tag per evitare la perdita del sistema di gerarchie dei gruppi.

La seconda possibilità è mantenere i tag e i gruppi separati, quindi specificare, per ogni dispositivo, sia i gruppi di cui fa parte sia i tag che gli sono stati assegnati. In questo caso, per accedere ai sensori mediante i tag, basterà fare una query su MongoDB per trovare tutti i sensori che contengono quel determinato tag, mentre per gli attuatori bisogna mandare il messaggio a tutti i gruppi multicast interessati. Una volta ricevuto il messaggio, se nel payload è specificato un tag, ogni attuatore controllerà se possiede quel tag e solo in caso affermativo il dispositivo eseguirà l'istruzione ricevuta.

Un tag all'interno di un attuatore con questo approccio, dunque, non si riferisce più a un gruppo, ma rappresenta un'etichetta che gli utenti danno per classificare i loro dispositivi.

Pro

- utilizzo dei tag più classico;
- l'utente può creare quanti tag desidera senza inficiare sulle prestazioni del singolo attuatore, in quanto quest'ultimo sta in ascolto solamente dei gruppi.

Contro

- ricerca dei tag piuttosto onerosa negli attuatori se va fatta su molti dispositivi, perché il messaggio deve essere mandato a tutti i gruppi su cui si desidera effettuare la ricerca del tag, quindi tutti i dispositivi riceveranno il messaggio per controllare se possiedono il tag o meno;
- gli utenti devono gestire sia i tag sia i gruppi, perché questi non sono più collegati, il che implica una maggiore complessità del sistema, sia a livello concettuale che a livello progettuale.

Tra i contro del primo approccio e i pro del secondo ve ne sono due che non sono stati commentati volutamente, perché rappresentano gli aspetti che, più di tutti, hanno fatto scegliere di adottare nel progetto la seconda opzione, dunque necessitano di un discorso più approfondito, che si affronterà qui di seguito.

Il pro a cui si fa riferimento è l'utilizzo dei tag più classico, mentre il contro è lo snaturamento del concetto di tag per evitare la perdita del sistema di gerarchie dei gruppi. Nei vari social network non si è mai visto che i tag fossero organizzati in maniera gerarchica, perché sarebbe molto complicato da gestire per l'utenza media. In questo progetto, tuttavia, la gerarchia dei gruppi, sia di dispositivi sia di utenti, è presente tra i requisiti, quindi non è qualcosa alla quale si può rinunciare e, se si vuole unire il concetto di tag a quello di gruppo, è inevitabile che si debba introdurre una gerarchia per i tag. Siccome l'utenza è abituata a usare i tag nella sua quotidianità ed ha ben chiaro in mente cosa rappresentano, si è ritenuto che pensare i tag in maniera gerarchica risulti più complicato di mantenere separati i concetti di tag e gruppi per l'utente medio, quindi si è deciso di adottare la seconda opzione.

In definitiva, i gruppi devono essere usati per tutti gli accessi che si fanno più frequentemente, mentre i tag per operazioni che si fanno molto raramente, in modo da non sovraccaricare il sistema durante le operazioni più frequenti (cosa che accade nel primo approccio, in quanto ogni attuatore sta in ascolto su tutti i tag) senza perdere la possibilità di attivare servizi particolari mediante i tag quando se ne ha la necessità.

In ogni caso per selezionare i gruppi in maniera smart, vi sono anche i tag per gruppi, chiamati "group tags", che consentono di selezionare tutti i gruppi che hanno dei determinati tag.

3.4. Requisiti

La soluzione sviluppata prevede che ciascun componente del sistema risponda a determinati requisiti.

3.4.1. Utente

L'utente deve potere eseguire le seguenti operazioni:

Gestione gruppi di utenti

- creare gruppi di utenti;
- aggiungere altri utenti ai propri gruppi di utenti;
- rimuovere altri utenti dai propri gruppi di utenti;
- verificare a quali gruppi di utenti si appartiene;
- verificare quali altri utenti appartengono a un gruppo a cui si appartiene;
- uscire da un gruppo di utenti;
- avere accesso a tutti i dispositivi che sono accessibili a tutti i gruppi di utenti ai quali si fa parte;
- organizzare i propri gruppi di utenti con una gerarchia.

Gestione della sicurezza dei dispositivi

- rendere un proprio dispositivo **privato**: il dispositivo può essere aggiunto solo ai gruppi di dispositivi di possesso del proprietario del dispositivo e inoltre solo quest'ultimo può accedervi;
- rendere un proprio dispositivo ristretto solo ad alcuni gruppi di utenti: l'utente specifica quali gruppi di utenti possono accedere al proprio dispositivo;
- rendere un proprio dispositivo ristretto solo ad alcuni gruppi di dispositivi: l'utente specifica a quali gruppi di dispositivi può essere aggiunto il proprio; dispositivo;
- rendere un proprio dispositivo ristretto solo ad alcuni gruppi di utenti e di dispositivi;
- rendere un proprio dispositivo **pubblico**: il dispositivo può essere aggiunto a qualsiasi gruppo di dispositivi e, inoltre, qualsiasi utente può accedere a quel dispositivo.

Gestione della sicurezza generale

- Scambiare dati con il server remoto in maniera sicura e affidabile.

Gestione dispositivi

- registrare i propri dispositivi online;
- creare gruppi di dispositivi;
- aggiungere a un gruppo di cui si è il proprietario altri dispositivi di proprietà, pubblici o che hanno quel gruppo tra i gruppi ristretti;
- creare dei servizi personalizzati, installarli sui propri server e usarli in modo agevole;
- aggiungere e rimuovere tag ai propri dispositivi;
- accedere ai dispositivi mediante filtri come marca, tag, posizione e nome;
- organizzare i propri gruppi di dispositivi con una gerarchia.

Gestione del proprio account

- registrarsi al servizio specificando username, password ed email;
- eseguire il login mediante username e password;
- eseguire il logout;
- eliminare il proprio account.

3.4.2. Server remoto

Il server remoto deve implementare le seguenti funzionalità:

- salvare i dispositivi che vengono registrati dagli utenti;
- scambiare dati in maniera sicura e affidabile con l'utente e i server CoAP;
- fornire dei mezzi per facilitare la creazione, l'installazione e l'utilizzo dei servizi personalizzati;
- funzionare da "ponte" tra utenti e server CoAP;
- non deve salvare informazioni personali nel cloud.

3.4.3. Server CoAP

Il server CoAP deve implementare le seguenti funzionalità:

- installare ed utilizzare i servizi;
- funzionare da "ponte" tra server remoto e dispositivi, mandando al server remoto le misurazioni dei sensori e mandando agli attuatori le richieste che provengono dal server remoto;

- deve comunicare con il server remoto mediante HTTP e con i dispositivi con il protocollo CoAP;
- attivare i gruppi di attuatori in base alla locazione dell'interruttore attivato e all'orario, con regole stabilite dall'utente.

3.4.4. Dispositivi

Tutti i dispositivi

- Devono poter comunicare mediante il protocollo CoAP.

Sensori

- Nei sensori deve esserci la possibilità di impostare una soglia che stabilisca se la misurazione effettuata è significativa o meno.

Attuatori

- Devono potersi collegare a gruppi multicast.

3.5. Funzionalità

Oltre alle varie funzionalità descritte precedentemente, ve ne sono alcune più specifiche che sono spiegate qui di seguito.

3.5.1. Soglia dei sensori

Per quanto riguarda i sensori si è pensato di introdurre una soglia, che stabilisce se la misura effettuata ha necessità di essere notificata al server oppure no. Ad esempio, per un sensore di temperatura nella maggior parte dei casi ha poco senso specificare che la temperatura è cambiata di 0,04 °C, quindi l'utente per evitare di intasare la propria sottorete e di salvare misure inutili, potrebbe impostare 0,1 come soglia per quel sensore, in modo tale da apprezzare solamente cambiamenti significativi della grandezza che si sta misurando, in quanto il sensore manderà una nuova misurazione solo se questa si discosterà di un valore superiore di 0,1 dalla misura inviata precedentemente.

Ogni volta che un sensore effettua una nuova misurazione, dunque, la confronta con quella precedente. Se la misurazione differirà da quella precedente di un valore superiore in valore assoluto alla soglia, allora la misurazione verrà ritenuta valida, quindi verrà mandata al server CoAP e sostituita con quella precedente per i futuri confronti.

3.5.2. Servizi personalizzati

Nel server CoAP è stato realizzato un database mysql, che registra le corrispondenze tra gruppi di attuatori, orari e le locazioni, in modo tale che se un interruttore di una determinata locazione viene attivato a una certa ora del giorno si attivavano o si disattivavano solo determinati gruppi.

Questa, tuttavia, è solo una delle svariate possibili funzioni che può avere questo sistema e si è pensato che in alcune situazioni potrebbe essere l'utente stesso a volerle sviluppare. Per questo motivo è stata data la possibilità di creare i servizi per i server CoAP al possessore del server stesso.

Nello specifico nel server CoAP viene creata una cartella nominata con il nome del servizio, all'interno della quale deve obbligatoriamente essere presente un programma nominato "main.py", che è il programma che viene lanciato quando viene attivato il servizio. Oltre a questo possono essere presenti tutti gli altri file che sono necessari al servizio stesso, come ad esempio altri script python o i dati di cui ha bisogno il programma.

Il programma viene lanciato da remoto, mediante una POST (CoAP o HTTP), indicando nel payload il nome del servizio e i parametri che si vogliono passare ad esso. Questi ultimi verranno interpretati come parametri della linea di comando, quindi se il contenuto del payload è: "service1 parametro1 parametro2" nel coap server verrà lanciato il comando "main.py parametro1 parametro2", dove main.py è contenuto nella cartella service1.

Registrare un nuovo servizio è semplicissimo, infatti basterà selezionare, mediante un'interfaccia grafica user friendly, la cartella nominata come il servizio, contenente tutti i file di cui esso necessita, compreso il file "main.py", e indicare a quali server di propria proprietà si vuole inviare il servizio. Il client comprimerà la cartella e la invierà al server remoto, che si occuperà di smistare il contenuto del file zip ai vari server CoAP.

In fase di registrazione del servizio viene chiesto se lo si vuole rendere pubblico, ristretto solo ad alcuni gruppi di utenti o privato, ovvero accessibile solo dal proprietario del server. Se si vuole che il servizio sia disponibile anche a dispositivi non dotati di autenticazione (ad esempio un interruttore comune), bisogna impostare il servizio come pubblico.

L'esempio fatto precedentemente sul contenuto del payload e dei parametri della linea di comando è inesatto in quanto, per semplicità, si è omesso un parametro che deve essere sempre presente, ovvero quello chiamato "actuators_groups". Mediante questo

parametro bisogna specificare la lista di gruppi di attuatori su cui si vuole attivare il servizio, specificandone il nome e l'utente che li possiede (se non è specificato si sottindende che siano di proprietà dell'utente che sta usando il servizio).

Il server remoto si occuperà di trovare gli ip dei gruppi di attuatori tra i gruppi degli utenti indicati e gli ip trovati verranno sostituiti ai nomi dei gruppi all'interno del parametro "actuators_groups". Esso, inoltre, manderà la richiesta di servizio a tutti i server che contengono almeno uno dei gruppi di attuatori indicati. Il programma "main.py" del servizio specifico, dunque, sarà richiamato solo sui server che contengono almeno uno dei gruppi di attuatori indicati. Questi server, come parametri di linea di comando, si ritroveranno gli stessi parametri che chi ha richiesto il servizio ha mandato al server remoto, con la differenza che ora "actuators_groups" conterrà una lista degli indirizzi ip presenti nella sottorete di quel server, invece che i loro nomi. Ovviamente il server remoto, prima di distribuire le richieste di servizio ai vari server CoAP, farà le dovute verifiche per verificare che l'utente che richiede il servizio è abilitato a essere servito.

Un altro parametro speciale che viene fornito al servizio è quello dei tag, in quanto l'utente può attivare un servizio personalizzato solo sugli attuatori che hanno dei determinati tag. I tag inseriti dall'utente, dunque, verranno concatenati con delle virgole e si troveranno dopo la dicitura "tags:_" dopo gli ip dei gruppi di attuatori e prima degli altri parametri inseriti dall'utente.

Per fornire un esempio di come utilizzare i servizi, il comportamento degli orari e delle locazioni di prima viene trattato come se fosse un servizio, chiamato "green_activator" anche se, in realtà, per questioni di ottimizzazione, è impostato diversamente rispetto altri servizi.

A partire da questo esempio, dunque, potremmo introdurre una categoria di "servizi admin", che hanno le seguenti caratteristiche:

- possono essere aggiunti solamente dal codice sorgente del progetto stesso, quindi si possono fare delle operazioni di ottimizzazione;
- il modo in cui li si usa non deve essere diverso da quello dei servizi normali;
- se esiste un servizio utente nominato come il servizio admin, quando l'utente richiede il servizio con quel nome, viene attivato solo il servizio admin.

L'obiettivo, dunque, è quello di fornire alcuni servizi base di uso comune mediante i servizi admin e di lasciare la possibilità di sviluppare servizi personalizzati in base alle esigenze dell'utente mediante i servizi normali, che d'ora in poi verranno chiamati "servizi utente".

3.6. Casi d'uso

3.6.1. Login

1. L'utente richiede di autenticarsi inviando username e password
2. Il server invia le credenziali al database SQL
3. Il database SQL controlla se le credenziali corrispondono a un utente registrato al database e, se questo è vero, manda una risposta positiva
4. Il server manda una risposta positiva all'utente, che ora risulta autenticato

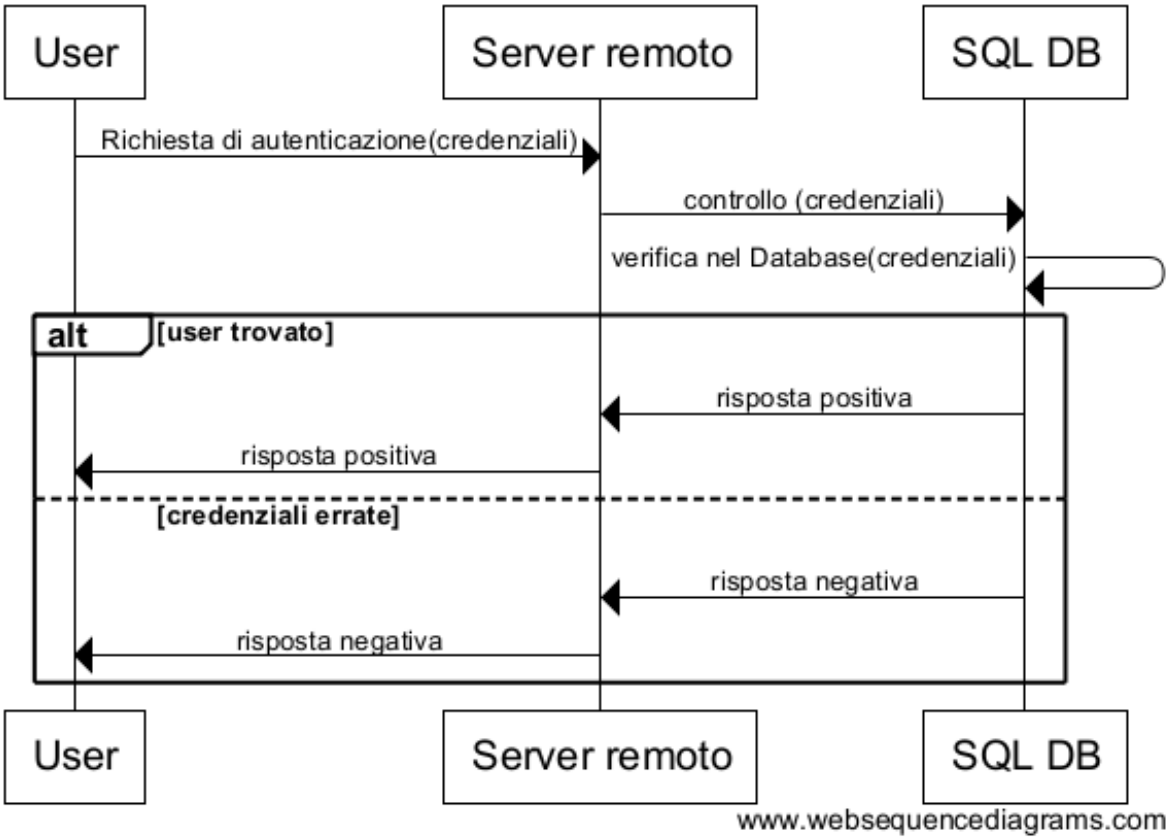


Figura 3.5. Sequence diagram del login

3.6.2. Leggere le misurazioni del sensore

Condizioni iniziali: l'utente ha già effettuato il login.

1. L'utente manda al server il nome del sensore di cui vuole sapere le misurazioni
2. il server legge le misurazioni nel database e le manda all'utente

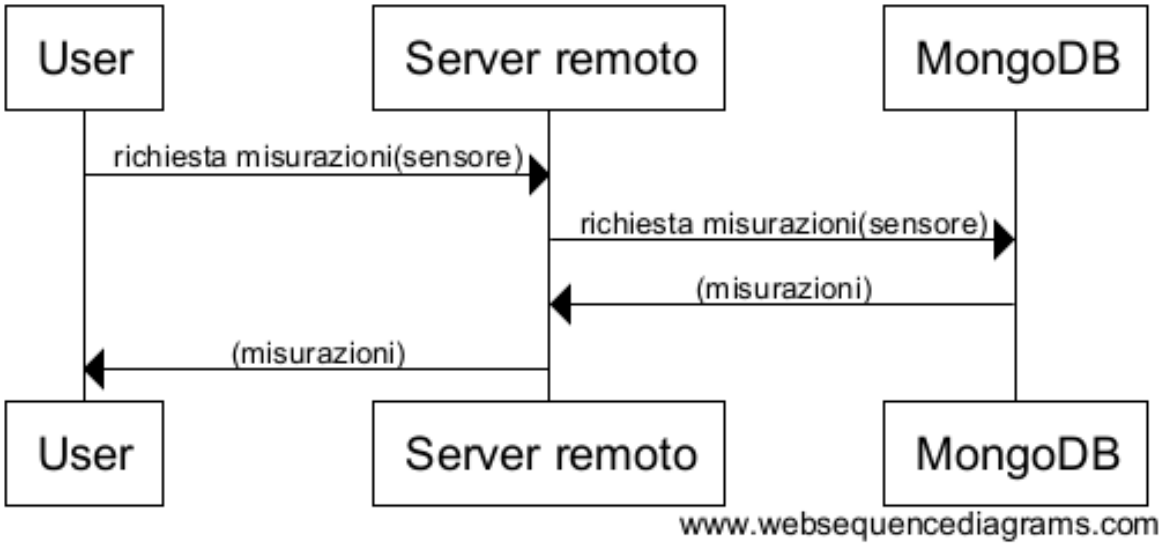


Figura 3.6. Sequence diagram della richiesta delle misurazioni

3.6.3. Attivare un servizio su un gruppo di attuatori

Condizioni iniziali: l'utente ha già effettuato il login.

1. L'utente manda al server remoto quale servizio vuole attivare, su quale gruppo di attuatori vuole attivarlo, specificando anche i corrispettivi proprietari dei gruppi, nel caso in cui essi non fossero di sua appartenenza
2. Il server remoto chiede al database a quali server CoAP appartengono i gruppi di attuatori che si vuole controllare e gli IP dei gruppi di attuatori
3. Il server invia ad ogni server CoAP trovato il nome del servizio e gli IP dei gruppi di attuatori presenti nella sua sottorete su cui attivare il servizio
4. Ogni server CoAP esegue le istruzioni del servizio, inviando i relativi comandi agli ip specificati dal server remoto.

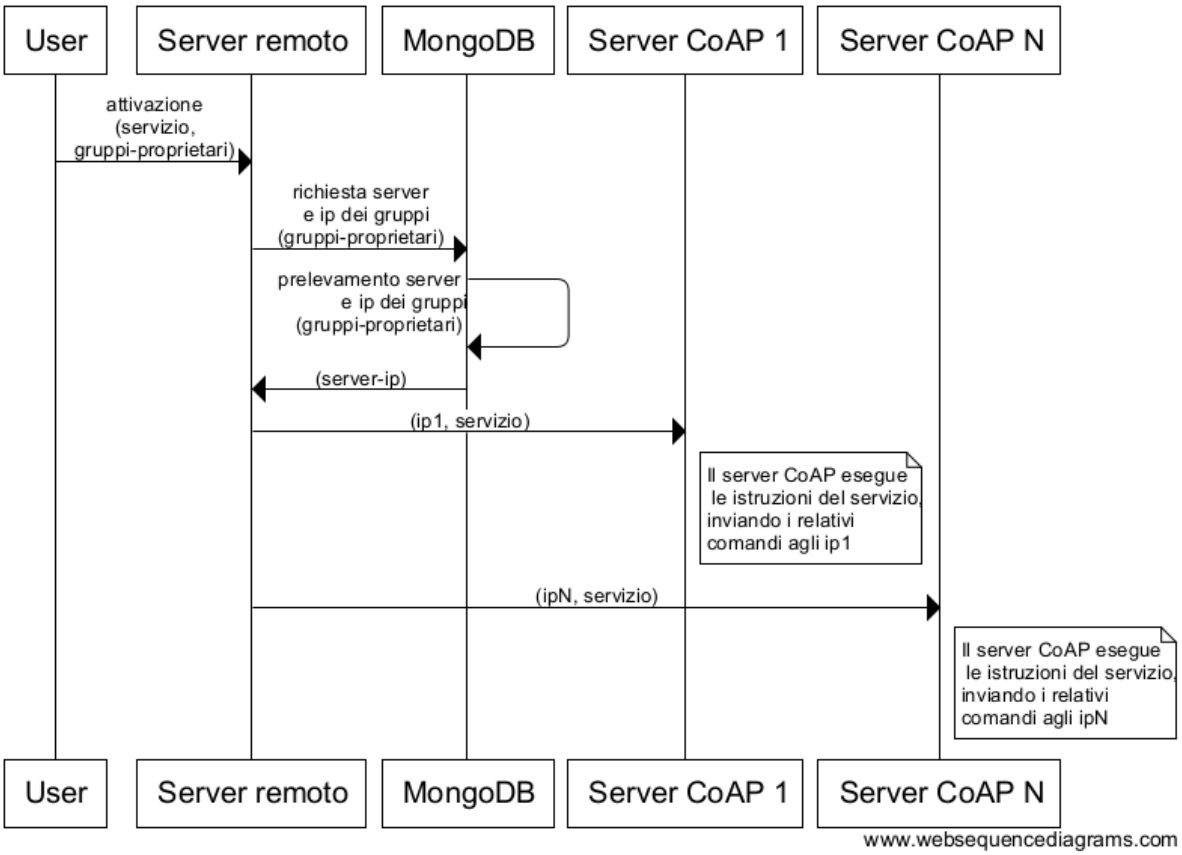


Figura 3.7. Sequence diagram dell'attivazione dei servizi

3.6.4. Inserimento di un nuovo sensore

Condizioni iniziali: l'utente ha già effettuato il login.

1. L'utente inserisce nel sensore il suo id, la sua soglia e specifica il server CoAP al quale vuole inviare i dati;
2. L'utente registra il sensore nel server remoto, specificando tutte le sue caratteristiche.
3. Il server remoto controlla se non è già stato registrato un sensore con lo stesso nome da parte dello stesso utente e, nel caso in cui il nome sia disponibile, da una risposta positiva al client

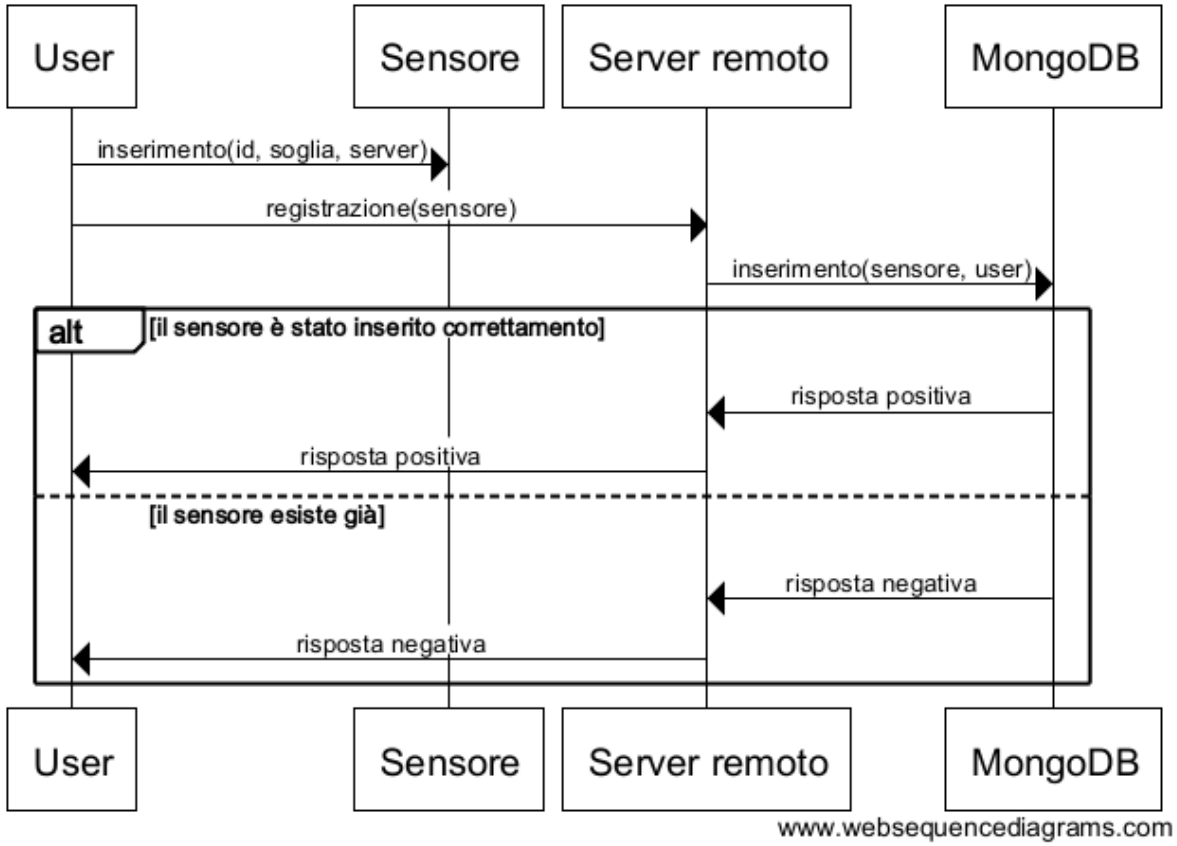


Figura 3.8. Sequence diagram dell'inserimento di un nuovo sensore

3.6.5. Inserimento di un nuovo attuatore

Condizioni iniziali: l'utente ha già effettuato il login.

1. L'utente inserisce nell'attuatore i suoi tag e i gruppi multicast al quale l'attuatore appartiene

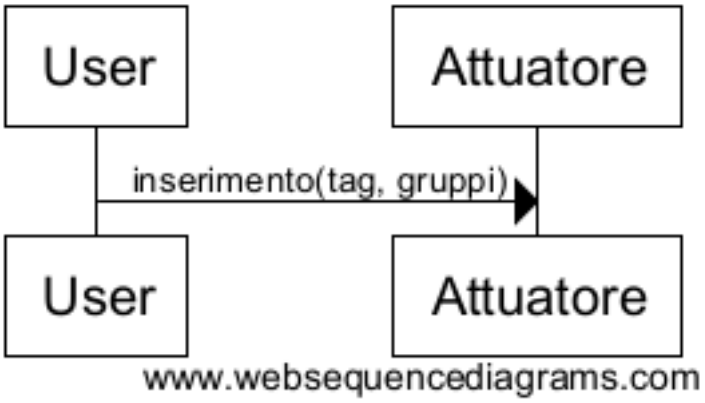


Figura 3.9. Sequence diagram dell'inserimento di un nuovo attuatore

Si noti che il server remoto non tiene traccia dei singoli attuatori in quanto il multicast ci permette di creare una astrazione e di trattarli, insieme, come gruppi.

3.6.6. Inserimento di una misurazione

Condizioni iniziali: l'utente ha già effettuato il login.

1. Il sensore effettua una misurazione
2. Il sensore confronta la misurazione effettuata con la misurazione valida precedente e se questa differisce da quest'ultima di un valore superiore in valore assoluto alla soglia, allora la misurazione verrà ritenuta valida
3. Se la misurazione è ritenuta valida, il sensore la manda al server CoAP e la sostituisce alla misurazione valida precedente per i futuri confronti
4. Il server CoAP manda la misurazione al server remoto, indicando anche l'id del sensore che l'ha effettuata
5. Il server remoto salva la misurazione ricevuta, insieme all'id del sensore e a un timestamp.

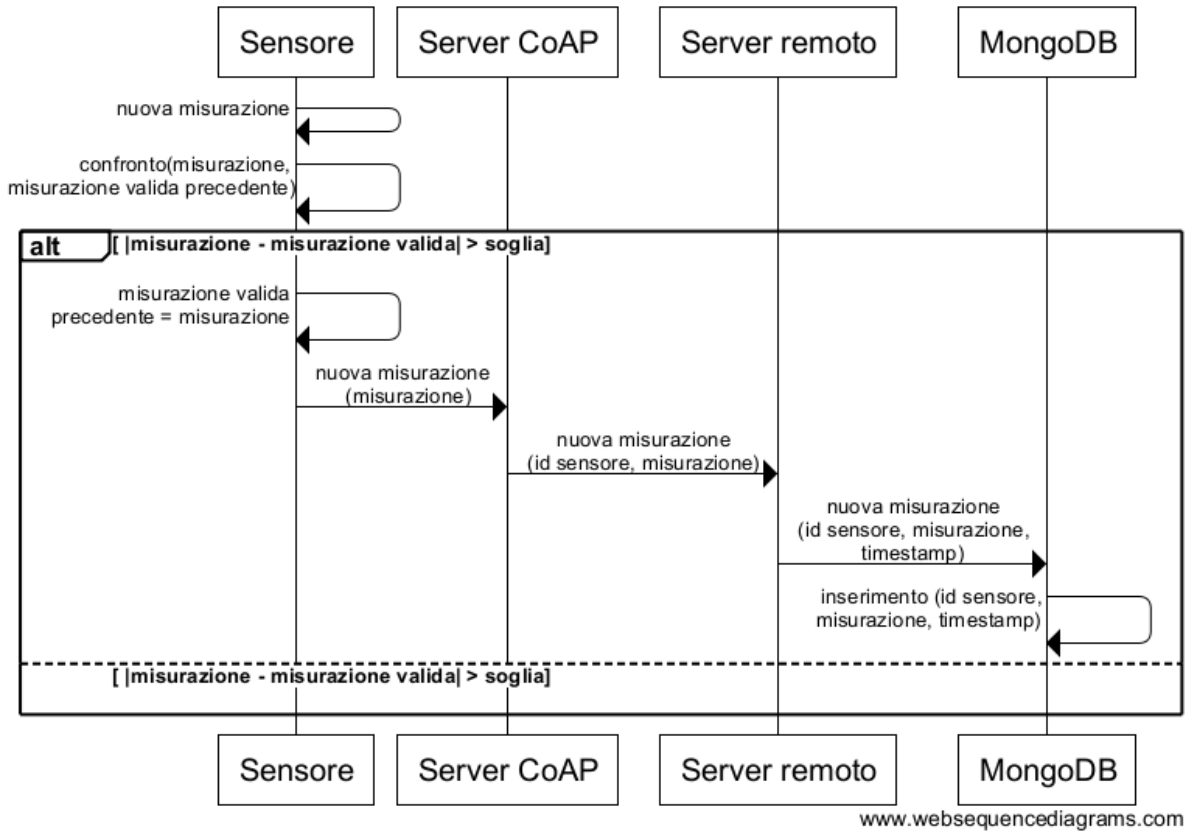


Figura 3.10. Sequence diagram dell'inserimento di una nuova misura

Capitolo 4. Dettagli implementativi

Dopo aver spiegato il funzionamento del sistema nel capitolo precedente, si illustrano di seguito i dettagli implementativi.

4.1. Linguaggi di Programmazione

E' stato utilizzato java nella versione 8 per le parti che riguardano il client, il server remoto e MongoDB (in quanto, per interfacciarsi con quest'ultimo, è stato utilizzato il driver java), mentre per il CoAP server, gli interruttori, i sensori e gli attuatori è stato utilizzato python 2.7.

La scelta della versione di python è stata forzata, in quanto il CoAPthon è basato su di essa.

Come regole di indentazione del codice si sono seguiti gli standard di default di eclipse mars, ovvero 120 caratteri massimi per riga per il codice e 80 per i commenti.

4.2. Gruppi

4.2.1. Gestione della gerarchia

Tra i vari metodi elencati nella documentazione di MongoDB per gestire gli alberi, si è scelto la struttura con materialized paths. Con questa struttura ogni nodo dell'albero viene salvato in un documento e, nello stesso documento, viene salvato il suo percorso all'interno dell'albero, ovvero gli id degli altri nodi genitori in modo ordinato.

Esempio:

```
{ _id: "gruppo1", path: null }
{ _id: "gruppo2", path: ",gruppo1," }
{ _id: "gruppo3", path: ",gruppo1,gruppo2," }
{ _id: "gruppo4", path: ",gruppo1,gruppo2," }
{ _id: "gruppo5", path: ",gruppo1,gruppo2,gruppo3," }
{ _id: "gruppo6", path: ",gruppo1,gruppo2,gruppo3," }
```

Questi documenti rappresentano la seguente struttura:

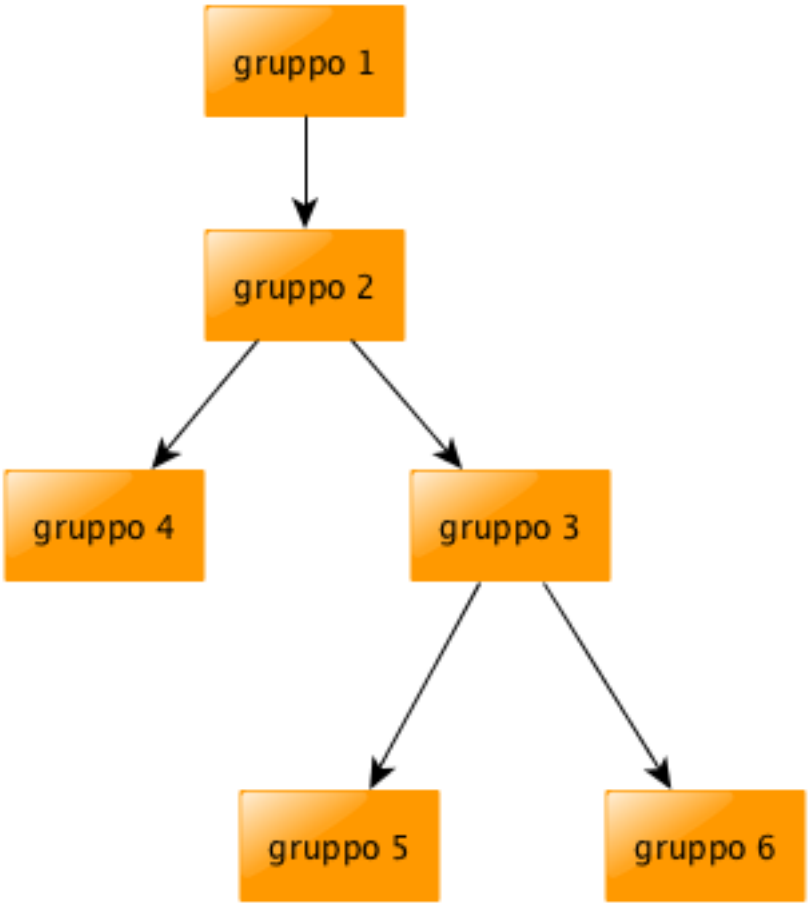


Figura 4.1. Esempio di gerarchia dei gruppi

Si è scelta questa struttura per caratterizzare tutte le collection dei gruppi del progetto perché consente di avere la struttura ad albero con cui si è scelto di organizzare i gruppi per instaurare una gerarchia tra di essi già salvata all'interno del documento.

Un'altra possibilità che è stata presa in considerazione è quella della referenza al padre in cui, per ogni nodo dell'albero, viene salvato solamente il suo nodo genitore piuttosto che tutto il suo percorso fino al nodo root. Questo sistema permette di risparmiare in termini di spazio occupato, ma va ad inficiare negativamente sulla complessità computazionale in modo non indifferente. Per gestire query come "trovare tutti i figli di un certo nodo" e non fare troppi accessi su disco per interrogare il DB è necessario creare e mantenere aggiornata una struttura dinamica in memoria, quindi per diminuire sia la complessità computazionale che quella del progetto in se, si è deciso di evitare questa struttura in favore di quella con riferimento al path completo "materializzato" nel DB stesso. Inoltre, creando un indice sul path query come quella descritta precedentemente si risolvono in tempi molto brevi grazie all'indicizzazione di MongoDB, in quanto in ogni documento è scritto ogni predecessore di quel nodo.

Per comprendere meglio questo tipo di struttura si riporta di seguito il codiche della funzione InserGroup, presente all'interno della classe DBManager e tutte le funzioni più importanti che essa richiama. Questa funzione, come si intuisce dal titolo, consente di inserire un gruppo all'interno del database MongoDB.

Inserimento di un gruppo

```
public Boolean insertGroup(GenericalGroup gg) {
 Document doc = new Document(DatabaseDefines.GROUP_ADMIN,
 getUserId(gg.getAdmin())).append(DatabaseDefines.NAME, gg.getName());
 if (gg.isPublic())
  doc.append(DatabaseDefines.IS_PUBLIC, true);
 else
  doc.append(DatabaseDefines.IS_PUBLIC, false);
 if (gg.getDescription() != null) {
  doc.append(DatabaseDefines.GROUP_DESCRIPTION, gg.getDescription());
 }
 if (gg.getTags().size() > 0)
  doc.append(DatabaseDefines.TAGS, gg.getTags());
 try {
  if (gg instanceof SensorsGroup) {
   SensorsGroup sg = (SensorsGroup) gg;
   if (sg.getPath() == null) {
    assignGroupPath(sg, DatabaseDefines.SENSORS_GROUP_COLLECTION);
   }
   doc.append(DatabaseDefines.PATH, sg.getPath());
   if (sg.getSensors() != null)
    doc.append(DatabaseDefines.SENSORS, getSensorsId(sg.getSensors()));
   db.getCollection(DatabaseDefines.SENSORS_GROUP_COLLECTION).insertOne(doc);
  } else if (gg instanceof UsersGroup) {
   UsersGroup ug = (UsersGroup) gg;
   if (ug.getPath() == null) {
    assignGroupPath(ug, DatabaseDefines.USERS_GROUP_COLLECTION);
   }
   doc.append(DatabaseDefines.PATH, ug.getPath());
   if (ug.getUsers() != null)
    doc.append(DatabaseDefines.USERS, ug.getUsers());
   db.getCollection(DatabaseDefines.USERS_GROUP_COLLECTION).insertOne(doc);
  } else {
   ActuatorsGroup ag = (ActuatorsGroup) gg;
   if (ag.getPath() == null) {
    assignGroupPath(ag, DatabaseDefines.ACTUATORS_GROUP_COLLECTION);
   }
   doc.append(DatabaseDefines.PATH, ag.getPath());
   if (ag.getIp() != null) {
    doc.append(DatabaseDefines.ACTUATORS_GROUP_IP, ag.getIp());
    doc.append(DatabaseDefines.SERVER_IP_PORT, ag.getServer().getIpPort());
   }
   db.getCollection(DatabaseDefines.ACTUATORS_GROUP_COLLECTION).insertOne(doc);
  }
 } catch (Exception e) {
  e.printStackTrace();
  return false;
 }
```

```
 return true;
}

public void assignGroupPath(GenericalGroup gg, String collection) {
 if (gg.getParentGroupName() == null) {
  gg.setPath(",");
 } else {
  String parentGroupPath = findParentGroupPath(gg, collection);
  if (parentGroupPath == null) {
   System.err.println("parent group does not exists, the group will be saved as root");
   gg.setPath(",");
  } else {
   String path = parentGroupPath + gg.getParentGroupName() + ",";
   gg.setPath(path);
  }
 }
}
```

4.2.2. Gruppi di attuatori

Quando si inserisce un gruppo di attuatori vengono chiesti anche l'ip e la porta del server di appartenenza del gruppo, in modo tale che il server remoto sappia come raggiungere il gruppo.

In ogni caso, per un gruppo di attuatori l'IP e il server possono essere NULL perché, in questo caso, si sta esplicitando un gruppo logico, ovvero un gruppo che serve unicamente per avere altri gruppi figli, in modo da richiamarli contemporaneamente quando l'utente si riferisca a questo gruppo logico.

I gruppi, inoltre, possono avere sia IP e server specificati sia avere dei gruppi figli. In tal caso, infatti, la richiesta di servizio verrà inviata sia all'IP del gruppo padre sia all'IP dei gruppi figli. Per la gestione della sicurezza, nel documento degli actuators group viene salvato il campo "allowed users groups", in modo tale da specificare i gruppi di utenti che sono abilitati ad accedere a quel gruppo. Questo ovviamente ha effetto solo se il gruppo non è pubblico.

Per far capire meglio questi concetti viene riportata la parte di codice che si occupa di attivare un servizio, da quando viene presa in carico la richiesta da parte del server a quando viene mandata ai server CoAP.

Attivazione di un servizio

funzione all'interno di DBManager:

```
public Vector<ActuatorsGroup> getDescendantsActuatorsGroup(ActuatorsGroup group) {
```

```
  Document find_conditions = new Document(DatabaseDefines.GROUP_ADMIN, group.getAdmin())
      .append(DatabaseDefines.PATH, Pattern.compile("," + group.getName() + ","));
  Vector<ActuatorsGroup> descendants = new Vector<ActuatorsGroup>();
  MongoCursor<Document> cursor = db.getCollection(DatabaseDefines.ACTUATORS_GROUP_COLLECTION)
      .find(find_conditions).iterator();
  while (cursor.hasNext()) {
    descendants.add(new ActuatorsGroup(cursor.next().toJson().toString()));
  }
  return descendants;
}
```

funzioni all'interno di UserThread:

```
private void activateService() {
 Vector<ActuatorsGroup> actuatorsGroups = (Vector<ActuatorsGroup>)
 receiveAesDecryptedObject();
 String service = receiveAesDecryptedString();
 String commandLineParameters = receiveAesDecryptedString();
 Vector<ActuatorsGroup> allGroups = getGroupsForService(actuatorsGroups);
 requestService(allGroups, service, commandLineParameters);
}

private Vector<ActuatorsGroup> getGroupsForService(Vector<ActuatorsGroup> actuatorsGroups) {
 Iterator<ActuatorsGroup> it = actuatorsGroups.iterator();
 Vector<ActuatorsGroup> allGroups = new Vector<ActuatorsGroup>();
 Vector<ActuatorsGroup> descendants;
 ActuatorsGroup ag = null;
 while (it.hasNext()) {
  ag = it.next();
  if (!dbm.isUserAllowed(ag, lm.getNickname()))
   continue;
  ag = dbm.getCompleteInformations(ag);
  // I check if the user can access to this actuators group

  allGroups.add(ag);
  descendants = dbm.getDescendantsActuatorsGroup(ag);
  if (descendants.size() > 0) {
   allGroups.addAll(descendants);
  }
 }
 return allGroups;
}

/**
 * it sends an HTTP requests to the servers of the groups to activate the
 * services
 *
 * @param groups
 *            all the groups that you want to get access.
 * @param service
 *            the service that you want to activate
 * @param commandLineParameters
 *            the parameters for the service
 */
public void requestService(Vector<ActuatorsGroup> groups, String service, String
 commandLineParameters) {
```

```
 String server;
 Iterator<ActuatorsGroup> i;
 ActuatorsGroup ag;
 Vector<String> groupsIp = new Vector<String>();
 while (groups.size() > 0) {
  server = groups.get(0).getServer().getIpPort();
  i = groups.iterator();
  while (i.hasNext()) {
   ag = i.next();
   if (ag.getServer().getIpPort().equals(server)) {
    groupsIp.add(ag.getIp());
    i.remove();
   }
  }
  requestServiceToServer(server, groupsIp, service, commandLineParameters);
  groupsIp.removeAllElements();
 }
}

private void requestServiceToServer(String serverIpPort, Vector<String> groupsIpOfTheServer,
 String service,
    String commandLineParameters) {
  String url = "http://" + serverIpPort + "/basicService";
  List<NameValuePair> urlParameters = new ArrayList<NameValuePair>();

  String groupsIpString =
 groupsIpOfTheServer.toString().replace("[", "").replace("]", "").replace(" ", "");
  urlParameters.add(new BasicNameValuePair("groupsIps", groupsIpString));
  if (commandLineParameters != null) {
    urlParameters.add(new BasicNameValuePair("service", service));
    urlParameters.add(new BasicNameValuePair("commandLineParameters",
 commandLineParameters));
  }
  try {
    http.sendPost(url, urlParameters);
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

Oltre a questa modalità di attivazione sono presenti anche quelle mediante tags e group tags.

4.2.3. Gruppi di sensori

Per quanto riguarda i gruppi di sensori, invece, verranno mostrate all'utente solo le misure dei sensori a cui l'utente è abilitato ad accedere. Quando vengono richieste le misure di un gruppo, infatti, il server remoto fornisce solo le misure dei sensori del gruppo a cui l'utente è abilitato ad accedere.

Per chiarire meglio l'accesso ai gruppi di sensori viene riportata la parte di codice che se ne occupa:

Lettura di misure di gruppi di sensori

Funzione all'interno di UserThread:

```
private void sendSensorsGroupMeasurements() {
 SensorsGroup group = (SensorsGroup) receiveAesDecryptedObject();
 // I pass the nickname of the user in order to send only the
 // measurements that he is able to see
 sendAesEncryptedObject(dbm.getMeasurements(group, lm.getNickname()));
}
```

funzioni all'interno di DBManager:

```
/**
 *
 * @param group
 * @param username
 * @return the measurements of the group and the measurements of the
 *         descendants of that group, but only those ones that the user is
 *         allowed to see
 *
 */
public Vector<Measurement> getMeasurements(SensorsGroup group, String username) {
 Vector<SensorsGroup> descendants = getDescendantsSensorsGroup(group);
 Vector<Sensor> sensors = getSensors(group);
 Iterator<SensorsGroup> descendantsIt = descendants.iterator();
 SensorsGroup sg;
 while (descendantsIt.hasNext()) {
  sg = descendantsIt.next();
  sg.setAdmin(getNickname(sg.getAdmin()));
  sensors.addAll(getSensors(sg));
 }
 Vector<Measurement> measurements = new Vector<Measurement>();
 Iterator<Sensor> it = sensors.iterator();
 Sensor s;
 while (it.hasNext()) {
  s = it.next();
  s.setOwner(getNickname(s.getOwner()));
  if (isUserAllowedForSensor(s, username))
   measurements.addAll(s.getMeasurements());
 }
 return measurements;
}
```

4.3. Gestione della privacy e della sicurezza

4.3.1. Approccio ibrido SQL e NoSQL

Visto il requisito del server di non poter salvare dati personali nel cloud si è utilizzato un approccio ibrido in cui tutti i dati personali sono salvati in un database SQL insieme all'ID

dell'utente, che viene generato dal database stesso, mentre tutti gli altri dati vengono salvati nel database MongoDB, identificando ogni utente solo mediante il suo ID.

In questo modo nel cloud non vi sono username, password, email e altri dati che si potrebbero aggiungere in future versioni del software come nome, cognome, eccetera, ma solamente un ID, la cui corrispondenza con gli altri dati si può trovare unicamente consultando il database relazionale.

4.3.2. Accesso ai dispositivi

Nel capitolo precedente abbiamo detto che possono accedere ai vari disposotivi solamente gli utenti abilitati a farlo. A titolo di esempio riportiamo il codice che si occupa di verificare che un utente sia abilitato ad accedere a un sensore.

```
public Boolean isUserAllowedForSensor(Sensor sensor, String username) {
 // try {
 if (isSensorPublic(sensor) || sensor.getOwner().equals(username)) {
  return true;
 } else {
  // I check if one of the Users Group of the user is able
  // to access to the sensor
  Vector<UsersGroup> userGroups = getUsersGroups(username);
  Iterator<UsersGroup> it = userGroups.iterator();
  UsersGroup ug;
  while (it.hasNext()) {
   ug = it.next();
   ug.setAdmin(getNickname(ug.getAdmin()));
   if (isUsersGroupAllowedForSensor(sensor, ug)) {
    return true;
   }
  }
 }
 return false;
}

private Boolean isUsersGroupAllowedForSensor(Sensor sensor, UsersGroup ug) {
 return isUsersGroupAllowedForSensor(sensor.getName(), sensor.getOwner(), ug.getName(),
 ug.getAdmin());
}

public Boolean isUsersGroupAllowedForSensor(String sensorName, String owner, String
 groupName, String groupAdmin) {
 return isGroupAllowedForSensor(sensorName, owner, getUsersGroupId(groupName, groupAdmin),
   DatabaseDefines.ALLOWED_USERS_GROUPS);
}

private Boolean isGroupAllowedForSensor(String sensorName, String owner, ObjectId groupId,
 String field) {
  MongoCursor<Document> cursor = db.getCollection(DatabaseDefines.SENSORS_COLLECTION)
      .find(and(eq(DatabaseDefines.NAME, sensorName), eq(DatabaseDefines.OWNER,
 getUserId(owner)),
          eq(field, groupId)))
```

```
        .iterator();
    return cursor.hasNext();
}
```

4.3.3. Gestione delle chiavi

Per gestire al meglio le chiavi sono stati seguiti i seguenti accorgimenti:

- Dopo che si registra, l'utente crea una cartella, chiamata come il suo username, che contiene la chiave pubblica e privata RSA dell'utente e la chiave simmetrica AES. La chiave pubblica RSA del server non è in questa cartella ma è nella cartella superiore a questa, ovvero quella in cui sono contenute le varie cartelle degli username. Questo perché la chiave pubblica rsa del server è comune a tutti gli utenti;
- E' stata creata la classe KeysManager, che serve a gestire chiavi AES e RSA. Le chiavi vengono scritte e lette sui file, però quando viene fatta una operazione di questo tipo la chiave viene aggiunta al dizionario delle chiavi, una struttura dati che risiede in memoria e che, quindi, offre dei tempi di lettura più brevi di quelli del disco fisso creando, così, un meccanismo di caching, che evita di fare una lettura a file tutte le volte che serve una determinata chiave;
- Nel server, la chiave AES di ogni utente viene salvata nel database SQL come Blob, criptandola con la chiave pubblica RSA del server, in modo tale che possa essere decriptata solo con la chiave privata del server;
- Il server non salva la chiave pubblica del client, in quanto la utilizza solamente quando deve mandare la chiave simmetrica Aes al client e, siccome si assume che questa operazione avviene poche volte, è uno spreco di memoria salvarla e, dunque, si è preferito preferito mandarla ogni volta che il client richiede la propria chiave simmetrica.

4.4. Shell

Si riportano di seguito i menu che vengono presentati rispettivamente al client e all'amministratore del server, che fungono anche da riassunto delle funzionalità del sistema.

4.4.1. Lato client

```
Enter:
1 Login
2 Sign up
```

```
3 Quit
```

Login effettuato

```
Enter:
1 Get Measures
2 Services
3 Sensors
4 My servers
5 Users Groups
6 My Devices Groups
7 Logout
8 Delete Account
9 Quit
```

1 Get Measures

```
1 Get sensor measurements
2 Get sensors measurements by tags
3 Get sensors group measurements
4 Get sensors groups measurements by tags
5 Get sensors group measurements by position
```

2 Services

```
1 New service
2 Activate Service
3 Activate service by tags
4 Activate service by group tags
```

3 Sensors

```
1 Add allowed devices groups
2 Add allowed users groups
3 Add sensor to group
4 Add tag to sensor
5 Delete sensor from sensors group
6 Show my devices
7 Insert new sensor
8 Delete sensor
```

5 Users Groups

```
1 Add users to my group
2 Add tags to my group
3 Get the groups I belong
```

```
4 Get users of a group
5 New users group
6 Delete user from users group
7 Exit from group
8 Get my users groups
```

6 My Devices Groups

```
1 Add tags to actuators group
2 Add tags to sensors group
3 Add allowed users groups to actuators group
4 New Devices Group
5 Get my sensors group
6 Get my actuators group
```

4.4.2. Lato server

```
Enter:
1 Print Online users
2 Generate a new Key Pair
3 Print Server Key Pair
4 Set key size
5 Brands and Models
6 Quit
7 Sensors and Actuators
8 Users
9 Drop a collection
```

5 Brands and Models

```
1 Insert a new brand
2 Delete a brand
3 Insert a new model
4 Delete a model
```

7 Sensors and Actuators

```
1 Print Sensors Collection
2 Print Sensors group Collection
3 Print Actuators group Collection
```

8 Users

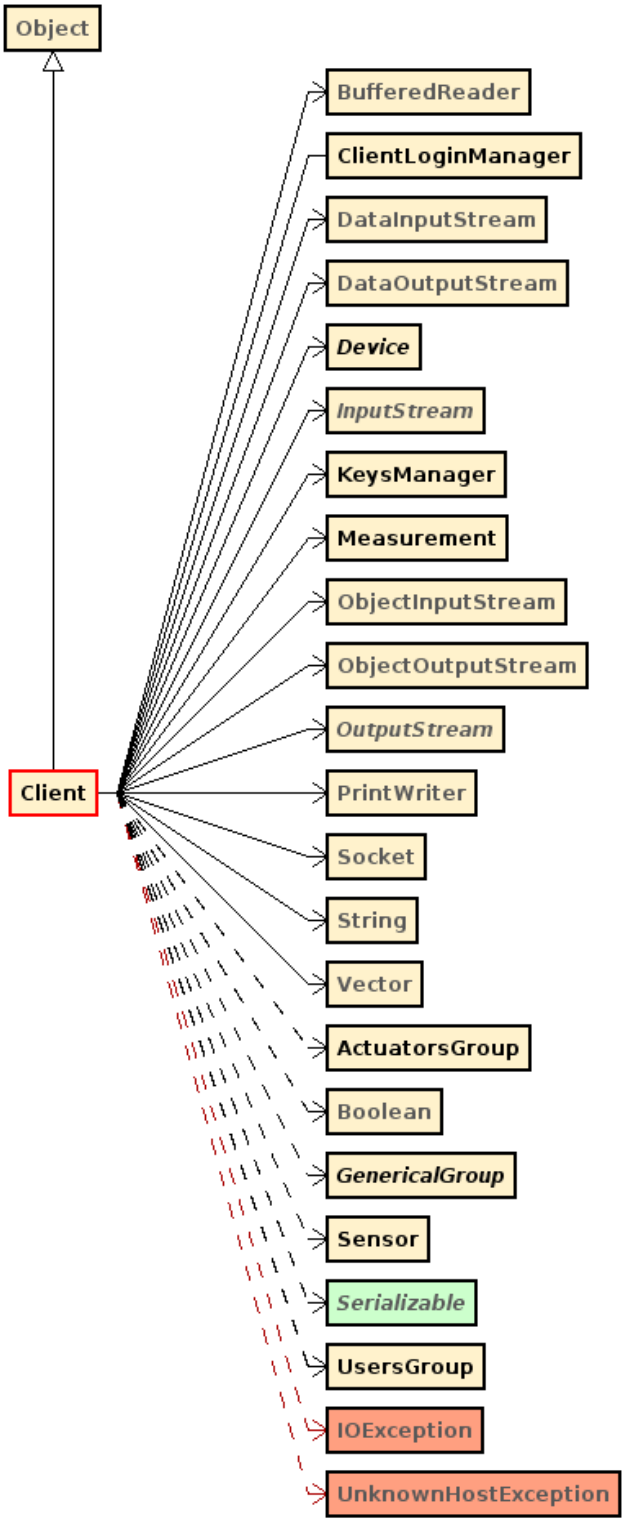
```
1 Print Sensors Collection
2 Print Sensors group Collection
3 Print Actuators group Collection
```

4.5. Diagramma delle classi

Di seguito vengono rappresentati i diagrammi e una descrizione delle classi di tutta la parte del codice in Java.

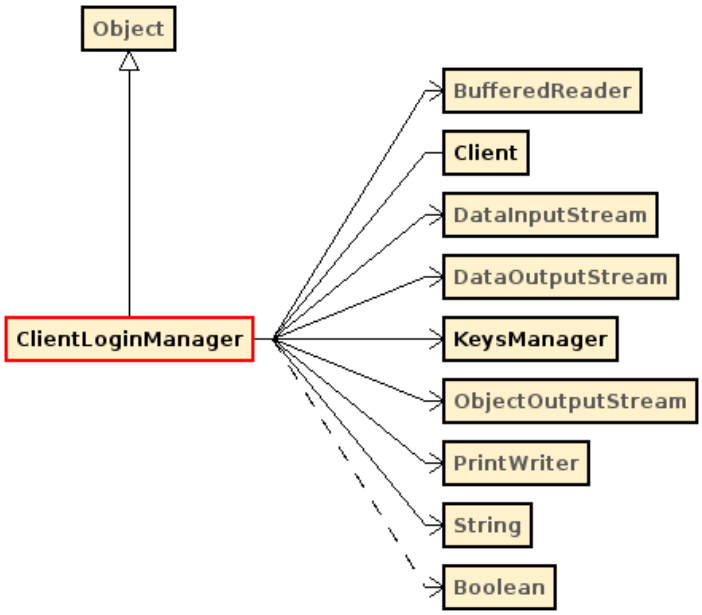
4.5.1. Client

Client
Si occupa dello scambio dei dati con il server e include molte funzioni per gestire l'input e l'output dell'utente.



ClientLoginManager

Si occupa delle operazioni di login, registrazione, logout e eliminazione dell'account e gestisce il nickname, la password e il path della chiave AES dell'utente.



4.5.2. Server remoto

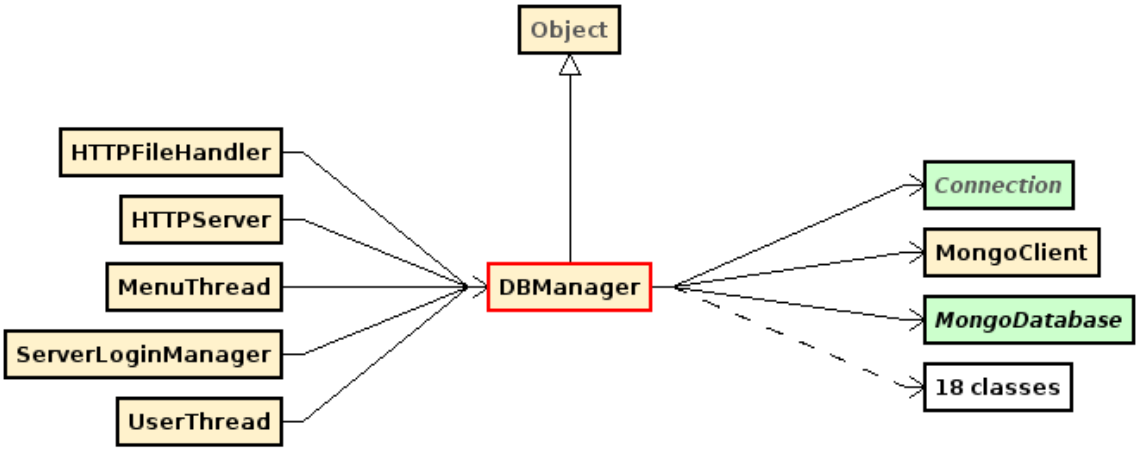
DBManager

Legge e scrive dai database SQL e NoSQL.

Le dipendenze della classe DBManager sono troppe per essere riportare nell'immagine e, quindi, vengono elencate qui di seguito:

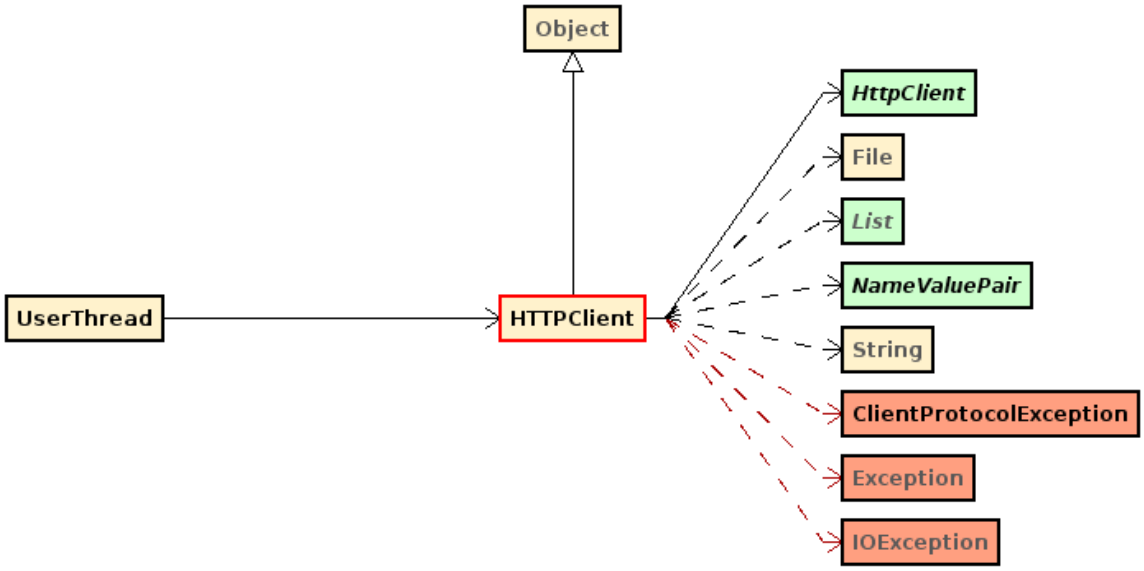
- ActuatorsGroup (groups.ActuatorsGroup)
- ArrayList<E> (java.util.ArrayList<E>)
- Boolean (java.lang.Boolean) «final»
- Device (devices.Device) «abstract»
- Document (org.bson.Document)
- Double (java.lang.Double) «final»
- GenericalGroup (genericalClasses.GenericalGroup) «abstract»
- GenericalServer (genericalClasses.GenericalServer)
- Measurement (devices.Measurement)
- MongoCursor<TResult> (com.mongodb.client.MongoCursor<TResult>) «abstract» «interface»

- ObjectId (org.bson.types.ObjectId) «final»
- RSAPrivateKey (java.security.interfaces.RSAPrivateKey) «abstract» «interface»
- RSAPublicKey (java.security.interfaces.RSAPublicKey) «abstract» «interface»
- Sensor (devices.Sensor)
- SensorsGroup (groups.SensorsGroup)
- String (java.lang.String) «final»
- UsersGroup (genericalClasses.UsersGroup)
- Vector<E> (java.util.Vector<E>)



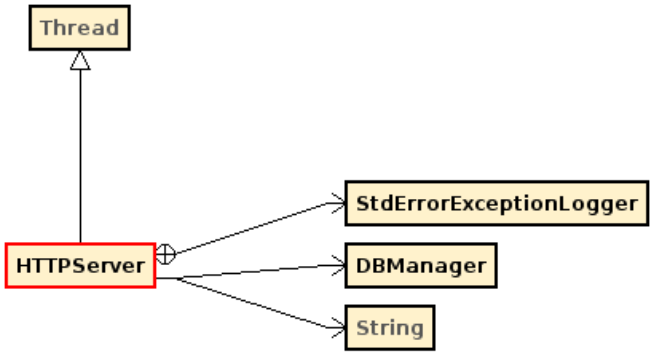
HTTPClient

Fornisce funzioni per mandare richieste a un server HTTP. E' utilizzata quando bisogna richiedere l'attivazione di un servizio o quando bisogna registrarne uno nuovo.



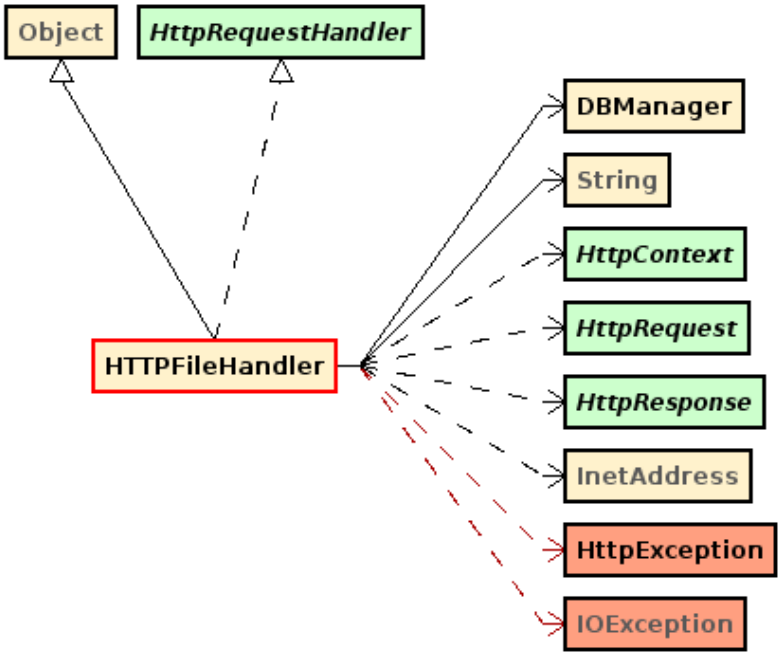
HTTPServer

Riceve dai server CoAP tutte le misurazioni effettuate dai sensori.



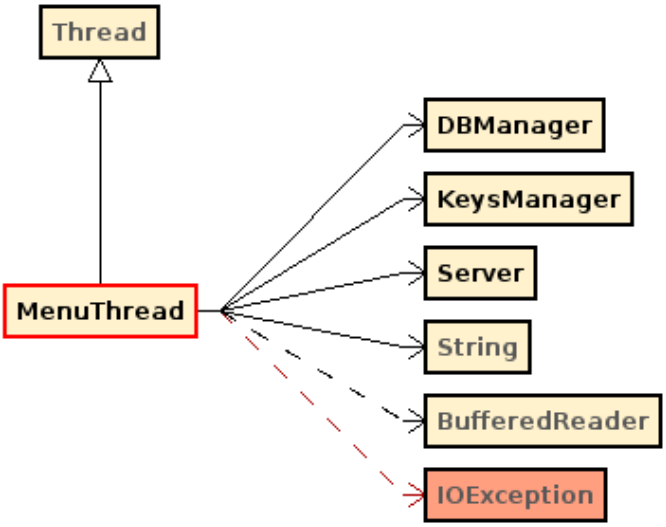
HTTPFileHandler

Si occupa di gestire le richieste che arrivano alla classe HTTPServer.



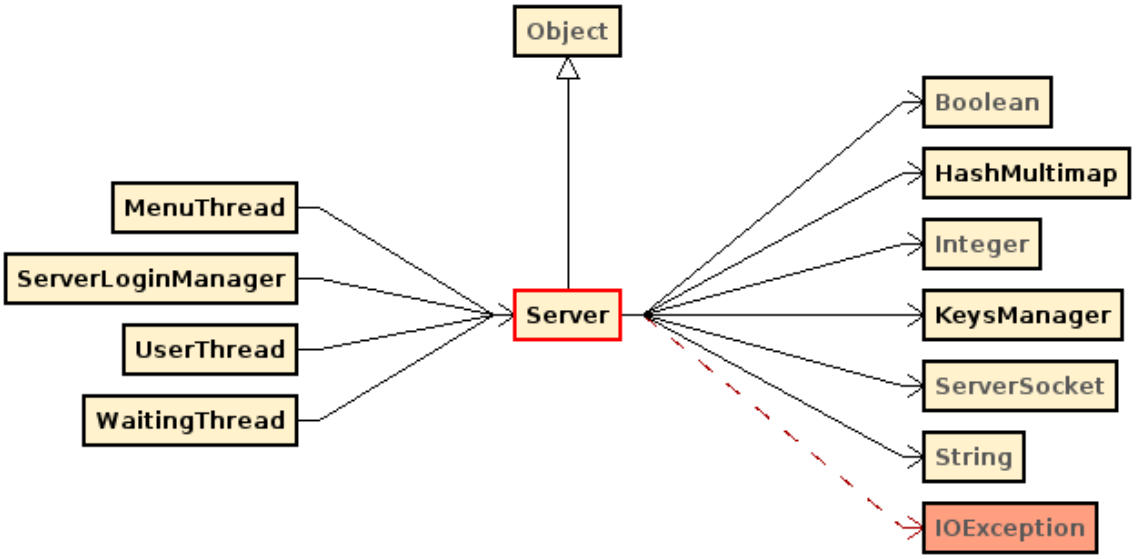
MenuThread

Si occupa dell'interazione con l'admin del server remoto, permettendogli di fare operazioni come visualizzare o eliminare collection dal database NoSQL o visualizzare gli utenti Online.



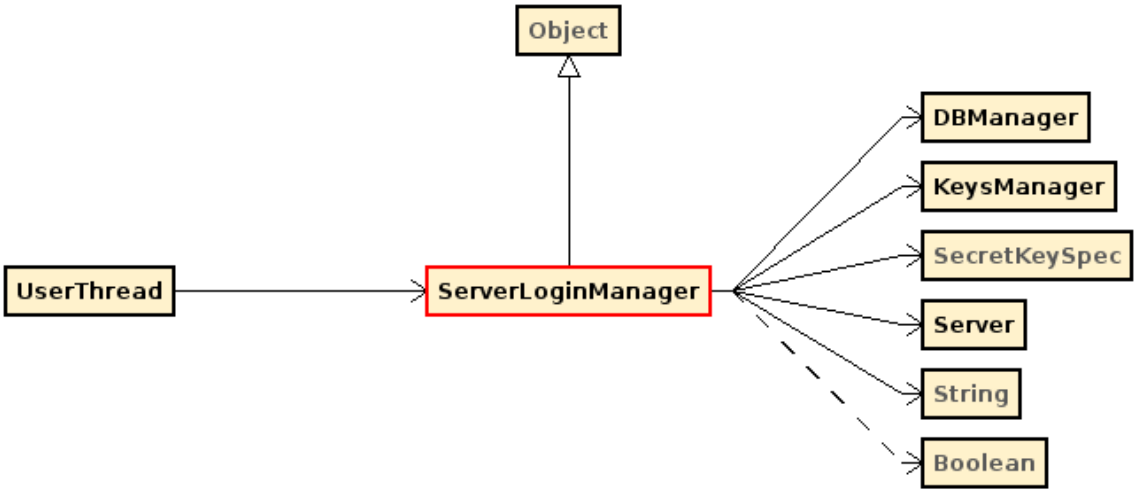
Server

Istanzia un nuovo UserThread per ogni client che vuole collegarsi e tiene traccia di tutti gli utenti connessi.



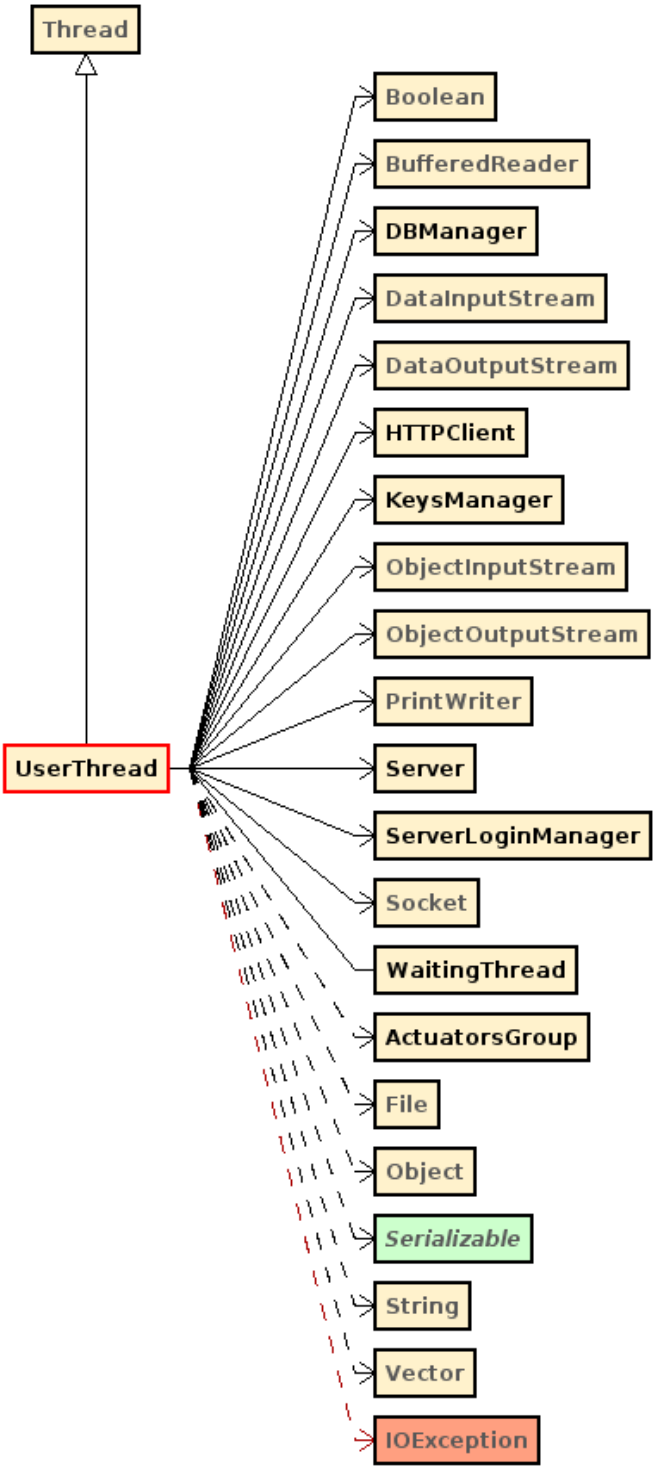
ServerLoginManager

Si occupa delle operazioni di login, registrazione, logout e eliminazione dell'account e gestisce il nickname, la password e la chiave AES dell'utente.



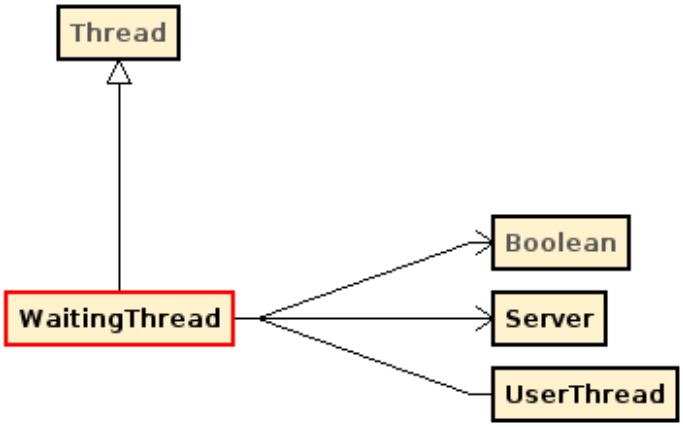
UserThread

Si occupa dello scambio dei dati con il client e interagisce con il DBManager e l'HTTPClient per soddisfare le richieste dell'utente.



WaitingThread

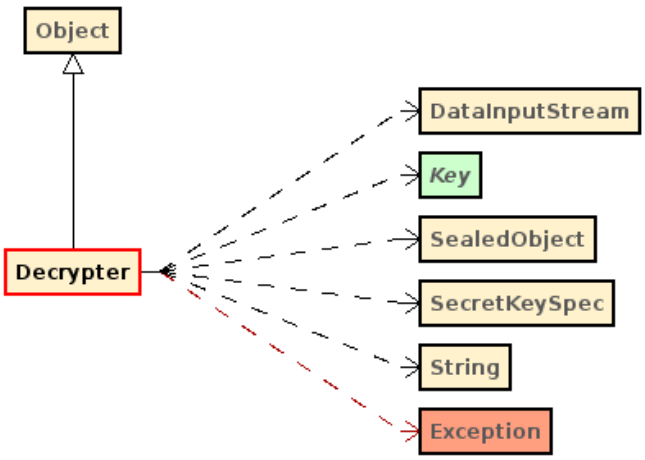
Impedisce al client di tenere bloccata la chiave per troppo tempo.



4.5.3. Crittografia

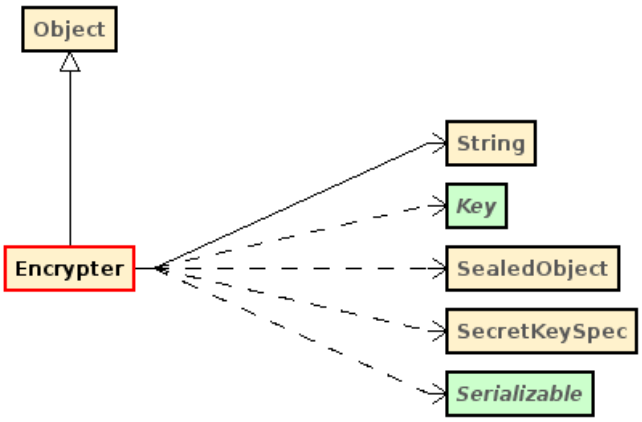
Decrypter

Decripta oggetti e stringhe criptate con gli algoritmi RSA o AES



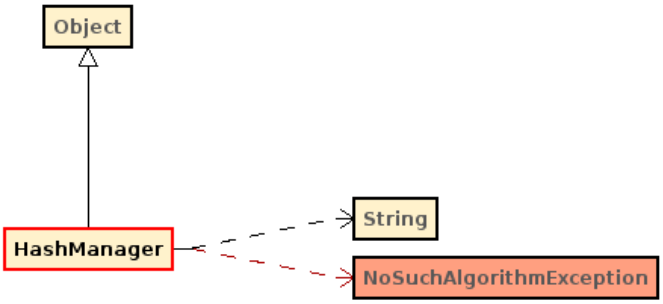
Encrypter

Cripta oggetti e stringhe con gli algoritmi RSA o AES



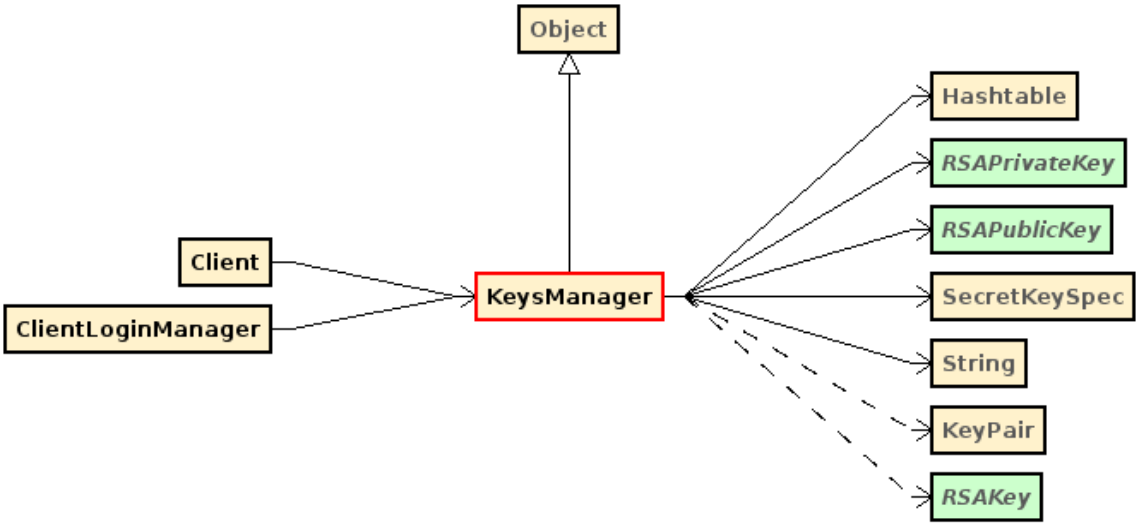
HashManager

Calcola un valore hash con l'algoritmo SHA-256 di una stringa in ingresso. E' utilizzato per salvare le password degli utenti nel database.



KeysManager

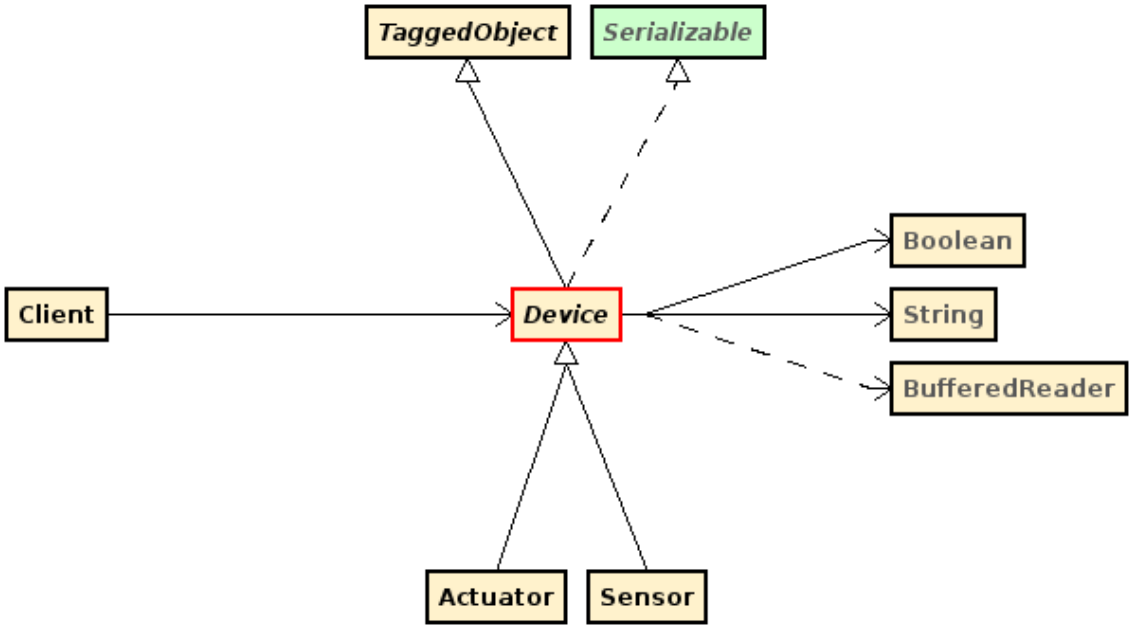
Si occupa della gestione delle chiavi RSA e AES.



4.5.4. Dispositivi

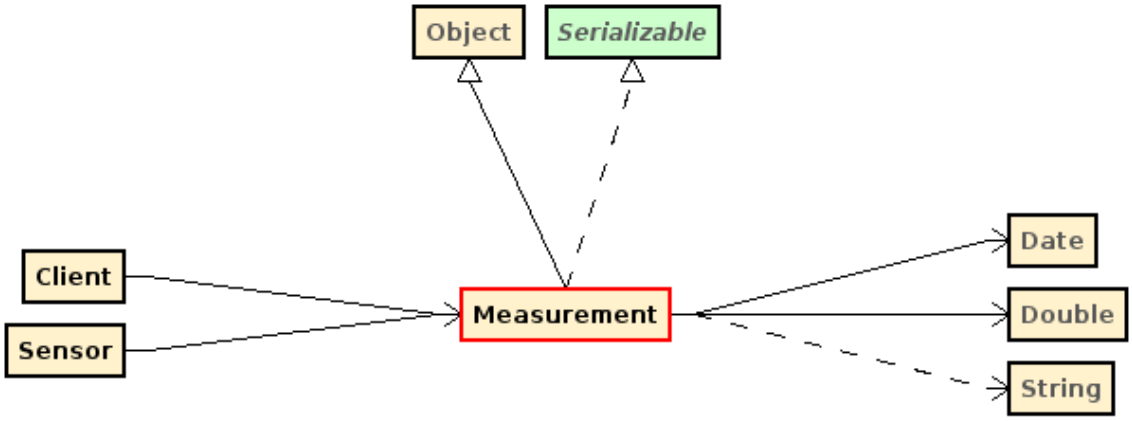
Device

Classe astratta che raccoglie gli attributi e i metodi comuni a tutti i dispositivi



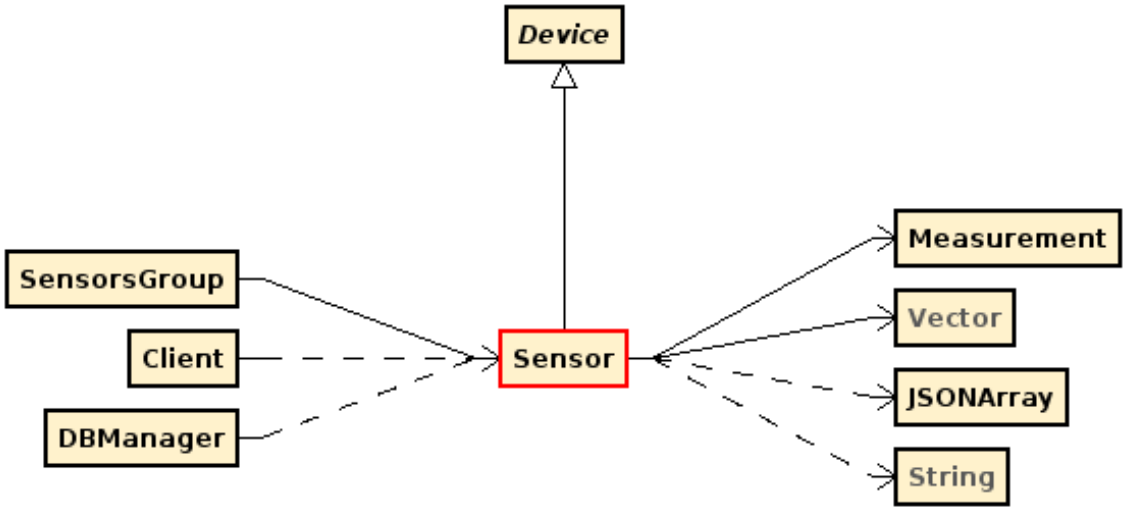
Measurement

Rappresenta una misura di un sensore



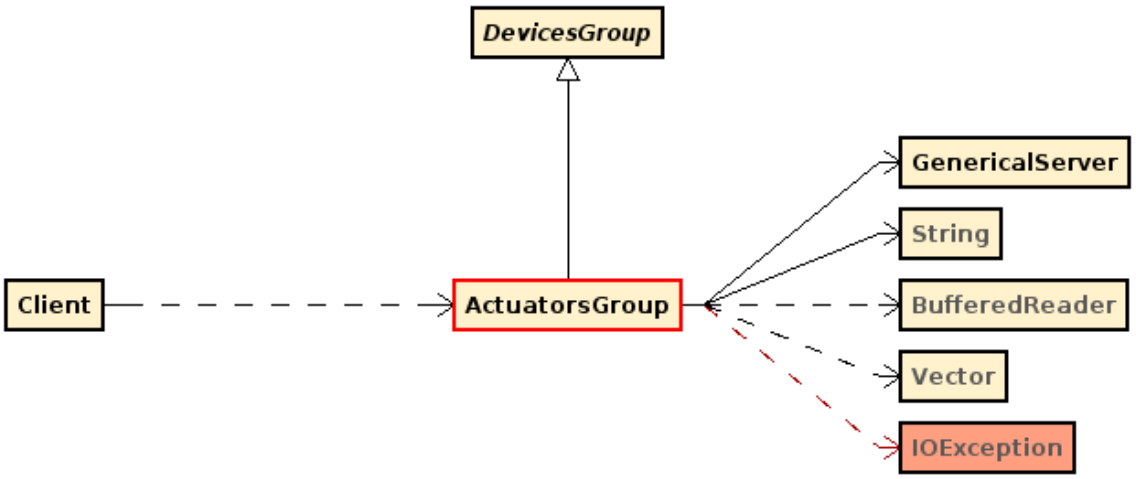
Sensor

Rappresentazione dei sensori. E' presente un costruttore che permette di creare un nuovo sensore a partire da un documento Json.



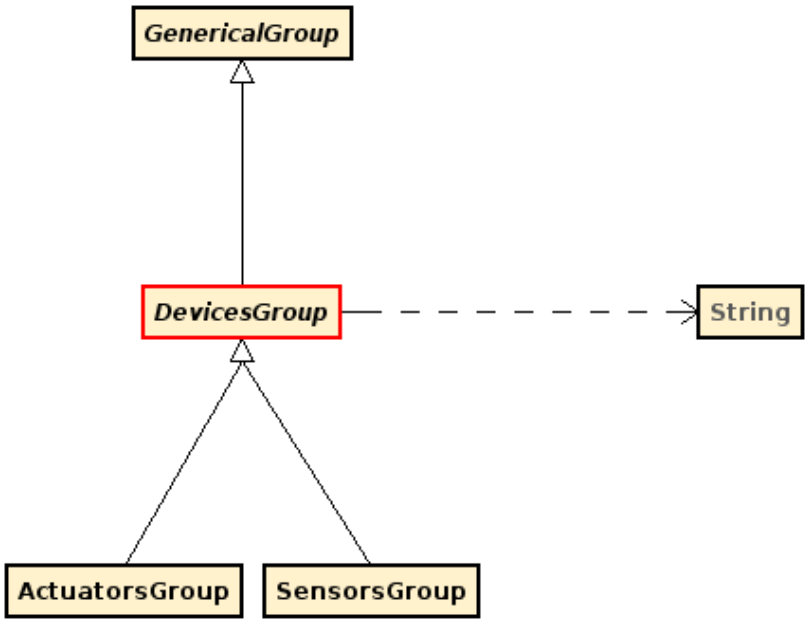
ActuatorsGroup

Rappresentazione dei gruppi di attuatori. E' presente un costruttore che permette di creare un nuovo gruppo di attuatori a partire da un documento Json.



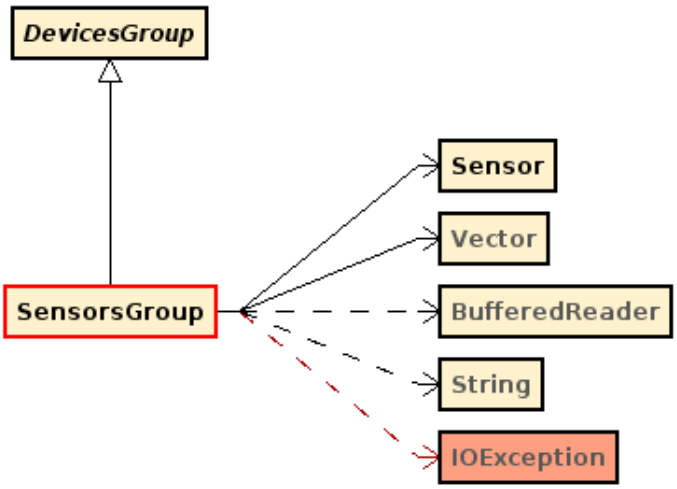
DevicesGroup

Classe astratta che raccoglie gli attributi e i metodi comuni a tutti i gruppi di dispositivi.



SensorsGroup

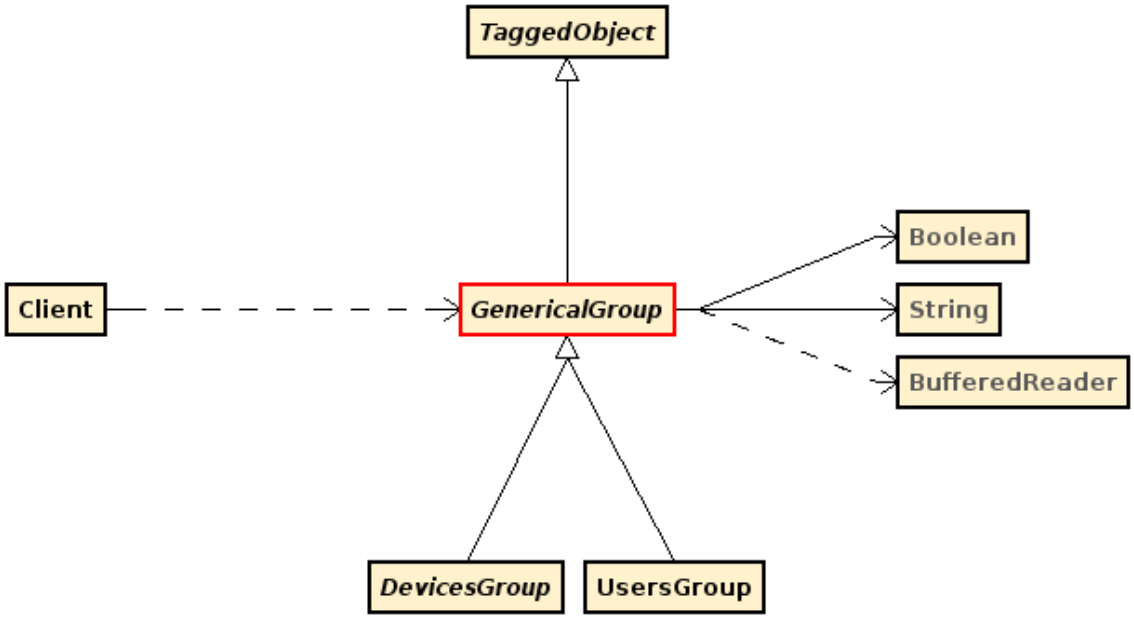
Rappresentazione dei gruppi di sensori. E' presente un costruttore che permette di creare un nuovo gruppo di sensori a partire da un documento Json.



4.5.5. Varie

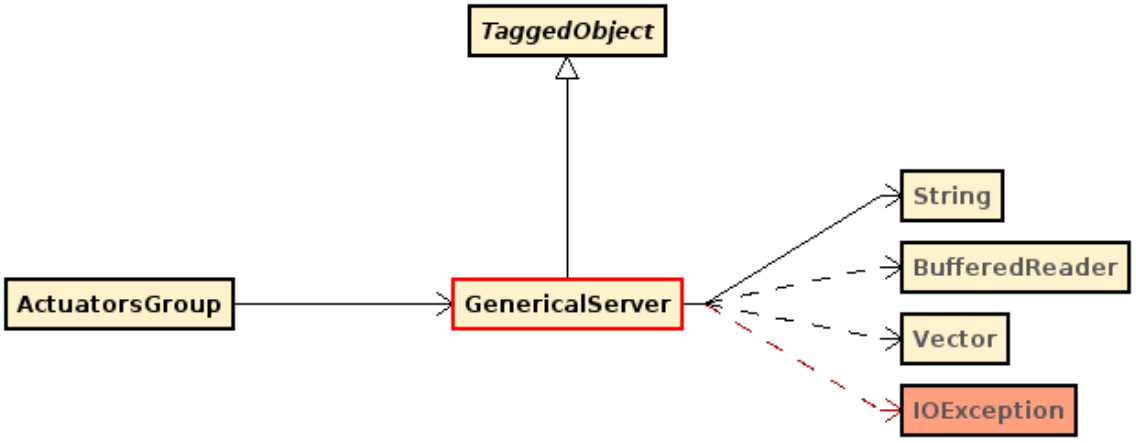
GenericalGroup

Classe astratta che raccoglie gli attributi e i metodi comuni a tutti i gruppi, come il nome del gruppo, l'admin o il gruppo genitore.



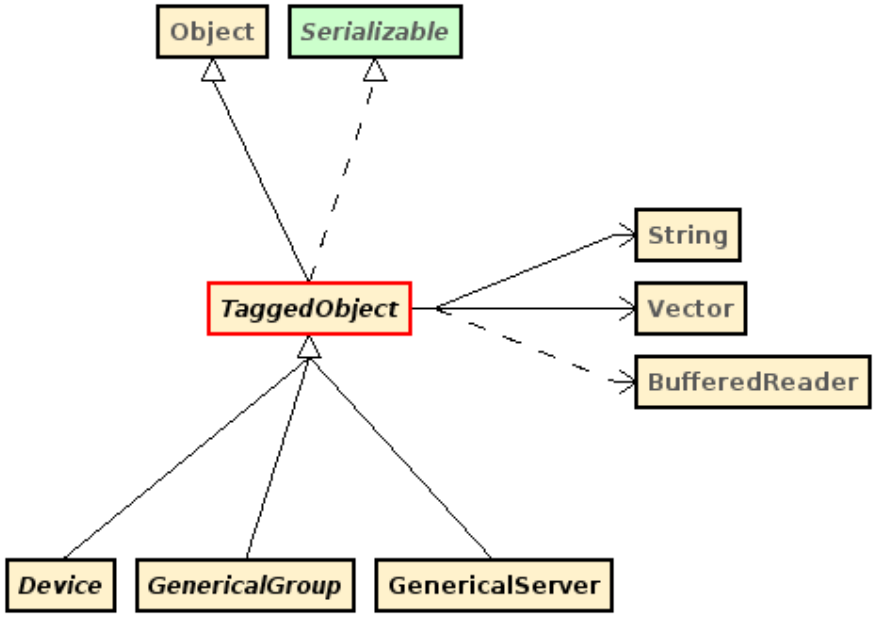
GenericalServer

Rappresentazione di un server generico.



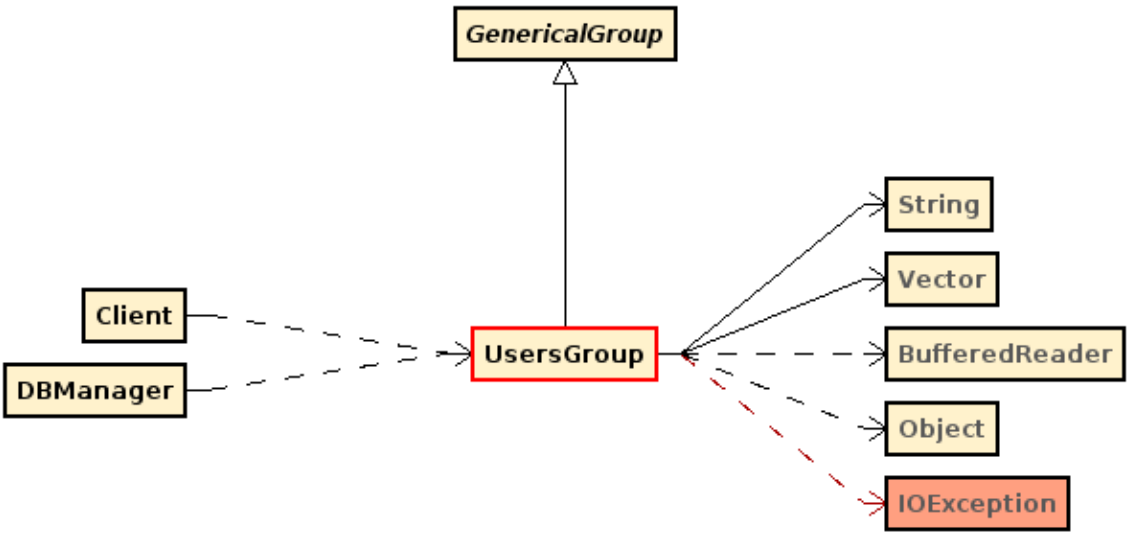
TaggedObject

Classe astratta che fornisce tutti gli strumenti per gestire i tag. Viene estesa da tutti gli oggetti che presentano tag.



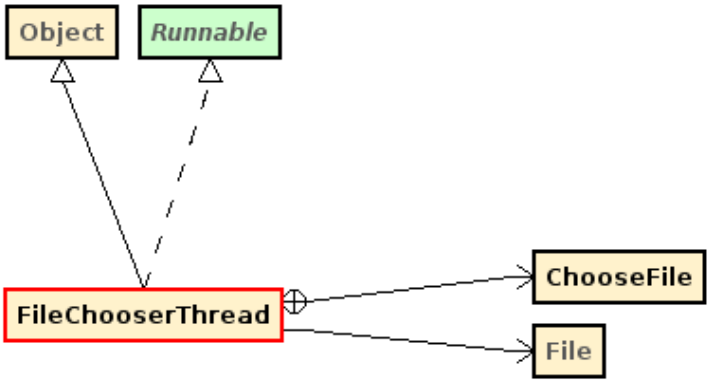
UsersGroup

Rappresentazione dei gruppi di utenti. E' presente un costruttore che permette di creare un nuovo gruppo di utenti a partire da un documento Json.



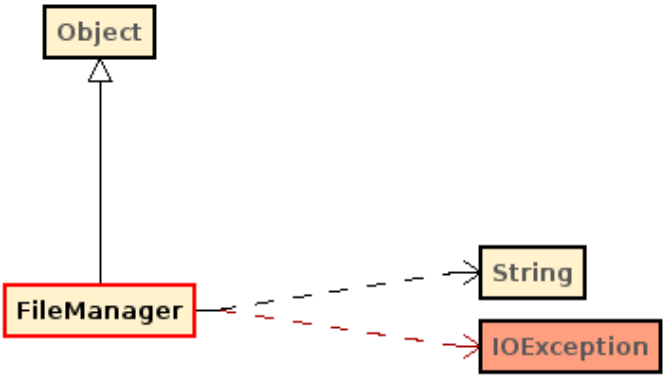
FileChooserThread

Thread che permette di selezionare una directory all'interno del file system mediante una interfaccia user-friendly.



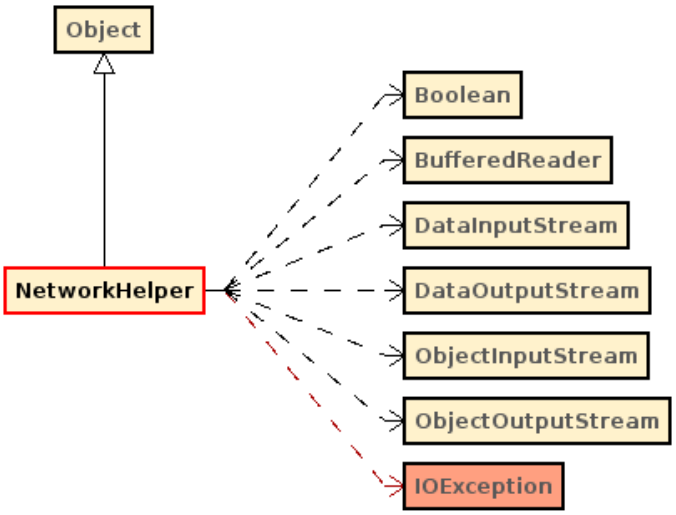
FileManager

Permette di leggere e scrivere oggetti da e su file.



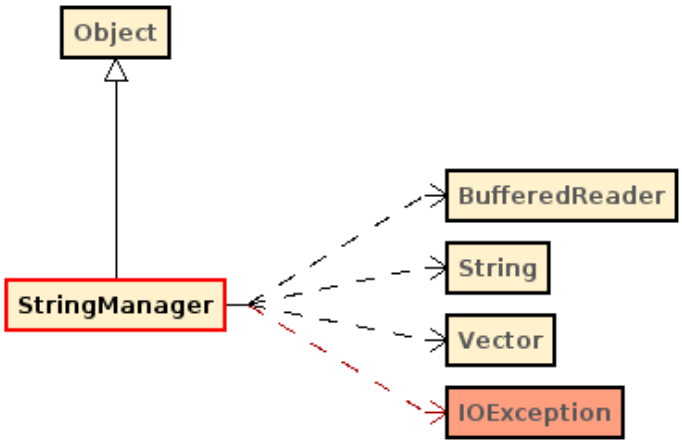
NetworkHelper

Fornisce funzioni utili a livello di rete, come mandare e ricevere bytes e oggetti.



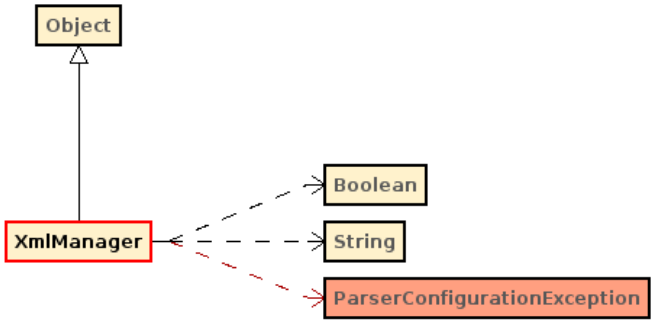
StringManager

Fornisce funzioni utili per la gestione delle stringhe, come funzioni per eseguire il parsing di un documento Json trattato come una stringa.



XmlManager

Fornisce funzioni utili per modificare e leggere un file XML.



4.6. Ulteriori dettagli

Si riportano di seguito altri dettagli implementativi. Se qualcosa non è stato trattato in questo capitolo si invita a consultare l'appendice, in cui è presente tutto il codice scritto per questo progetto.

4.6.1. Installazione delle dipendenze

Per installare le dipendenze richieste da python e dal coapthon sono state eseguite le seguenti istruzioni:

```
$ sudo sudo apt-get install libmysqlclient-dev
$ sudo easy_install MySQL-python
$ sudo easy_install Twisted sudo apt-get install python-sphinx
$ sudo easy_install bitstring sudo easy_install futures
```

4.6.2. Correzione della gestione del log

Nel coap protocol si è modificata la gestione del log per una corretta simulazione di più componenti del sistema(interruttori, server, attuatori, ecc.) su una sola macchina. Prima, infatti, tutti i componenti scrivevano sullo stesso file. In particolare, all'interno di ogni progetto nei defines, si è aggiunto il campo COAP_LOG_DIRECTORY, che ora specifica qual è il file log di ogni progetto, che specifica la cartella di ogni file di log per ogni componente.

4.7. Strumenti utilizzati

Per realizzare questa tesi sono stati utilizzati solamente programmi free e, per la maggior parte, open source, ovvero:

Eclipse
: Per la programmazione in Java e in Python. Sono stati installati, inoltre, i plugin PyDev, essenziale per python e M2Eclipse, per integrare maven in eclipse.

Maven
: Per risolvere tutte le dipendenze esterne senza scaricare manualmente le librerie

Asccidoc e Asciidoctor
: Il primo come formato della tesi e il secondo per convertirla in PDF

Atom e Gedit
: Come editor di testo per modificare la tesi

yED
: Per la creazione degli schemi

https://www.websequencediagrams.com/
: Per la creazione dei sequence diagram

Class Visualizer
: Per la creazione dei diagrammi delle classi

Capitolo 5. Sperimentazione

Si ripropone qui di seguito un esempio utilizzato per testare il sistema, in cui quest'ultimo è stato utilizzato per creare uno Smart Building Activity Based.

5.1. Scenario di riferimento

Lo scenario di riferimento è quello di un edificio di un campus, di cui si vuole gestire l'impianto di riscaldamento in maniera intelligente.

La struttura dell'edificio è la seguente:

- vi sono 4 piani;
- delle scale dividono l'edificio in due blocchi;
- un piano e un blocco identificano una zona, che è costituita da un corridoio, delle stanze e un bagno.

I vari spazi dell'edificio sono suddivisi nel seguente modo:

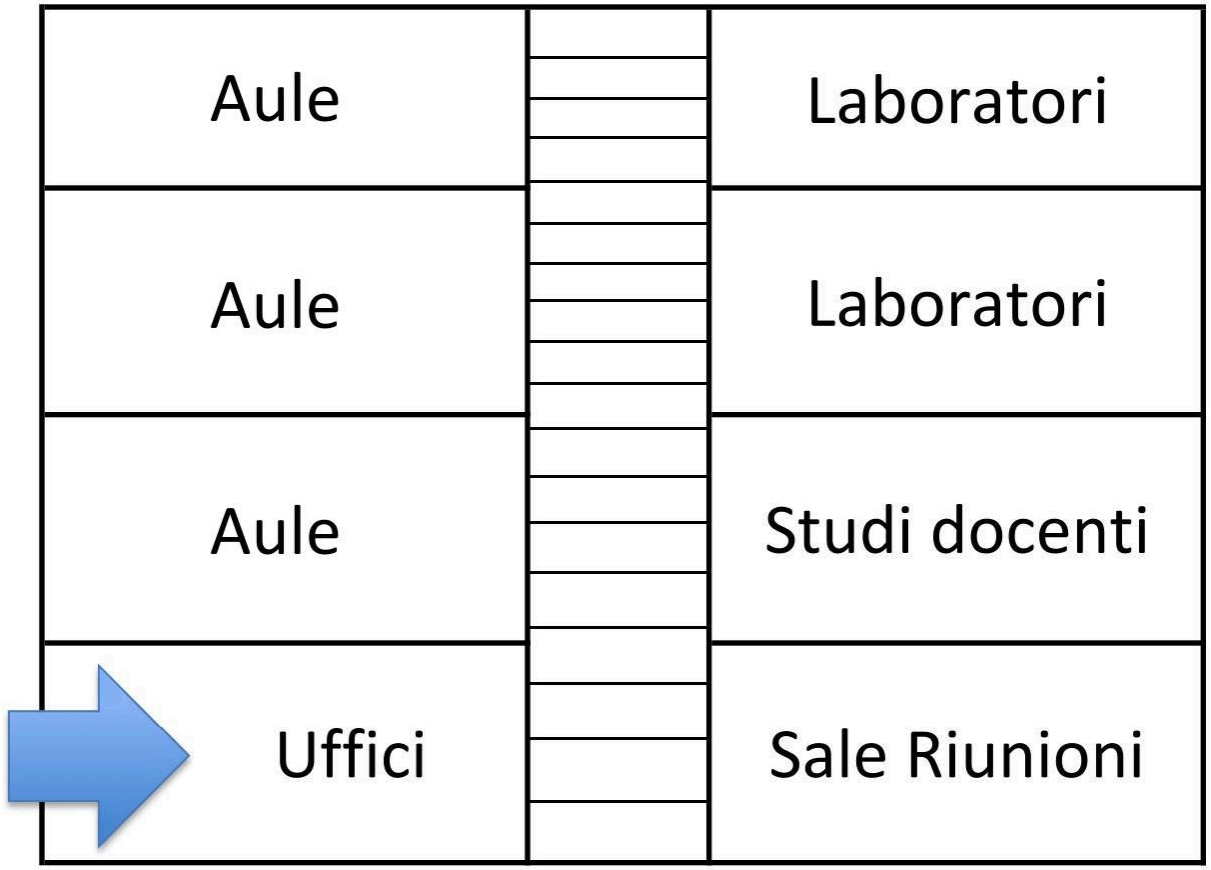


Figura 5.1. Struttura dell'edificio

La vista in pianta di una singola zona, invece, è la seguente:

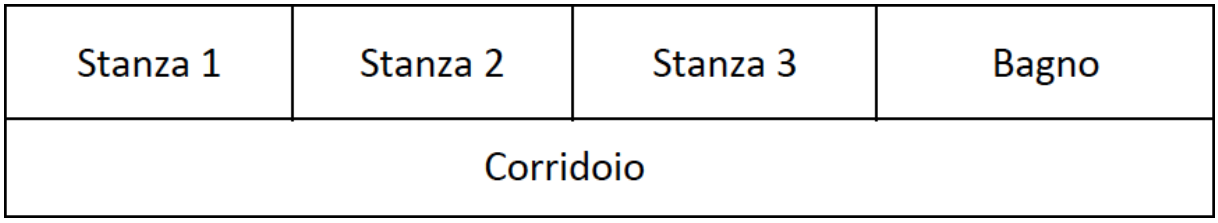


Figura 5.2. Pianta di una zona

Gli oggetti che consentono il riscaldamento intelligente dell'edificio sono:

- sensori di temperatura e umidità;
- fonti di calore comandabili dall'esterno;
- interruttori, che vengono attivati quando vi è la presenza di almeno una persona nell'ambiente.

I sensori e le fonti di calore sono posizionate in tutti gli ambienti, ovvero nei corridoi, nelle stanze e nelle scale e possono essere presenti in numero variabile a seconda delle esigenze dello specifico ambiente, mentre in ogni stanza è presente un solo interruttore.

5.2. Obiettivo

L'obiettivo è quello di mantenere le persone all'interno dell'edificio in condizioni termiche di benessere, consumando meno energia possibile, ovvero attivando solamente le fonti di calore necessarie.

Ad esempio, sapendo che nel blocco B, al secondo piano in un laboratorio c'è un gruppo di persone al lavoro, l'impianto di riscaldamento dovrà permettere l'attivazione ed il controllo delle fonti di calore posizionate nel corridoio e nei bagni degli uffici, nelle scale fino al secondo piano, nel corridoio e nei bagni dei laboratori del secondo piano e nel laboratorio dove vi sono le persone che stanno lavorando.

5.3. Soluzione sviluppata

Per risolvere la problematica sopra descritta, all'interno dell'edificio è inserito un server CoAP, a cui i sensori di temperatura e umidità mandano le loro misurazioni e i vari interruttori che segnalano la presenza delle persone richiedono l'attivazione di un servizio, a seconda se essi vengono attivati o disattivati. In base al servizio richiesto

e alle misure dei sensori, il server invia il segnale di attivazione o spegnimento a determinate fonti di calore che, quindi, rappresentano gli attuatori del sistema.

Queste ultime sono organizzate in gruppi multicast, dunque il server avrà il vantaggio di interfacciarsi con esse e non con le singole fonti di calore, portando ad un'efficiente gestione dell'intero sistema, riducendo i ritardi ed ottimizzando le prestazioni del sistema di comunicazione.

Il comportamento di ogni attuatore all'interno del gruppo viene personalizzato mediante l'ausilio dei tag, che verranno inviati dal server CoAP a seconda della temperatura e dell'umidità presenti all'interno dell'ambiente.

Infatti, se il server dall'interruttore che richiede il servizio deduce a quali gruppi accedere, dalle misurazioni dei sensori ricava quali tag mandargli, poichè questi ultimi vengono scelti in base alle condizioni climatiche dell'ambiente in cui il gruppo di attuatori è inserito.

I tag possibili per gli attuatori sono "cold", "cold-warm", "warm", "warm-hot", "hot". Predisponendo i vari attuatori con una combinazione di questi vari tag, è possibile ottenere la giusta quantità di fonti di calore attive in base alla necessità climatica del momento, che viene fornita dai sensori.

Per capire quali gruppi di attuatori attivare, invece, viene sfruttata la posizione dell'interruttore dell'illuminazione elettrica. Quest'ultimo, infatti, viene attivato quando è presente qualcuno nella stanza e viene disattivato quando non vi è nessuno.

Sfuttando la posizione dell'interrutore, dunque, è possibile decidere quali gruppi di attuatori attivare, perchè da essa vengono dedotti tutti gli ambienti in cui le persone che si trovano in quella stanza vorrebbero accedere e, in particolare, si stabilisce che per delle persone che si trovano in una determinata stanza bisogna riscaldare tutto il percorso che porta a quella determinata stanza e il bagno della zona della stanza.

Il percorso che porta ad una stanza è costituito da:

- il corridoio di ingresso (che è quello presente nella zona degli uffici);
- le scale che portano fino al piano della stanza;
- il corridoio della zona della stanza.

In base a queste caratteristiche è stato sviluppato un albero che rappresenta la struttura dell'edificio:

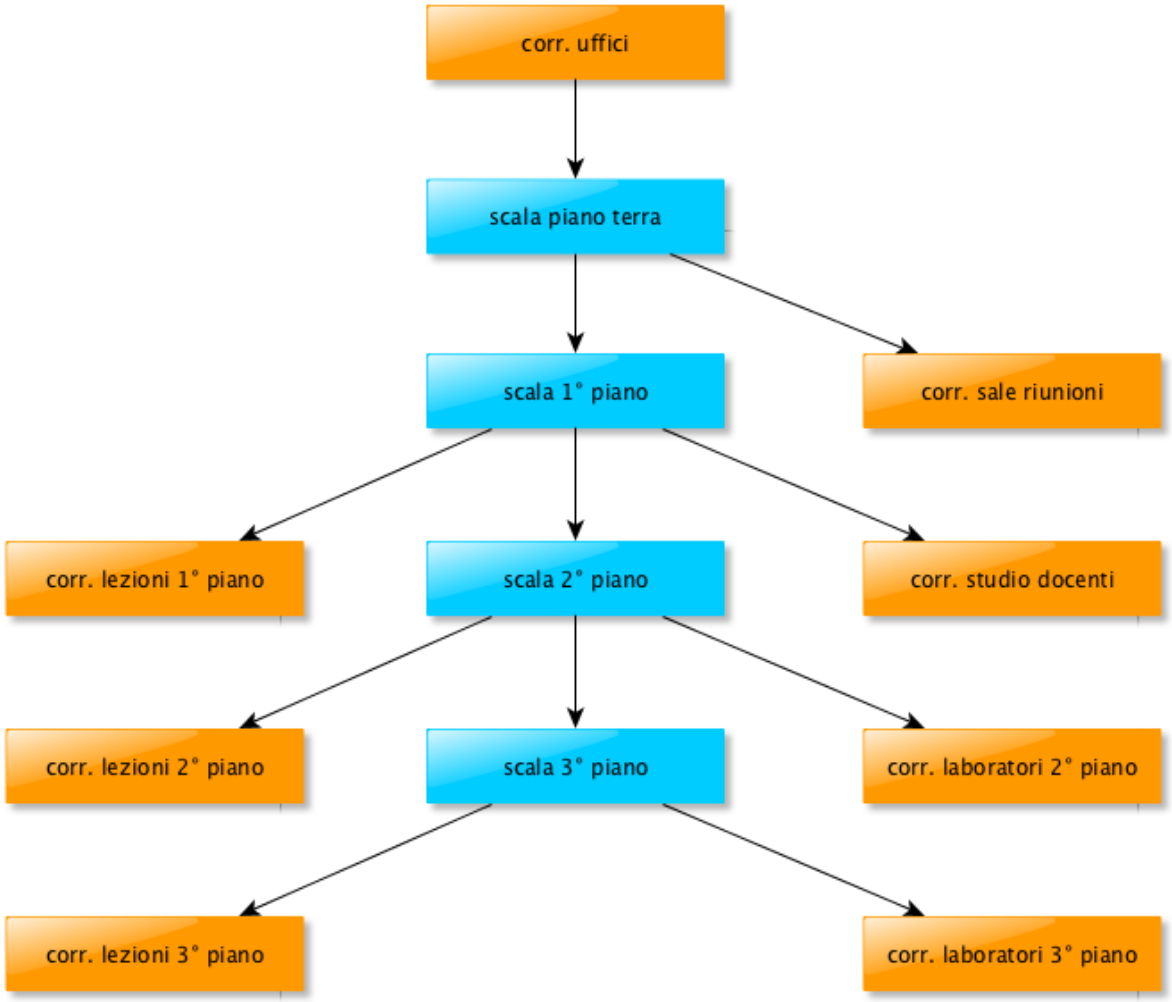


Figura 5.3. Gerarchia dell'edificio

Nello schema in Figura 5.3 mancano le stanze. Esse sono state omesse per garantire la leggibilità della figura, ma bisogna considerare che i gruppi in arancione hanno come sottogruppi tutte le stanze che il corridoio contiene.

Non sono da considerare invece i bagni, semplicemente perchè essi non hanno bisogno di una trattazione diversa rispetto ai corridoi in quanto, in tutti i possibili scenari che possono presentarsi, i riscaldamenti dei bagni vanno sempre attivati insieme a quelli dei corridoi e quindi appartengono allo stesso gruppo multicast.

Quando un interruttore di illuminazione di una stanza viene attivato, dunque, il server parte dal nodo della stanza in cui era presente l'interruttore e ripercorre la gerarchia in Figura 5.3 verso l'alto attivando, dunque, il nodo stesso e tutti i suoi antenati.

Ad esempio, se venisse attivato l'interruttore appartenente a un laboratorio appartenente ai laboratori del secondo piano verrebbero attivati i gruppi appartenenti a:

- il laboratorio stesso;
- il corridoio dei laboratori del secondo piano;
- la scala al secondo piano;
- la scala al primo piano;
- la scala al piano terra;
- il corridoio degli uffici.

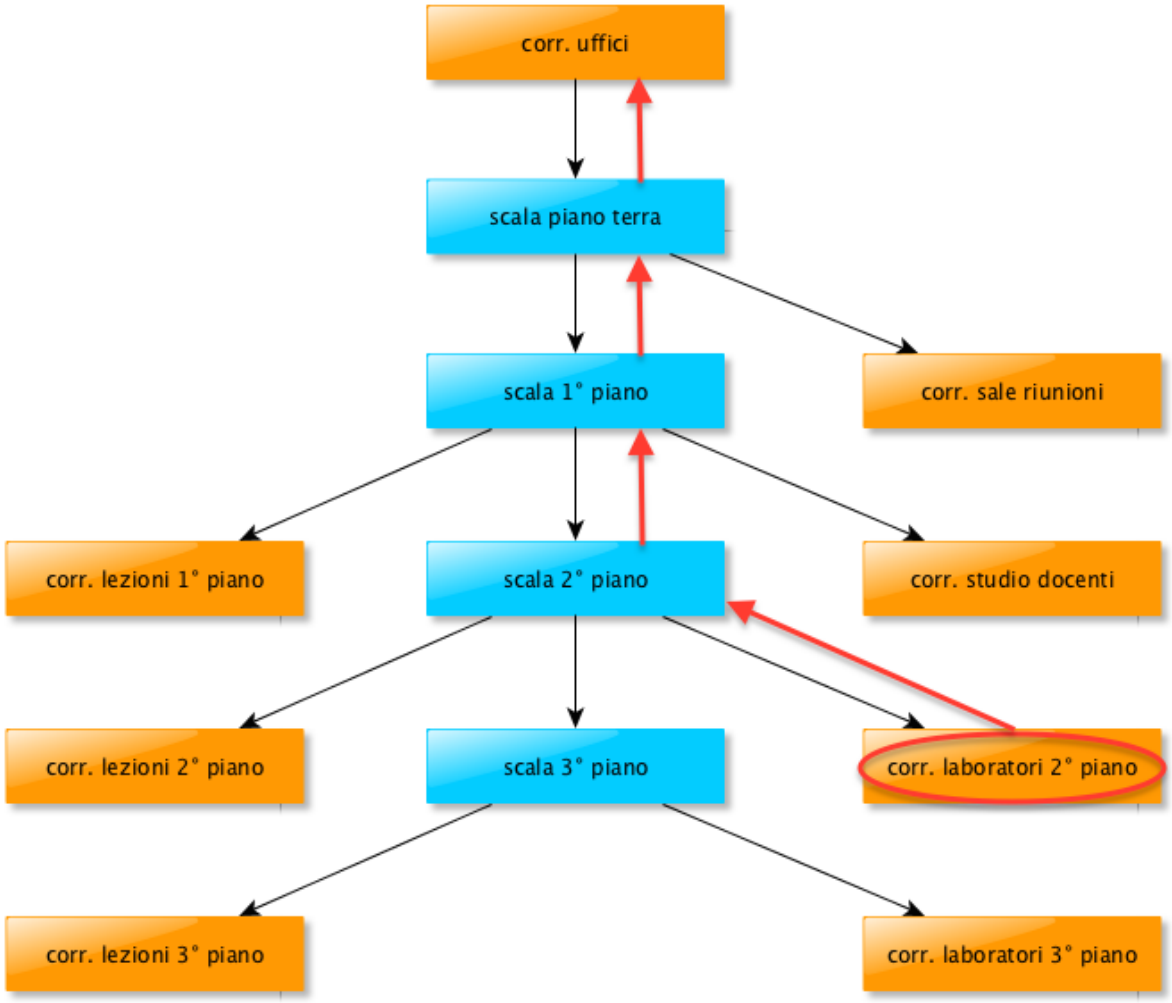


Figura 5.4. Percorso della gerarchia

Quando un interruttore viene disattivato, invece, il server manda il comando a quel gruppo di attuatori di disattivarsi, in modo che le fonti di calori di quella stanza si spengano e ripercorre, come in precedenza, la gerarchia verso l'alto e controlla se esiste almeno un figlio attivo per ogni nodo nell'albero mostrato in Figura 5.3. Se quest'ultima condizione non è vera, allora il server manda il comando di spegnersi anche a quel gruppo, perchè significa che non vi sono persone che stanno in quelle zone o che devono transitarvi.

I sensori devono appartenere a gruppi di sensori che devono essere chiamati come i gruppi di attuatori appartenenti agli attuatori che si trovano nello stesso ambiente dei sensori stessi. In questo modo si capisce il clima dell'ambiente di ogni gruppo di fonti di calore.

5.4. Possibili scenari

Gli scenari che sono stati considerati nella sperimentazione di questa tesi sono descritti nelle sezioni seguenti.

5.4.1. Mattinata fredda infrasettimanale

Tutte le zone dell'università sono utilizzate, tranne la zona delle riunioni. Nelle sale riunioni, dunque, gli interruttori saranno spenti e dunque il server non accenderà né le singole sale nè il corridoio che porta ad esse nè il bagno della zona riunioni.

I sensori delle varie zone, inoltre, indicano che ci sono una temperatura e un'umidità molto al di sotto della temperatura ottimale e, dunque, sarà necessario riscaldare velocemente gli ambienti. Il server manderà il tag "cold" ai vari gruppi di attuatori che si trovano nelle zone interessate dalla presenza di studenti, professori e personale tecnico-amministrativo. Se gli attuatori sono stati impostati in maniera adeguata, tutti sono configurati con il tag "cold" e iniziano a lavorare ad alto regime.

5.4.2. Giornata di sola attività amministrativa

In giornate particolari, come ad esempio il 28, il 29 e il 30 Dicembre, gli unici a lavorare all'interno del campus è il personale tecnico-amministrativo. In queste giornate vi è il maggior risparmio di energia possibile (escludendo quelle dove l'università è completamente chiusa ovviamente), poichè basterà riscaldare solamente la zona degli uffici, ovvero il bagno della zona degli uffici, il corridoio d'ingresso e gli uffici.

Dalle altre stanze, infatti, gli interruttori risulteranno disattivati e, dunque, non vi sarà nessun gruppo ad essere attivato oltre a quelli citati precedentemente

5.4.3. Riunione di dipartimento in una giornata calda

Durante la riunione di dipartimento si presume che la maggior parte delle persone siano impegnate nelle sale riunioni, dunque gli interruttori di queste sale saranno accesi. Si ipotizzi però che vi siano dei ricercatori non impegnati in sala riunione e che si trovano in un laboratorio al terzo piano. In questo caso i gruppi di attuatori attivati saranno quelli appartenti ai seguenti ambienti:

- il corridoio degli uffici;
- il bagno degli uffici;
- le scale fino al terzo piano;
- il corridoio che conduce alle sale riunioni;
- le sale riunioni impegnate;
- il bagno delle sale riunioni;
- il laboratorio impegnato dai ricercatori;
- i bagni della zona dei laboratori del terzo piano.

Riducendo il numero di ambienti riscaldati, c'è un risparmio in termini di fonti di energia globale utilizzata per il riscaldamento dell'edificio.

Se la giornata è abbastanza calda, quindi il server manda ai gruppi il tag "hot". Se gli attuatori sono stati impostati in maniera adeguata pochi attuatori hanno il tag "hot", poichè in questa situazione non c'è bisogno di riscaldare molto e, dunque, si ha la necessità di poche fonti di calore attive.

5.5. Inserimento dei dati nel sistema

Una volta che sono stati identificati e caratterizzati tutti gli attuatori dell'edificio, questi vengono organizzati in gruppi. I gruppi di attuatori vengono inseriti nel sistema tramite la voce di menù "New Actuators Group", che si trova all'interno del menù "My devices Groups" - "New Devices Group". In particolare, rispetto ai nomi presenti nella gerarchia di Figura 5.3 sono stati eliminati i punti e gli spazi sono stati sostituiti da un underscore, quindi ad esempio "corr. uffici" è diventato "corr_uffici".

L'inserimento del gruppo "scala piano terra", per esempio, è il seguente:

```
name: scala_piano_terra
public [Y/n]: y
description (optional): il gruppo che contiene tutti gli attuatori presenti nella scala del
 piano terra
parent group name (optional): corr_uffici
tag (optional):

IP(optional) : 228.5.6.9
server data:
IP: 127.0.0.1
port: 8081
```

Il gruppo nell'esempio ha il campo tag vuoto, in quanto i tag caratterizzano i singoli attuatori e non il gruppo a cui appartengono, poichè lo scopo di utilizzare i tag è proprio

quello di ottenere un comportamento differenziato tra gli attuatori appartenenti allo stesso gruppo.

Ai fini della nostra sperimentazione, oltre ai gruppi visibili nella gerarchia, sono stati inseriti i gruppi "ufficio1" e "ufficio2" come sottogruppi del gruppo "corr_uffici" e il gruppo "laboratorio1" come sottogruppo del gruppo "corr_laboratorio_2_piano".

I tag vengono inseriti all'interno di un file xml chiamato "tag.xml", che è presente in ogni attuatore. Quando l'attuatore si accende, legge il contenuto di questo file e carica i tag contenuti al suo interno in un array. Quando arriva un comando dal server, l'attuatore esegue il comando solo se il tag inviato dal server è contenuto all'interno dell'array dei tag.

Ad esempio se in un attuatore si vogliono mettere i tag "hot" e "warm-hot" il file tag.xml sarà il seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<tags>
  <tag>hot</tag>
  <tag>warm-hot</tag>
</tags>
```

I sensori vengono registrati mediante la voce di menù "Insert new sensor" presente nel sottomenù "Sensors". Si riporta l'esempio di un inserimento di un sensore:

```
name: sensore_finestra1
server: myserver
public [Y/n]: y
model (optional): SHT15
brand (optional): Sensirion
latitude (optional):
tag (optional):
```

Come si può vedere i campi latitude e tag non sono stati inseriti perchè per questa applicazione non erano utili, ma il loro funzionamento è stato comunque testato a parte ed è risultato funzionante.

I gruppi di sensori vengono inseriti mediante la voce di menù "New Sensors Group", che si trova all'interno del menù "My devices Groups" - "New Devices Group". Ecco un esempio dell'inserimento del gruppo "ufficio1", che è una stanza del gruppo "corr_uffici".

```
name: ufficio1
public [Y/n]: y
description (optional): gruppo che contiene tutti i sensori presenti nella stanza "ufficio1"
```

```
parent group name (optional): corr_uffici
tag (optional):
```

I sensori vengono aggiunti ai gruppi mediante la voce di menù "Add sensor to group" presente nel sottomenù "Sensors". Di seguito si riporta l'esempio effettuato per aggiungere il gruppo "sensore_finestra1" al gruppo "ufficio1".

```
sensor name: sensore_finestra1
group name: ufficio1
```

All'interno file di configurazione xml dell'interruttore chiamato "conf_interrutore.xml", invece, il campo locazione corrisponde al nome del gruppo della stanza in cui esso è inserito. Quando l'interruttore si accende, legge la locazione all'interno di questo file e la carica in una variabile in memoria. Ogni volta che l'interruttore viene attivato o disattivato, insieme al suo stato invia anche il contenuto di questa variabile, in modo tale da indicare la posizione dell'interruttore.

Ad esempio per la stanza "ufficio1", il file di configurazione si presenterà semplicemente così:

```
<?xml version="1.0"?>
<root>
    <locazione>salone1</locazione>
</root>
```

Conclusioni e sviluppi futuri

Il lavoro di tesi presentato in questa tesi ha permesso di sviluppare un sistema che incontra i principi dell'Internet of Things e che permette di accedere ai servizi di un ambiente intelligente in maniera sicura ed affidabile. Tale risultato è stato ottenuto combinando in modo appropriato diverse tecnologie, come HTTP e CoAP, e utilizzando la crittografia e un approccio ibrido di storage basato su sistemi SQL e NoSQL.

Il sistema, inoltre, permette di creare servizi personalizzati e di installarli sui propri server e di accedere ai dispositivi con meccanismi user-friendly come gruppi di dispositivi o tag.

La gestione dei tag utilizzata in questo lavoro non è legata alle caratteristiche dei dispositivi di monitoraggio ed attuazione, ma permette di associare il dispositivo al particolare scopo funzionale. Anche la creazione di gerarchie di gruppo non viene effettuata in base alla posizione dei dispositivi, come nei tradizionali approcci geolocalizzati, ma si basa sull'organizzazione logica di dispositivi impegnati per lo stesso obiettivo funzionale (o attività). Tale gestione di gruppi e dispositivi altamente dinamica permette di creare gerarchie di gruppi che possono cambiare nel tempo e essere sovrapposte. Ciò rende il sistema sviluppato altamente flessibile, dinamico, ed adattabile a diversi scenari applicativi.

Per il futuro potrebbe essere introdotto un versioning dei vari servizi, in modo tale da permettere una gestione più avanzata all'utente oppure si potrebbe creare un meccanismo di condivisione dei servizi, in modo tale da poter visualizzare i servizi pubblici degli altri utenti e installarli nei propri server, Inoltre, potrebbe essere effettuato l'aggiornamento a coapthon 3, visto che lo sviluppo del sistema è iniziato prima del suo rilascio e, quindi, si è usato coapthon 2.

Glossario

Accesso a un dispositivo	Per un sensore si intende leggere le misurazioni effettuate da esso mentre, per un attuatore, significa controllarlo.
server CoAP	Server di frontiera di ogni sottorete
server remoto	Server centrale, a cui fanno riferimento tutti i server CoAP
servizi utente	Servizi che vengono creati dall'utente
servizi admin	Servizi già presenti nel sistema
Disaster recovery	L'insieme delle misure tecnologiche e logistico/ organizzative atte a ripristinare sistemi, dati e infrastrutture necessarie all'erogazione di servizi di business per imprese, associazioni o enti, a fronte di gravi emergenze che ne intacchino la regolare attività [1]

[1] https://it.wikipedia.org/wiki/Disaster_recovery

Bibliografia

- RFC 7228, http://tools.ietf.org/html/rfc7228
- Joe Weinman, "Cloudonomics: A Rigorous Approach to Cloud Benefit Quantification", Ottobre 2011. https://www.csiac.org/sites/default/files/journal_files/stn14_4.pdf
- Joe Weinman, "What's the real reason to do cloud, again?", 3 Ottobre 2014. http://www.cloudcomputing-news.net/news/2014/oct/03/whats-the-real-reason-to-do-cloud-again/
- Joe Weinman, "The Nuances of Cloud Economics", Novembre 2014. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=7057586
- Dharshan, "Understanding durability & write safety in MongoDB", 8 Agosto 2014. http://blog.mongodirector.com/understanding-durability-write-safety-in-mongodb/
- https://it.wikipedia.org/wiki/Cloud_computing
- https://it.wikipedia.org/wiki/MongoDB
- http://coap.technology
- Joe Weinman, "Defining a cloud", 18 Gennaio 2014. http://www.techradar.com/news/internet/cloud-services/defining-a-cloud-1209348
- http://joeweinman.com
- Dan Allen, Seam in Action, Manning Publications, Settembre 2008.
- Peter Mell, Timothy Grance, The NIST Definition of Cloud Computing, NIST, Special Publication 800-145, Settembre 2011.
- Edge 2014: Cloudonomics: The Business Value of the Cloud with Joe Weinman, Cloud Analyst. https://www.youtube.com/watch?v=nxrU3-7Ait4