UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

"Gli standard per l'impiego di sensori nell'agricoltura di precisione: analisi della situazione e ipotesi di architettura"

Relatore: Prof. Massimo Rumor
Correlatore: Ing. Eduard Roccatello

Laureando: Nicolò Levorato

ANNO ACCADEMICO 2021-2022

Data di laurea 21.02.2022

Abstract

Nel settore dell'agricoltura sempre più spesso vengono utilizzati dispositivi digitali per il monitoraggio della fase di produzione. La maggior parte di questi dispositivi sono sensori che raccolgono dati sul campo e li inviano sfruttando la rete internet. L'integrazione della tecnologia *Internet of things* nel settore agricolo ha permesso la raccolta e l'analisi di grandi moli di informazioni che hanno portato ad un maggior controllo sui processi interni con conseguente riduzione dei costi e maggior qualità del prodotto finale. Una delle principali sfide di progettazione di un sistema IoT è affrontare la necessità di interoperabilità tra dispositivi: sensori differenti raccolgono informazioni in formati non omogenei che spesso risultano incompatibili tra loro. L'utente del sistema è costretto ad affidarsi a piattaforme e software diversi per la consultazione dei dati rendendo l'analisi di quest'ultimi complicata e macchinosa.
La soluzione a questo problema è stata individuata nell'utilizzo di uno *standard IoT* che renda omogeneo l'output dei dati. Uno standard IoT fornisce un modello di rappresentazione dei dati attraverso un'interfaccia con la quale i dispositivi IoT possono comunicare tra loro.
In questa tesi si propone un'ipotesi di architettura di sistema che mira a raccogliere le misurazioni prodotte dai sensori e a tradurle in uno standard, precedentemente scelto dopo un'analisi dello stato dell'arte, adeguato ai bisogni del settore dell'agricoltura di precisione. Grazie all'introduzione di un dispositivo *connettore* il sistema può ospitare al suo interno un qualsiasi numero di sensori differenti mantenendo i dati di output in un formato omogeneo. Ad ogni tipologia di sensore è associato un connettore specifico che intercetta i dati destinati al database e li traduce nel formato standard prima di essere inoltrati al server centrale. All'interno della tesi sono presentati alcuni esempi con sensori reali per illustrare nello specifico il funzionamento di questi nuovi dispositivi. L'architettura di sistema proposta punta ad unire flessibilità e semplicità d'uso con un costo economico ridotto. Nei lavori successivi si potrà estendere il bacino di dispositivi integrabili nel sistema includendo anche gli attuatori.

Indice

1 | Introduzione

Con la crescente scarsità di terra arabile dovuta a una serie di fattori umani e climatici in mezzo alla crescente domanda globale di cibo, la necessità di una gestione agricola sostenibile e produttiva sta diventando sempre più pressante. Per affrontare tali problemi, l'IoT è visto come uno strumento potente, grazie al suo potenziale per rendere l'agricoltura maggiormente basata sull'analisi dei dati, portando a sistemi di coltivazione tempestivi, economici ed efficienti.
Le soluzioni per l'agricoltura di precisione, sempre più diffuse, implicano pratiche di gestione basate su misurazioni spaziali che utilizzano il segnale GPS: per esempio, i fertilizzanti potrebbero essere applicati solo dove necessario, oppure la gestione idrica potrebbe essere automizzata per evitare sprechi ed adeguarsi ai bisogni del raccolto, in aggiunta al monitoraggio dei parametri dello stesso.
L'agricoltura intelligente si sta sviluppando al di là di queste applicazioni, migliorando l'uso dei dati spaziali con eventi in tempo reale. Gli agricoltori possono rispondere rapidamente sul campo a qualsiasi cambiamento significativo del tempo, dell'umidità e della qualità dell'aria, nonché della salute di ogni coltura o terreno grazie all'adozione di tali sistemi integranti sensori, veicoli agricoli intelligenti, droni e robot autonomi. In queste applicazioni, l'IoT facilita la documentazione e la supervisione di diverse attività, nonché la tracciabilità dei prodotti con sistemi di analisi, visualizzazione e gestione dei dati.
I dati critici vengono raccolti in tempo reale durante il processo di produzione agricola, i quali vengono trasmessi dalle reti wireless attraverso la piattaforma di supporto M2M (machine to machine). In questo modo si ottengono dati in tempo reale dell'ambiente utilizzando pattern SMS (Short Messaging Service) o web per poter aiutare gli agricoltori a prendere decisioni tempestive per poter migliorare la produzione e rispondere più rapidamente a cambiamenti ed eventi imprevedibili.
In sostanza, l'adozione delle tecnologie IoT ha il potenziale per migliorare l'agricoltura sotto molti aspetti. Alcune delle possibilità che queste tecnologie

offrono sono:

- **Dati raccolti dai sensori, ad es. condizioni meteorologiche, qualità del suolo, andamento della crescita del raccolto o salute del bestiame.** Questi dati possono essere utilizzati per monitorare lo stato dell'attività in generale, nonché le prestazioni del personale, l'efficienza delle apparecchiature, ecc.

- **Miglior controllo sui processi interni e, di conseguenza, minori rischi di produzione.** La capacità di prevedere l'output della produzione consente di pianificare una migliore distribuzione del prodotto.

- **Gestione dei costi e riduzione degli sprechi grazie al maggiore controllo sulla produzione.** Potendo rilevare eventuali anomalie nella crescita del raccolto o nella salute del bestiame si è in grado di mitigare i rischi legati alla perdita del raccolto.

- **Maggiore efficienza aziendale grazie all'automazione dei processi.** Utilizzando dispositivi intelligenti si possono automatizzare più processi durante il ciclo di produzione, ad es. irrigazione, fertilizzazione o controllo dei parassiti.

- **Miglioramento della qualità e dei volumi del prodotto.** Grazie all'utilizzo delle nuove tecnologie si può ottenere un migliore controllo sul processo di produzione e mantenere standard più elevati di qualità del raccolto e capacità di crescita attraverso l'automazione.

Per il raggiungimento di questi obiettivi è necessario che i diversi ambienti di lavoro siano monitorati grazie all'utilizzo di dispositivi IoT. Spesso l'utilizzo di dispositivi IoT si traduce nel posizionamento di sensori in punti strategici dedicati al monitoraggio di specifici parametri, come ad esempio:

- **Temperatura**
  I sensori di temperatura sono fondamentali in due categorie chiave dell'agricoltura intelligente: monitoraggio delle condizioni ambientali e monitoraggio delle risorse meccaniche. È noto che la raccolta del vino di ghiaccio, ad esempio, avviene all'interno della finestra di temperatura ristretta quando le temperature ambiente raggiungono per la prima volta tra -10°C e -12°C durante una stagione di raccolta. Sensori di temperatura e umidità altamente accurati e previsioni di temperatura predittive precise sono fondamentali per l'industria del vino ghiacciato. I sensori di temperatura non solo svolgono un ruolo significativo nel

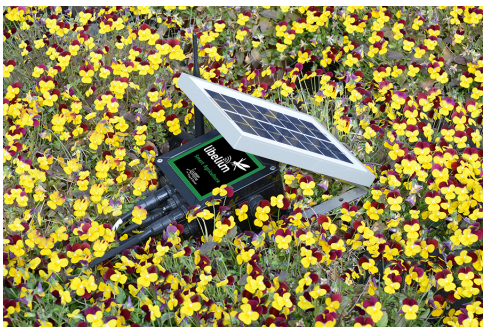
monitoraggio delle condizioni ambientali dello spazio fisico, ma svolgono un ruolo essenziale in quasi tutte le applicazioni di monitoraggio delle risorse dell'agricoltura intelligente.

- **Umidità**
  L'umidità del suolo determina lo stato di approvvigionamento idrico delle colture. L'umidità del suolo troppo alta o troppo bassa influenzerà la normale crescita delle colture sopra il suolo. Solo con un'adeguata umidità del suolo, l'assorbimento dell'acqua delle radici e la traspirazione delle foglie possono raggiungere uno stato equilibrato, favorendo così la crescita delle radici delle colture.

- **Composizione del suolo**
  La disponibilità di nutrienti è essenziale per la crescita delle piante quanto lo è per gli animali e gli organismi viventi. Per ottimizzare il potenziale di crescita di una pianta e ottenere raccolti altamente produttivi, è indispensabile avere una comprensione profonda e quantitativa delle condizioni del suolo da cui provengono i prodotti agricoli. La morfologia del suolo e la presenza di fertilizzanti possono essere analizzate misurando la conducibilità elettrica, il contenuto idrico volumetrico, il potenziale idrico del suolo e i livelli di ossigeno. L'uso di sensori di per il monitoraggio terreno fornisce un feedback critico sulle carenze di nutrienti del suolo o sulla presenza di sostanze chimiche indesiderate. Questi sensori aiutano l'agricoltura intelligente a monitorare le fluttuazioni giornaliere, settimanali, mensili e annuali del pH del suolo e dei livelli di nutrienti per continuare a educare il settore agricolo.

Sensore utilizzato per il controllo dei parametri del suolo

- **Luce**
  L'applicazione del sensore di luce nelle piantagioni agricole in serra può aiutare i coltivatori a comprendere con precisione la legge del tempo di sole, il punto di saturazione della luce e il punto di compensazione della luce della crescita delle piante, e quindi regolare le loro preferenze di luce attraverso la tecnologia di controllo manuale per controllare e migliorare La crescita scientifica di coltivazioni per ottenere rese elevate. Le colture assorbono continuamente CO2 nell'atmosfera per la fotosintesi e utilizzano la fotosintesi per produrre nutrienti per mantenere lo sviluppo e la crescita delle colture. Gli studi hanno scoperto che quando la concentrazione di anidride carbonica atmosferica aumenta, la fotosintesi delle piante sarà notevolmente migliorata.



Sensore che monitora l'intensità della luce solare.

- **Parametri atmosferici**
  Le stazioni meteorologiche agricole sono unità autonome che vengono posizionate in varie posizioni nei campi in crescita. Queste stazioni hanno una combinazione di sensori appropriati per le colture locali e il clima. Informazioni quali temperatura dell'aria, temperatura del suolo a varie profondità, precipitazioni, umidità fogliare, clorofilla, velocità del vento, temperatura del punto di rugiada, direzione del vento, umidità relativa, radiazione solare e pressione atmosferica vengono misurate e registrate a intervalli predeterminati. Questi dati vengono compilati e inviati in modalità wireless a un data logger centrale a intervalli programmati. La loro portabilità e i prezzi decrescenti rendono le stazioni meteorologiche attraenti per gli allevamenti di tutte le dimensioni.

Stazione metereologica utilizzata per il controllo dei parametri atmosferici

Nella raccolta dei dati provenienti da questi sensori però nasce un problema: nel caso in cui il sistema sia composto da dispositivi disomogenei, non esiste al momento uno standard di comunicazione condiviso tra tutti i dispositivi per la raccolta dei dati e la comunicazione. Il mercato attuale offre molteplici dispositivi IoT per l'agricoltura di precisione, alcuni dei quali sono gestibili attraverso software proprietario fornito dall'azienda produttrice altri sono invece programmabili dall'utente. Ogni dispositivo ha il suo metodo con cui raccoglie e invia i dati, perciò il risultato finale risulta essere un insieme di dati disomogenei inviati con specifiche diverse e quindi di difficile consultazione per l'utente finale. Quando un sensore invia un dato, soprattutto nell'ambito IoT, le informazioni trasmesse sono compresse per ridurre al minimo il consumo di risorse. Parte delle informazioni necessarie alla comprensione dei dati (come possono essere, ad esempio, unità di misura, precisione, tipo di misurazione, etc) sono lasciate implicite poiché, utilizzando il sistema standard fornito dalla casa produttrice dei sensori, vengono processate in fase di ricezione ed elaborazione dei dati. Se si volessero utilizzare sensori di produzione diversa, il risultato sarebbe un'insieme di dati incompleti o non decifrabili che sarebbero poco utili al fine dell'analisi.
Per capire meglio il problema, analizziamo un payload di un sensore preso ad esempio, in questo caso *SensorData* di produzione della *Digital Matter*, a cui è collegato un sensore per la misurazione della pressione atmosferica.
Il payload inviato dal sensore è il seguente:

"AF4D3276CE5F3334801260618231006A18"

Senza l'utilizzo di software fornito dal produttore sarebbe impossibile riuscire a decifrare le informazioni contenute nella stringa, che sono posizione

geografica del sensore e il valore di due misurazioni effettuate.
In questa tesi si cerca di affrontare il problema proponendo un'architettura di sistema utilizzante uno standard che sia condivisibile tra i vari dispositivi IoT collegati, in modo da creare una rete che possa ospitare dispositivi di tipologie diverse senza compromettere la comprensione e l'analisi dei dati finali.

Stato attuale

Una delle principali sfide di progettazione del sistema IoT è affrontare la necessità di interoperabilità tra dispositivi. Si stima che il raggiungimento dell'interoperabilità possa portare ad un aumento del 40% del valore di mercato potenziale dell'IoT.
Molti sistemi IoT complessi, ad esempio le città intelligenti, richiedono il coordinamento di diversi sottosistemi di dispositivi IoT che possono e spesso utilizzano formati di dati diversi. L'obiettivo è raggiungere l'interoperabilità semantica, ovvero "rappresentare le informazioni in una forma il cui significato è indipendente dall'applicazione che le genera o le utilizza".
Nei sistemi IoT, la maggior parte dei dati scambiati tra macchine e servizi rappresentano letture da parte di sensori e comandi per gli attuatori: si tratta in genere di numeri specifici in base al tipo di dispositivo e del dominio della misurazione, come ad esempio una lettura di una temperatura, con a volte informazioni aggiuntive come la natura delle misurazioni e le unità di misura, oppure possono essere stringhe di controllo che vanno a modificare parametri o ad azionare gli attuatori.
Le applicazioni all'estremità ricevente devono essere in grado di "comprendere" e interpretare i dati del messaggio per elaborarli e agire di conseguenza in modo appropriato. In altre parole, i due endpoint IoT comunicanti devono raggiungere l'interoperabilità semantica per interagire correttamente.
Ciò implica la formattazione dei dati in modo che vengano interpretati in maniera identica da parte di entrambe le estremità coinvolte in uno scambio di messaggi. Il problema è che non esiste una definizione o uno standard comunemente accettato per la rappresentazione dei dati nei sistemi IoT.
Un modello di informazioni IoT è una rappresentazione astratta e formale di tipi di cose IoT che spesso includono le loro proprietà, relazioni e operazioni che possono essere eseguite su di essi. Più specificamente, un modello di informazioni IoT deve specificare cos'è e cosa può fare un oggetto, le sue proprietà e i modi per interagire con esso. I modelli di informazioni IoT e le rappresentazioni dei dati in genere includono:

- Designazione del tipo di oggetto fisico, cos'è e cosa fa, ad es. Sensore di temperatura
- Formati dei dati, come rappresentarli e interpretarli, ad es. Valori (stringa, numero, int o float)
- Interazioni, metodi di accesso: come accedere all'oggetto per ottenerne lo stato e i dati o attivare funzioni intrinseche e attivare gli output
- Metadati che descrivono gli attributi delle cose e il contesto in cui vengono acquisiti i dati
- Collegamenti ad altri oggetti, utilizzati per rappresentare connessioni strutturali e per la composizione di oggetti

In particolare nel mondo dell'agricoltura di precisione, ad oggi, non esiste uno standard dedicato ed universalmente condiviso. Esistono diversi standard IoT che potrebbero rispettare le richieste di questo campo andando a modellare lo scambio di dati tra i vari sensori, ma, al momento, il mercato offre una serie di dispositivi disomogenei proprietari ognuno con la propria modalità specifica di comunicazione. Il primo lavoro di questa tesi cerca di analizzare gli standard disponibili al momento per poi andare a valutare i pro e i contro di ognuno di loro per poter scegliere quello che soddisfa maggiormente le esigenze richieste nell'ambito dell'agricoltura di precisione.

2 | Protocolli di comunicazione

Per poter scegliere lo standard adeguato è necessario prima comprendere quali sono i protocolli di comunicazione più utilizzati per inviare informazione nel mondo dell'IoT. I protocolli di comunicazione descrivono formalmente quali formati e quali regole deve seguire l'invio di messaggi digitali. Esistono diversi protocolli, alcuni dei quali sono stati progettati appositamente per essere utilizzati in ambiti dove le risorse disponibili sono limitate, come nel caso dei sensori da noi analizzati. È grazie a questi protocolli se è possibile ottenere un flusso di dati coerente e comprensibile dalla maggior parte dei dispositivi collegati in rete. Di seguito un elenco dei protocolli più diffusi e maggiormente utilizzati nel settore dell'agricoltura di precisione.

2.1 Protocolli IP

HTTP

HTTP è un protocollo che lavora con un'architettura di tipo client/server: il client esegue una richiesta e il server restituisce la risposta mandata da un altro host. Nell'uso comune il client corrisponde al browser ed il server la macchina su cui risiede il sito web. Vi sono quindi due tipi di messaggi HTTP: messaggi richiesta e messaggi risposta. I messaggi di richiesta utilizzano determinati comandi come `GET,PUT,POST` per poter interagire con il server mentre i messaggi di risposta forniscono dati e/o informazioni riguardo allo status delle interazioni. Una volta che una particolare richiesta (o una serie di richieste correlate) è stata soddisfatta, la connessione viene chiusa e per poter effettuare altre azioni è necessario ripristinarla, in questo modo si mantengono solo le connessione attive e si va a diminuire il carico nel server. Il protocollo HTTP è molto utizzato nelle comunicazioni attraverso internet, ma risulta inefficiente nelle comunicazioni IoT poiché necessita di un header molto "pesante" per poter funzionare, andando a caricare eccessivamente le risorse limitate di un sistema IoT.

CoAP

Constrained Application Protocol (CoAP) è stato sviluppato per soddisfare le esigenze di dispositivi e reti vincolati. Utilizza intestazioni più piccole e semplificate e supporta REST asincrono e gli scambi di messaggi di pubblicazione / sottoscrizione nella comunicazione diretta da macchina a macchina (M2M). Supporta anche il rilevamento integrato di servizi e risorse e supporto per URI e tipi di media web. CoAP è intenzionalmente progettato per essere una controparte leggera di HTTP e per rimanere il più vicino possibile al suo design al fine di semplificare l'integrazione con il Web, soddisfacendo nel contempo requisiti IoT aggiuntivi come supporto multicast, basso overhead e semplicità.
La comunicazione CoAP è progettata per soddisfare le esigenze di scambi IoT di messaggi che consistono principalmente in letture e comandi dei dati dei sensori. CoAP è composto di due livelli: uno superiore per richieste e risposte e uno inferiore di messaggistica, entrambi risiedono al di sopra del trasporto UDP. Utilizza il protocollo UDP a livello di trasporto tra gli endpoint per semplicità e minore latenza, a scapito dell'affidabilità. Nel caso si avessero bisogno di un sistema di comunicazione affidabile, CoAP include un'opzione per la consegna affidabile di una classe dei suoi messaggi.

MQTT

MQTT (Message Queuing and Telemetry System) è un sistema di pubblicazione-sottoscrizione specificamente progettato per i sistemi IoT. È ampiamente utilizzato nelle piattaforme IoT per fornire dati dei sensori ad applicazioni e servizi. È il meccanismo di importazione dei dati predefinito utilizzato dalle piattaforme commerciali cloud. MQTT è progettato per l'uso in reti che possono avere una larghezza di banda ridotta, un'elevata latenza e connessioni fragili tra dispositivi vincolati con poca memoria e bassa potenza.
MQTT presume che le connessioni di rete tra i suoi componenti forniscano i mezzi per inviare flussi di byte ordinati e senza perdite in entrambe le direzioni. In pratica, molte implementazioni di MQTT utilizzano TCP per soddisfare questo requisito. Il protocollo è leggero, nel senso che richiede solo un'intestazione di 2 byte per i messaggi, con alcune estensioni opzionali.

Confronto

Nella tabella sottostante sono stati messi riassunti e messi a confronto i protocolli di comunicazione descritti nella sezione precedente:

	HTTP	**MQTT**	**CoAP**
Generale	HTTP utilizza TCP come protocollo di trasporto sottostante. Il modello di interazione è richiesta/risposta con pipelining e keep-alive. La trasmissione dei dati è affidabile grazie alle proprietà del TCP. HTTP è un protocollo punto a punto (nessun supporto multicast). HTTP utilizza metodi (come `GET, POST, PUT, DELETE`) per indicare l'azione desiderata sulla risorsa identificata. I codici di stato (`200, OK, ecc.`) Indicano la condizione di successo o fallimento della richiesta.	MQTT fornisce un trasporto di messaggistica indipendente dal file contenuto del carico utile. MQTT fornisce la comunicazione bidirezionale tra i client tramite il server centrale. Utilizza un modello di messaggio di pubblicazione / sottoscrizione che fornisce la distribuzione dei messaggi uno-a-molti e il disaccoppiamento delle applicazioni.	CoAP supporta la consegna affidabile e inaffidabile dei dati. La consegna affidabile implica che il mittente contrassegna una richiesta come "confermabile" e si aspetta una risposta dal destinatario. Se quella risposta non arriva il client ritrasmette la richiesta. CoAP fornisce due livelli di comunicazione: un livello di affidabilità e un livello REST, che imita HTTP con metodi come `POST, PUT, GET, DELETE. PATCH` e `FETCH` sono stati aggiunti a CoAP in RFC 8132.
Trasporto	TCP	TCP	UDP
Supporto all'architettura REST	Si	No	Si
Supporto sub/pub	No	Si	No
Struttura payload	Il sovraccarico dell'intestazione è relativamente elevato poiché l'intestazione utilizza testo in chiaro leggibile dall'uomo anziché la codifica binaria. Inoltre, i parametri URI devono essere codificati in Base64. Con il pipelining e il keep-alive, il socket TCP può rimanere aperto, riducendo il numero di messaggi TCP per l'impostazione e l'interruzione della connessione TCP.	MQTT ha un piccolo overhead del protocollo (l'intestazione a lunghezza fissa è di soli 2 byte).	CoAP ha una struttura di intestazione a lunghezza variabile, che è lunga almeno 4 byte. CoAP utilizza una codifica binaria per l'intestazione e le opzioni contenute nell'intestazione. La codifica dei dati nel corpo del messaggio dipende dai dati dell'applicazione scambiati.

Tabella 2.1 Tabella di confronto dei protocolli IP

2.2 Protocolli non IP

Una connessione ad internet che richieda il minor dispendio di risorse è un problema essenziale per i dispositivi embedded a bassa potenza utilizzati all'interno dei sensori. Le tecnologie di comunicazione wireless convenzionali sono insufficienti se si considerano la copertura, il consumo energetico e il costo. Le LPWAN mirano a risolvere questi problemi in modo che possano essere scalabili e adattati a implementazioni su larga scala per i dispositivi a bassa potenza. Si suppone che le reti "Low Power Area Network"(LPWAN) operino a basse velocità di trasmissione dati per riuscire a coprire distanze di diversi kilometri sia in zone urbane densamente popolate sia nelle zone suburbane. LoRaWAN e SigFox come esempio di tecnologie di comunicazione sub-GHz forniscono con successo queste funzionalità.

Le LPWAN hanno alcune caratteristiche comuni che contraddistinguono queste tecnologie dalle reti di comunicazione tradizionali:

- Bassa potenza, la rete dei dispositivi finali dovrebbero consumare poca energia.
- I costi di comunicazione ridotti
- Possibilità di integrare la geolocalizzazione
- Modulazione robusta per evitare che in aree urbane ad alta densità porta le reti radio si inceppino.

La tabella 2.2 mette a confronto WiFi e Bluetooth con le tecnologie LPWAN più diffuse nel mercato moderno. Nell'ambito dell'agricoltura di precisione le tecnologie più diffuse sono LoRaWAN e SigFox, due protocolli proprietari ideati per dispositivi IoT che riescono a conciliare copertura di distanze dell'ordine di kilometri e basso consumo di energia.

	SigFox	NB-IoT	LoRaWAN	WiFi	Bluetooth
Standard	SigFox	3GPP	LoRa Alliance	IEEE 802.11	Bluetooth SIG
Modulazione	BPSK	QSPK	CSS	DSSS, OFDM	GFSK
Frequenze	ISM:433 MHz, 868 MHz, 915 MHz	Licenza sotto LTE	ISM:433 MHz, 868 MHz, 915 MHz	ISM:2.4 GHz, 5 GHz	2.4GHz
Copertura	10-40 km	2-20 km	1-10 km	10-100 m	10-100 m
Banda	100 Hz	200 kHz	125 kHz, 250 kHz	20 MHz, 40 MHz, 80 MHz, 160 MHz	1 MHz
Limite TX	140 pacchetti/giorno	Illimitato	Duty Cycle	Illimitato	Illimitato
Max Data rate	100 bps	200 kbps	50 kbps	Gbps	2 Mbps
Privato	No	no	si	si	si
Consumo energetico	Basso	Basso	Basso	Alto	Basso
Sicurezza	Bassa	Alta	Alta	Bassa-Alta	Bassa-Alta

Tabella 2.2 Tabella confronto tra LPWAN, WiFi e Bluetooth

SigFox

SigFox è un protocollo comunicativo proprietario sviluppato dall'omonima azienda francese fondata nel 2010, la quale si dedica allo sviluppo di reti wireless per dispositivi IoT. L'utilizzo della rete SigFox è sottoposto a licenze a pagamento che vengono gestite a livello nazionale dalle società delegate. In Italia la società che si occupa delle licenze SigFox è NetTrotter.
La tecnologia SigFox fa parte della famiglia di tecnologie LPWAN, impiegate principalmente per lo sviluppo delle reti IoT, quando il volume dei dati inviati (dati che spesso vengono prelevati dai sensori) è basso (si va da pochi byte a centinaia di kilobyte), il raggio d'azione è ottimo (raggiungendo decine di km) e l'assorbimento di corrente è molto basso (nell'ordine di mA o decine di mA per trasmissione). SigFox utilizza la modulazione D-BPSK (Differential Binary Phase-Shift Keying) per la quale il messaggio ha una larghezza di banda fissa di 100Hz e viene inviato con una velocità di 100bps (per l'Europa) o 600bps (per gli USA), entro uno spettro di frequenze senza licenza

inferiore a 1 GHz, 868 MHz per la regione Europa e 915 MHz per la regione USA. Questa tecnica di modulazione fa parte del tipo di modulazione UNB (Ultra-Narrow Band), che, insieme a Chirp Spreading Spectrum (utilizzato dai sistemi LoRaWAN) e Narrow-Band (utilizzato da NB-IoT), richiede un basso consumo energetico per garantire connessioni tra nodi e BS. I vantaggi dell'utilizzo della modulazione D-BPSK sono che porta un'elevata efficienza nell'accesso al mezzo dello spettro ed è facile da implementare. Un bit rate basso consente l'uso di componenti a basso costo da parte del ricetrasmettitore. La stazione base o il ricevitore gateway è altamente sensibile in quanto può demodulare segnali che sono molto vicini al limite del livello di rumore senza alcuno strato di codifica. La frequenza centrale di 868 MHz che fa parte della banda di frequenza SRD860 viene utilizzata in Europa, mentre la banda di frequenza ISM900 con una frequenza centrale di 915 MHz viene utilizzata negli Stati Uniti.

LoRa & LoRaWAN

LoRa (long range) è una WAN a bassa potenza di proprietà di "LoRA Alliance". Opera nello spettro senza licenza su più bande a partire da 169 a 430 MHz, di cui 868 MHz (Europa) e 915 MHz (Nord America) sono le più comuni. Il suo raggio d'azione è dell'ordine di 10 km in aree aperte e più vicino a 1 km in spazi densamente popolati come le città. La larghezza di banda arriva fino a 50 Kbps, a seconda della distanza e della potenza di trasmissione. La dimensione del payload è compresa tra circa 50 byte e 250 byte, a seconda della banda di frequenza. La rete è effettivamente half-duplex, ovvero comunicazione bidirezionale ma in una sola direzione alla volta. Di conseguenza, mittenti e destinatari devono coordinarsi nell'effettuare svolte direzionali. La specifica LoRa è costituita da livelli PHY e MAC che possono supportare applicazioni con dati grezzi. Definisce tre tipi di nodi:

- Dispositivi finali bidirezionali (Classe A) - Ogni trasmissione di uplink del dispositivo è seguita dalle due brevi finestre di ricezione del downlink. Nodi a bassa potenza, non supporta comunicazioni avviate da downlink.

- Dispositivi finali bidirezionali con slot di ricezione programmati (Classe B) - Oltre alle finestre di ricezione casuale di Classe A, i dispositivi di classe B aprono finestre di ricezione extra in orari programmati.

- Dispositivi finali bidirezionali con slot di ricezione massimi (Classe C): finestre di ricezione aperte quasi continuamente, chiuse solo durante la

trasmissione. Solitamente sono dispositivi alimentati da una rete, hano latenza più bassa per la comunicazione da server a dispositivo.

LoRa utilizza la modulazione dello spettro diffuso e supporta un algoritmo ADR (Adaptive Data Rate) che consente alle velocità dei dati sui singoli collegamenti di variare in base alla potenza del segnale e alla potenza radio. La velocità dei dati può variare tra 0,3 e 27 Kbps.

2.3 Comunicazione tra protocollo non IP e protocollo IP

Ad oggi, le soluzioni LPWAN end-to-end sono poco diffuse o non sempre implementabili e quindi spesso bisogna lavorare con più fornitori per acquisire nodi, gateway, server di backend e ogni altra parte dell'ecosistema separatamente. Sebbene ciò consenta molta flessibilità nelle applicazioni, lascia agli sviluppatori l'onere di implementare la rete gestendo la pacchettizzazione, il controllo del downlink, il multicast, ecc. Le Low-Power Wide-Area Networks (LPWAN), come LoRaWAN e SigFox , sono un insieme di tecnologie di comunicazione a lungo raggio dedicate all'implementazione dell'IoT. Una rete WAN a bassa potenza crea una connessione punto-punto tra un dispositivo e un server. Molte applicazioni e implementazioni LPWAN sono strettamente collegate alla tecnologia LPWAN sottostante. Pertanto, raggiungere nuovi mercati e integrare tecnologie aggiuntive significa reingegnerizzare laboriosamente sia il cloud che le applicazioni dei dispositivi. Queste tecnologie di rete emergenti non seguono il modello Internet basato sul protocollo IP. Non sono interoperabili tra loro e sono difficili da integrare con architetture e sistemi informativi esistenti. Ogni progetto IoT prevede prima di scegliere una tecnologia, poi di adeguare la scelta di dispositivi e piattaforme e di gestire l'integrazione con l'architettura esistente. Così il mercato rimane in qualche modo organizzato in silos tecnologici. Nel mondo di Internet, l'interoperabilità e la multiconnettività sono proprietà native. In effetti, il protocollo Internet è aperto, consentendo a ogni utente di operare attorno a uno standard di riferimento universale. Internet utilizza un modello a strati. Ogni livello implementa un protocollo e astrae la complessità degli altri livelli. Questo modello a clessidra isola l'applicazione dalla rete, risparmiando notevoli sforzi di progettazione per ogni attore della catena del valore. Nel suo nucleo, l'IP fornisce servizi ottimizzati come l'indirizzamento e il routing, servizi che qualsiasi applicazione può ereditare senza dover reimplementare tutto. Per praticità, in questo paragrafo prenderemo ad esempio la tecno-

logia LoRaWAN per mostrare come un protocollo non IP comunica con un protocollo IP.

Application	Dal processo di rete all'applicazione
Presentation	Rappresentazione dei dati e criptazione
Session	Comunicazione inter-host
Transport	Connessione end-to-end ed affidabilità
Network	Determinazione dei percorsi ed indirizzo logico
Data link	Indirizzamento fisico (MAC e LLC)
Physical	Mezzo,segnale,trasmissione binaria

Figura 2.1 Modello ISO/OSI

A causa delle severe restrizioni sulla larghezza di banda, LoRaWAN non è in grado di trasportare il pesante overhead del protocollo IP, perciò implementa a suo modo il 3° livello del modello di rete OSI che normalmente è occupato dal protocollo IP. In particolare LoRaWAN si occupa di implementare il 2° e il 3° livello del modello di rete OSI (il collegamento dati e il livello di rete). L'implementazione da parte di LoRaWAN del 3° livello fa si che una rete LoRaWAN non possa comunicare con una rete IP se non grazie ad un gateway che faccia da "ponte" tra le due reti. I gateway LoRaWAN sono moduli radio i cui dispositivi sono dotati di un concentratore LoRa che consente la ricezione dei pacchetti LoRa. Un gateway LoRaWAN viene utilizzato principalmente per trasmettere i dati dei dispositivi IoT al cloud. La stessa tecnologia LoRaWAN viene utilizzata solo per la comunicazione tra i rispettivi dispositivi finali e i gateway ad essi collegati. I gateway LoRaWAN includono un sistema operativo che si occupa di inoltrare i pacchetti LoRa alla rete di destinazione. La velocità dei dati tra il dispositivo finale e il gateway LoRaWAN è relativamente bassa, ma questo è un sacrificio necessario per consentire una lunga durata della batteria e un'elevata portata radio. Il gateway stesso infine è connesso ad una rete IP a larghezza di banda elevata come WiFi, Ethernet o cellulare per fornire connettività end-to-end tra i nodi finali LoRaWAN e il server delle applicazioni. L'immagine sottostante sintetizza come avvengono

le comunicazioni tra una rete LoRaWAN ed Internet, partendo dai nodi IoT fino ad arrivare al server dell'applicazione.

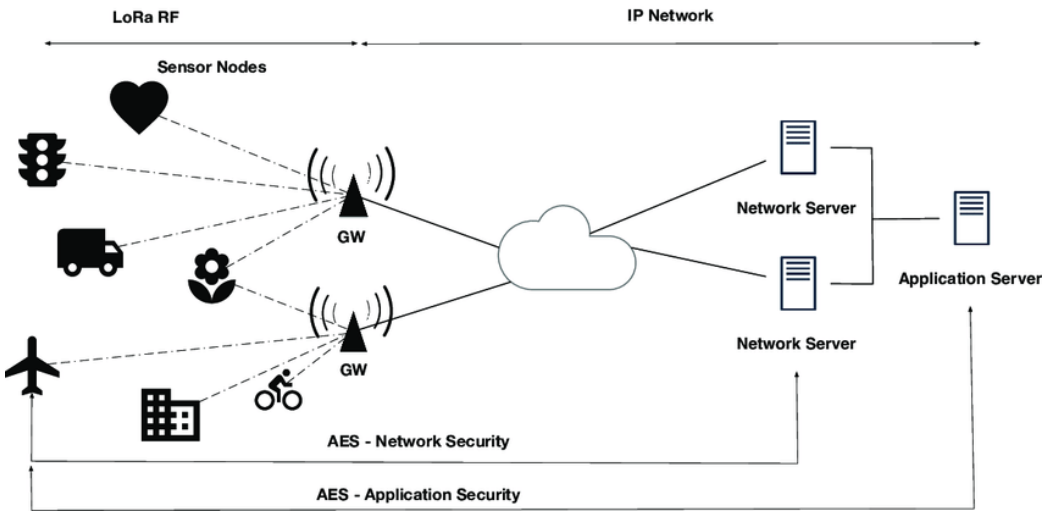


Figura 2.2
Dispositivo o nodo finale: un oggetto con un dispositivo di comunicazione a bassa potenza incorporato.
Gateway: antenne che ricevono le trasmissioni dai dispositivi finali e inviano i dati al cloud.
Server di rete: server che instradano i messaggi dai dispositivi finali all'applicazione corretta e viceversa.
Applicazione: un software in esecuzione su un server.

Riassumendo, tutti i gateway LoRaWAN nel raggio di un nodo finale ricevono dati dal dispositivo. Quindi, i gateway inoltrano questi dati al server di rete che è responsabile della rimozione dei pacchetti di dati duplicati, della verifica dell'integrità dei dati e dell'esecuzione dei controlli di sicurezza prima di inviare il messaggio al server delle applicazioni.

3 | Analisi e scelta dello standard

Uno standard IoT è un livello middleware al di sotto di una o più applicazioni IoT che presenta un'interfaccia dell'applicazione collegata alla rete attraverso la quale interagiscono i nodi collegati. Gli standard spesso supportano più tecnologie di comunicazione e tecniche di trasmissione dei messaggi. Gli standard IoT hanno quattro obiettivi di progettazione principali:

- ridurre i tempi di sviluppo e portare prima sul mercato le soluzioni IoT
- ridurre l'apparente complessità dell'implementazione e del funzionamento di una rete IoT
- migliorare la portabilità e l'interoperabilità delle applicazioni
- migliorare la funzionalità, l'affidabilità e la manutenibilità

Data la vasta gamma di metodi di comunicazione dei dispositivi IoT, è insostenibile che le applicazioni possano gestire ogni metodo esistente. Gli standard mirano a nascondere questa complessità proponendo un modello comune per il passaggio di dati ai livelli comunicativi superiori.
Le organizzazioni di standardizzazione, le quali non sono enti di normazione formale ma sono più generalmente denominati Standards Definition Organizations (SDO), aiutano a raggiungere gli obiettivi precedentemente elencati attraverso la standardizzazione dell'interconnessione a livello di framework, la definizione dell'interfaccia di passaggio dei messaggi e le definizioni dei dati sfruttate dalle applicazioni.
I framework semplificano le reti IoT creando un'astrazione delle reti di dispositivi IoT nascondendo gran parte della complessità sottostante esponendo le interfacce e le funzioni dello standard facilitando l'interoperabilità tra dispositivi. Il framework fornisce una descrizione semanticamente dei nodi, oggetti e interazioni tra dispositivi IoT che consentono ai progettisti di reti IoT di concentrarsi unicamente sulla semantica dell'interazione dei nodi piuttosto che sui dettagli della connettività.
Tutti gli standard illustrati modellano oggetti fisici, come sensori e attuatori,

in rappresentazioni software che i server, che ospitano le istanze delle varie risorse, possono utilizzare per visualizzare i loro stati. Uno standard IoT si pone come layer tra i servizi offerti dall'applicazione e la rete su cui essa opera. Nella figura 3.1 si vede come lo standard IoT si pone come ultimo livello nel modello TCP/IP delle comunicazioni network:

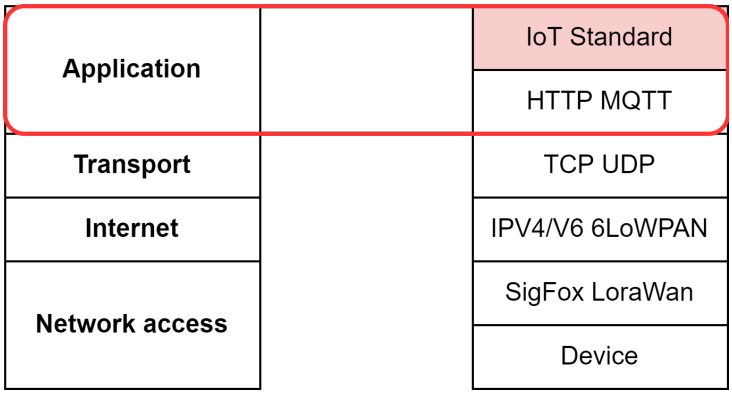


Figura 3.1

Grazie ai servizi forniti dal server, i client IoT utilizzano lo standard per creare le proprie istanze di rappresentazione e per interpretare le risposte dati da quest'ultimi. L'uso della specifica comune si traduce in interoperabilità sintattica e semantica nell'interpretazione del significato dei dati.

3.1 Requisiti richiesti dallo standard

I dispositivi IoT utilizzati all'interno dei sistemi di monitoraggio utilizzati nell'agricoltura di precisione spesso hanno specifiche tecniche differenti, dettate dai produttori, con le quali è possibile interagire ricevendo e inviando messaggi a prescindere dal protocollo fisico di comunicazione utilizzato. Tuttavia queste specifiche comportano l'introduzione di problematiche all'interno del sistema: oltre a dover conoscere le modalità e la sintassi delle comunicazioni di ogni singola tipologia di dispositivo, è necessario tener conto anche di tutte le variazioni alle specifiche tecniche introdotte dal produttore per la risoluzione di problemi specifici o il supporto a nuove funzionalità.
Per ottenere una comunicazione agile e trasparente alla tipologia di dispositivi utilizzati, il sistema proposto introduce uno strato logico di intercompatibilità a livello di memorizzazione del dato che consenta di fruire dei dati originati

dai sensori.
Sono stati individuati i requisiti fondamentali che uno standard per sensori IoT deve possedere per poter essere adatto ad implementare lo strato logico introdotto nel progetto.

- Flusso di dati leggero
- Semplicità e chiarezza d'utilizzo
- Agilmente modellabile in base alle circostanze d'impiego
- Aperto
- Supporto alla geolocalizzazione
- Supporto a diverse tipologie di dati
- Esistenza di implementazioni open source
- Adeguato ai bisogni del settore dell'agricoltura di precisione

Lo stato dell'arte degli standard propone numerose soluzioni che forniscono un framework completo per la comunicazione tra dispositivi IoT rispettando i requisiti individuati. Si è scelto di fare una selezione di esse valutando la diffusione nel mercato e la provenienza.

3.2 Standard selezionati

IPSO (IP for Smart Objects)

IPSO (IP for smart objects) è uno standard nato nel 2008 dalla ISPO Alliance con lo scopo di creare un modello di progettazione comune e un modello di dati in grado di fornire un'interoperabilità di alto livello tra dispositivi e applicazioni software per l'Internet delle cose. Nel 2018 IPSO Alliance si è fusa con OMA SpecWorks che tutt'oggi continua a sviluppare lo standard.
La specifica IPSO si concentra sugli oggetti e lavora a livello di framework e applicazione. L'unico binding IPSO pubblicato è la specifica OMA (Object Management Association) Lightweight M2M (LWM2M) che include anche la gestione di protocolli di livello inferiore, come, ad esempio, CoAP.
Le specifiche IPSO forniscono un modello a oggetti che rende possibile l'interoperabilità ad alto livello negli scambi tra dispositivi IoT e applicazioni software connesse ad altri dispositivi e servizi.

Nello standard IPSO, la struttura che rappresenta l'oggetto include una definizione del tipo di oggetto e una raccolta di risorse nominate formalmente dallo standard, ciascuna delle quali rappresenta un valore semplice che descrive una funzione dell'oggetto.
Gli oggetti e le risorse vengono mappati attraverso percorsi URI,ognuno dei quali identificativo di un oggetto unico, con la sequenza: ID del tipo di oggetto, ID dell'istanza dell'oggetto e ID del tipo di risorsa. La struttura è composta da tre numeri interi a 16 bit senza segno separati dal carattere "/" nella seguente forma:

```
Object ID/Instance ID/Resource ID
```

Gli oggetti sono contenitori tipizzati, che definiscono il tipo semantico delle istanze. Le istanze degli oggetti sono contenitori per le risorse, che sono le proprietà osservabili di un oggetto.
Un esempio di URI per un sensore di temperatura in IPSO è 3303/0/5700. "3303" è un ID oggetto IPSO per il sensore di temperatura, "0" indica l'ID istanza e "5700" è l'ID risorsa per il valore del sensore, chiamato proprietà nella terminologia IPSO.
Semanticamente, il tipo di oggetto rappresenta un singolo punto di misurazione, attuazione o controllo. La risorsa specifica un'osservazione particolare o una proprietà attiva di un oggetto. Ad esempio, un sensore di temperatura potrebbe esporre il valore corrente, la lettura minima e massima possibili, la lettura minima e massima in un intervallo e i metadati.
L'interoperabilità semantica si ottiene attraverso un insieme predefinito di identificatori di oggetti e risorse.
Pertanto, IPSO rappresenterebbe la lettura della temperatura dell'istanza 0 del sensore di temperatura (altri sensori di temperatura, se presenti, sarebbero le istanze 1, 2, ...) di 42,34 come `3303/0/5700 42.3`, e i suoi valori minimi e massimi consentiti sarebbero designati, rispettivamente, come `3303/0/5601 34 3303/0/5602 45`.
IPSO definisce tutte le proprietà utilizzabili che possono essere utilizzate per andare a modellare gli oggetti. Queste proprietà sono chiamate riutilizzabili perché sono abbastanza comuni nei dispositivi IoT e possono essere applicate a più tipi di oggetti. Queste proprietà sono definite nel registro delle risorse riutilizzabili gestito da OMNA. Ce ne sono oltre 50 che coprono una varietà di proprietà, come lo stato e la polarità di ingressi digitali e analogici, contatori, valori misurati minimi e massimi e intervallo, valori e intervalli di potenza attiva e reattiva, cicli di lavoro e valori di setpoint.
IPSO ha definito oltre 50 tipi di oggetti intelligenti. Includono ingressi e uscite digitali e analogici, sensori I/O generici, presenza, temperatura, umi-

dità, alimentazione, controllo del carico, controllo della luce, barometro, accelerometro, attuazione e set point.

Haystack

Project Haystack è un'associazione di aziende del settore nata nel 2014 con lo scopo di promuovere la collaborazione tra le società di software e tecnologia. I loro sforzi si sono concentrati sullo sviluppo di soluzioni per la modellazione semantica dei dati relativi ai dispositivi intelligenti, tra cui: sistemi di apparecchiature per l'edilizia, dispositivi di automazione e controllo, sensori e dispositivi di rilevamento e sistemi di automazione degli edifici.
Haystack fornisce una metodologia comune per la definizione dei metadati attraverso una serie di tag, espressi come coppie nome-valore, che vengono utilizzati per esprimere attributi e proprietà associati alle entità del sistema. Haystack definisce un vocabolario comune sotto forma di librerie di tag definiti dalla comunità. I tag hanno nomi fissi e grazie alle librerie è possibile definire quali attributi o proprietà descrivono. Haystack non definisce quali e quanti tag devono essere utilizzati per descrivere un determinato dato, ciò che definisce è come devono essere denominati quando vengono utilizzati. Un endpoint, chiamato punto in Haystack, può essere un sensore, un comando a un attuatore (cmd) o un set point (sp) che può essere scritto. I tipi di base in Haystack includono punti, attrezzature e sito. Ad esempio, un punto che è un sensore di temperatura può avere alcuni o tutti i seguenti tag, così come altri, associati ad esso:
`temp, sensor, unit, air, discharge, equipRef, siteteRef`
Tutti questi sono nomi di tag Haystack che, se utilizzati, devono essere denominati come specificato nella libreria di tag appropriata.
Haystack nasce come supporto all'automazione degli edifici, all'illuminazione e alla gestione dell'energia, tuttavia, l'approccio architettonico Haystack alla gestione dei metadati è applicabile ad altri domini e all'IoT in generale.
La filosofia di Haystack è che i progettisti/installatori di sistemi abbiano la libertà di annotare quanto ritengano utile mentre le applicazioni utilizzano ed interpretano i tag che ritengono significativi per il loro scopo per capire il contesto. La serializzazione dei dati dei payload Haystack è supportata in JSON, CSV (valori separati da virgole), Zinc (simile a CSV con alcuni tipi di dati) e griglie (strutture dati tabulari bidimensionali).

OCF (Open Computing Foundation)

La Open Connectivity Foundation (OCF) è un'organizzazione del settore nata nel 2016 che sviluppa standard, promuove una serie di linee guida sul-

l'interoperabilità e fornisce un programma di certificazione per i dispositivi coinvolti nell'Internet delle cose.
OCF specifica un framework IoT completo che include un modello di informazione, identificazione e indirizzamento, rilevamento, messaggistica, gestione dei dispositivi e sicurezza. Gli endpoint di ciascuna interazione sono indicati come client e server, con il server che fornisce un'istanza della rappresentazione dell'oggetto d'interesse e il client che invia richieste per recuperare lo stato di un server o per avviare l'attivazione.
Nella terminologia OCF, le entità sono oggetti fisici modellati come risorse. OCF fornisce un modello di risorsa e un modello di interazione. Le interazioni vengono definite utilizzando il modello CRUDN (crea, leggi, aggiorna, elimina, notifica) mappato attraverso il protocollo CoAP. Per ogni tipo di risorsa, una specifica OCF descrive come si comporta in risposta a ogni possibile interazione.
Ogni risorsa può avere più proprietà. Inoltre, ogni risorsa ha due proprietà obbligatorie che sono attributi di collegamento "if" (tipi di interfaccia) e "rt" (tipi di risorsa) le cui varianti sono descritte nella specifica. Altre proprietà rappresentano dati e metadati e sono espresse come coppie nome-valore. Ad esempio, la risorsa sensore ha proprietà "valore", "intervallo" e "unità" mentre la proprietà denominata "link" può essere utilizzata per collegare una risorsa ad altre e creare raccolte e così via.
La proprietà "rt" (resource-type) descrive il tipo di dispositivo fisico. In OCF le risorse sono descritte utilizzando le specifiche API e lo schema JSON: le API definiscono metodi consentiti e tipi di interfaccia supportati dalle risorse specifiche; lo schema JSON definisce i payload, le proprietà e i tipi di dati.
La proprietà "if" (interface type) definisce i metodi supportati da un tipo di risorsa e i payload che utilizza. Un'istanza di una risorsa può esporre più di un tipo di interfaccia che a sua volta può avere metodi e payload differenti. Una recente specifica dei tipi di risorsa OCF include oltre 120 tipi di risorse che vanno a modellare entità fisiche, come sensori, dispositivi per l'automazione domestica, dispositivi di sicurezza, intrattenimento e salute, misurazione e controllo dell'energia.

OGC SensorThings

L'Open Geospatial Consortium (OGC) è un'organizzazione internazionale nata nel 1994 che si dedica allo sviluppo e alla divulgazione di standard comunicativi senza scopi di lucro. All'interno di OGC collaborano più di 500 organizzazioni commerciali, governative e di ricerca per lo sviluppo di standard aperti per la gestione di contenuti geospaziali e servizi IoT. Nel 2015 ha pubblicato la prima parte dello standard dedicato ai sensori con il nome

"SensorThings".
L'API OGC SensorThings fornisce un modo aperto, geospaziale e unificato per interconnettere i dispositivi, i dati e le applicazioni dell'Internet of Things sul Web.
L'API SensorThings è progettata specificamente per i dispositivi IoT con risorse limitate. Segue i principi REST, la codifica JSON e il protocollo OASIS OData e le convenzioni URL. Inoltre, ha un'estensione MQTT che consente agli utenti/dispositivi di pubblicare e sottoscrivere aggiornamenti dai dispositivi e può utilizzare CoAP oltre a HTTP.
Il fondamento dell'API SensorThings è il suo modello di dati basato sulla ISO 19156 (Osservazioni e misurazioni ISO / OGC), che definisce un modello concettuale per le osservazioni e per le caratteristiche coinvolte nel campionamento quando si effettuano osservazioni. Nel contesto di SensorThings, i dispositivi sono modellati attraverso entità che ne esprimono le proprietà. L'API SensorThings fornisce un modello di gestione delle osservazioni interoperabile, particolarmente utile per riconciliare le differenze tra sistemi di rilevamento eterogenei (ad esempio sensori in sito e sensori remoti).
Un dispositivo IoT è modellato come un oggetto. Una oggetto ha un numero arbitrario di posizioni e un numero arbitrario di `DATASTREAM`. Ogni `DATASTREAM` osserva una proprietà monitorata attraverso un sensore e raggruppa le osservazioni raccolte dal sensore. Ogni osservazione monitora una particolare `FEATURE OF INTEREST`, cioè una proprietà di un oggetto reale. Il modello basato su O&M consente a SensorThings di ospitare dispositivi IoT eterogenei e i dati raccolti dai dispositivi.
L'API SensorThings fornisce due funzionalità principali, ciascuna gestita da una delle due sezioni di cui è formato lo standard. Le due sezioni sono la parte Sensing e la parte Tasking. La parte di rilevamento fornisce un modo standard per gestire e recuperare osservazioni e metadati da sistemi di sensori IoT eterogenei e le funzioni della parte di rilevamento sono simili al servizio di osservazione dei sensori OGC. La parte Tasking fornisce un modo standard per la parametrizzazione, chiamata anche tasking, di dispositivi IoT in grado di operare, come sensori o attuatori. Le funzioni della parte Tasking sono simili al servizio di pianificazione del sensore OGC.

3.3 Analisi degli standard

È stata effettuata un'analisi sui vantaggi e sugli svantaggi che ognuno degli standard descritti porterebbe al sistema in caso di sua adozione. Per effettuare l'analisi sono stati presi in considerazione sia i requisiti richiesti dal sistema individuati in precedenza, sia le caratteristiche uniche di ogni stan-

dard.
Nella tabella 3.1 sono state riassunte le caratteristiche chiave di ogni standard.

	Ipso	**Haystack**	**OCF**	**SensorThings**
Modello di base	Lwm2m	-	-	Odata OASIS
Modellazioni Oggetti	Attraverso l'utilizzo di modelli predefiniti	Attraverso l'utilizzo dei tag definiti dall'utente	Attraverso l'utilizzo di tipologie di risorse e interfacce predefinite dallo standard	Attraverso l'utilizzo delle entità definite dallo standard
Proprietà degli Oggetti	Definite dal modello ed identificate da un ID	Definite dai tag utilizzati	Definite dalla tipologia di risorsa con aggiunte personalizzabili	Definite dal tipo di entità
Geolocalizzazione	Espressa come risorsa oggetto	Espressa come proprietà attraverso i tag	Definita dal tipo di risorsa	Nativa attraverso entità

Tabella 3.1 Tabella confronto degli Standard

Per ogni standard sono stati individuati punti deboli e punti di forza.

- **Ispo**

Pro	Contro
– Semplice da utilizzare per oggetti semplici – Ampia lista di oggetti già modellati tra cui scegliere – Ogni oggetto appartenente alla stessa categoria possiede le stesse proprietà, mantenendo omogenei i dati	– Necessario utilizzare gli ID a disposizione – Se nessuno degli ID disponibili è sufficiente a modellare l'oggetto, è necessario fare richiesta ad OMA per aggiungere un nuovo ID – La posizione di un dispositivo può essere espressa solo con il sistema di riferimento World Geodetic System 1984 – Per la modellazione di oggetti più complessi è necessario l'utilizzo degli oggetti compositi che rende complicato la comprensione dello standard ad utenti meno esperti

- **Haystack**

Pro	Contro
– Personalizzabile grazie alla libera scelta dei tag – Riduce al minimo le informazioni scambiate, mantenendo solo quelle significative per l'applicazione – I tag sono contrassegnati da nomi che rendono facile intuire il loro scopo – È possibile scegliere i tag da una lista precostruita oppure crearne nuovi	– Numerosi tag rende complicato il filtraggio dei dati – è necessario associare ad ogni dispositivo un numero arbitrario di tag per descriverlo scelti dall'utente – c'è il rischio che oggetti simili vengano modellati con tag diversi, rendendo i dati non omogenei

- **OCF**

Pro	Contro
– Database che modella un'ampia gamma di dispositivi – Possibilità di aggiungere proprietà personalizzate agli oggetti – Già supportato e utilizzato nel mercato da diverse aziende	– Molto complicato e poco intuitivo, comprendere un payload richiede una conoscenza approfondita dello standard – è necessario associare l'oggetto che si vuole modellare ad uno predefinito presente all'interno del database – Ogni tipologia di oggetto ha il suo tipo di interfaccia – Alcune proprietà dell'oggetto potrebbero necessitare di un inserimento manuale complicando la fase di installazione – Supporta solo il protocoppo CoAP

- **SensorThing**

Pro	Contro
– Ogni entità possiede delle proprietà fisse, rendendo facile l'installazione e mantenendo omogenei i dati scambiati – Semplice ed intuitivo – Supporto nativo alla geolocalizzazione – Supporto nativo a data e ora delle misurazioni – Basato sullo standard ISO 19156	– Poco personalizzabile – Alcune proprietà potrebbero risultare ridondanti per certi dispositivi

3.4 Scelta Standard

Confrontando l'analisi effettuata sugli standard con i requisiti richiesti dal sistema, l'unico che soddisfa completamente ogni punto è lo standard OGC SensorThings come mostrato nella tabella seguente.

	Ipso	Haystack	OCF	SensorThings
Implementazione open source	✓	✓	✓	✓
Flusso dati leggero	✓	✓	✓	✓
Modellabile agilmente		✓		✓
Semplicità d'utilizzo		✓		✓
Supporto a dati diversi	✓		✓	✓
Supporto nativo geolocalizzazione				✓

Come già accennato in precedenza, gli standard presentati rispettano in larga parte i requisiti richiesti e ognuno di loro potrebbe essere utilizzato all'interno del nostro sistema senza molte difficoltà. Si è scelto di procedere utilizzando lo standard OGC SensorThings poiché, oltre a soddisfare tutti i requisiti, possiede alcune caratteristiche intrinseche che lo rendono funzionale allo scopo dell'architettura che si vuole proporre.

Modellazione oggetti

Lo standard OGC SensorThings garantisce di modellare tutti gli oggetti nello stesso modo: ogni oggetto descritto con lo standard OGC SensorThings possiede le stesse tipologie di entità e di proprietà e ciò facilita l'omologazione dei dati ricevuti dai sensori. In particolare il dispositivo che si occuperà della "traduzione" dei dati non ha la necessità di associare ad ogni oggetto o proprietà un codice o un tag identificativo, ma è libero di poter modellare qualsiasi oggetto con la struttura fornita dallo standard OGC SensorThings senza il pericolo che le informazioni in suo possesso risultino obsolete o incomplete. Infatti, non dovendo archiviare nella sua memoria una lunga lista di codici e/o tag, non deve preoccuparsi che le informazioni inviate risultino errate causa aggiornamento/cambiamento all'intenro dello standard.

Geolocalizzazione

Un ulteriore vantaggio dell'utilizzo dello standard Sensorthings è la sua gestione della geolocalizzazione: a differenza degli altri standard analizzati, nei quali la posizione del dispositivo viene espressa attraverso una proprietà dell'oggetto, OGC SensorThings gestisce la geolocalizzazione degli oggetti in maniera nativa. In particolare, mentre con gli altri standard lo storico delle

posizioni degli oggetti deve essere gestita esternamente andando ad estrapolare le posizioni dalle varie proprietà, SensorThings raccoglie in maniera automatica tutte le posizioni occupate dai dispositivi inserendole in una "entità" chiamata `HISTORICAL LOCATION` ogni qual volta la posizione geografica di un dispositivo cambia. In un ambiente dove i sensori possono essere spostati o si muovono in autonomia, questa caratteristica risulta molto comoda ed efficiente per la raccolta e la visualizzazione dei dati.

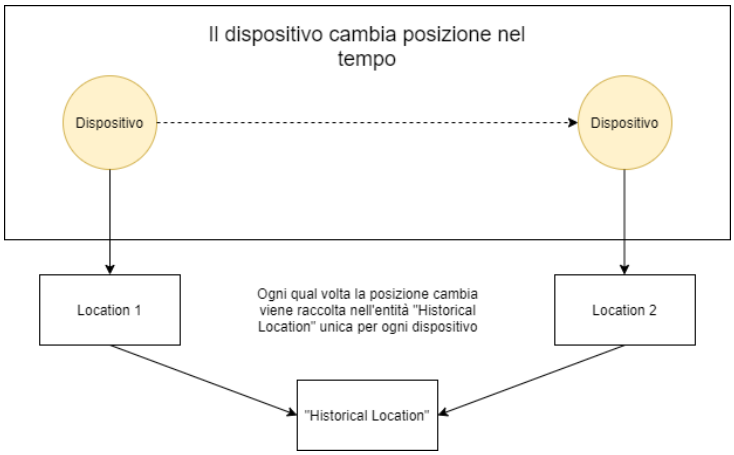


Figura 3.2 Esempio di raccolta delle posizioni di un dispositivo

Data e Tempo

Un altra caratteristica di rilievo per cui è stato scelto lo standard OGC SensorThings è il suo supporto nativo alla gestione di data e ora. Ogni osservazione effettuata dal dispositivo ed ogni posizione segnalata possiede una proprietà che include il timestamp della misurazione. Nel caso in cui il sensore non invii data e ora della misurazione, è compito del server che la riceve provvedere a fornirle come indicato da specifiche dello standard.
Nell'ambiente dell'agricoltura è fondamentale conoscere il momento della giornata in cui viene effettuata una misurazione ai fini dell'analisi, soprattutto se è necessario incrociare i dati per conoscere quale posizione occupava il sensore al momento della rilevazione.

Facilità d'uso

Ultimo fattore che ha influito sulla scelta dello standard OGC SensorThings è la sua facilità di utilizzo: le specifiche dello standard sono chiare ed intuitive,

le varie entità e proprietà hanno nomi che fanno capire il loro scopo e non ci sono grosse quantità di dati/codici da memorizzare per il suo utilizzo. Questa caratteristica è molto importante soprattutto in fase di installazione, quando parte dei dati deve essere inizializzata nel database ed essere inserita manualmente. Questa fase è affidata generalmente ad un utente che non possiede una conoscenza approfondita degli standard IoT ed inoltre è una fase delicata poiché un errore nell'inizializzazione dei dati può portare ad un malfunzionamente dell'intero sistema, perciò è necessario facilitare il più possibile questa fase per minimizzare al massimo il rischio di errore.

4 | Analisi dello standard OGC SensorThings

Dopo aver descritto lo standard in modo generale nel capitolo precedente, in questo capitolo si analizza più nel dettaglio le API dello standard OGC SensorThings
Lo standard è suddiviso in due parti: una prima parte che si dedica alla gestione dei sensori ed una seconda parte dedicata agli attuatori.
L'architettura proposta utilizza solamente la prima parte dello standard, quella dedicata ai sensori, ma in lavori futuri potrebbe essere implementata anche la seconda parte. La figura 4.1 rappresenta il modello relazionale con cui è strutturato lo standard.
Nell'immagine sono riassunte le entità necessarie alla creazione di un database OGC SensorThings. Per chiarezza sono state messe in risalto solamente le proprietà che devono essere esplicate obbligatoriamente per la creazione di quell'entità.
Le specifiche forniscono una breve descrizione delle entità che vanno a comporre lo schema:

- **Thing**: L'API OGC SensorThings segue la definizione ITU-T, ovvero, per quanto riguarda l'Internet of Things, una cosa è un oggetto del mondo fisico(cose fisiche) o del mondo dell'informazione(cose virtuali) che può essere identificato e integrato nelle reti di comunicazione. Proprietà obbligatorie:

  1. *Nome: String*
  2. *Descrizione: String*

- **Location**: L'entità `LOCATION` individua l'ultima posizione nota di una cosa Nel contesto dell'IoT, la principale posizione di interesse è solitamente associata alla posizione della Cosa, specialmente per le applicazioni di rilevamento in loco. Ad esempio, la posizione di interesse di un termostato connesso al Wi-Fi dovrebbe essere l'edificio o la stanza in

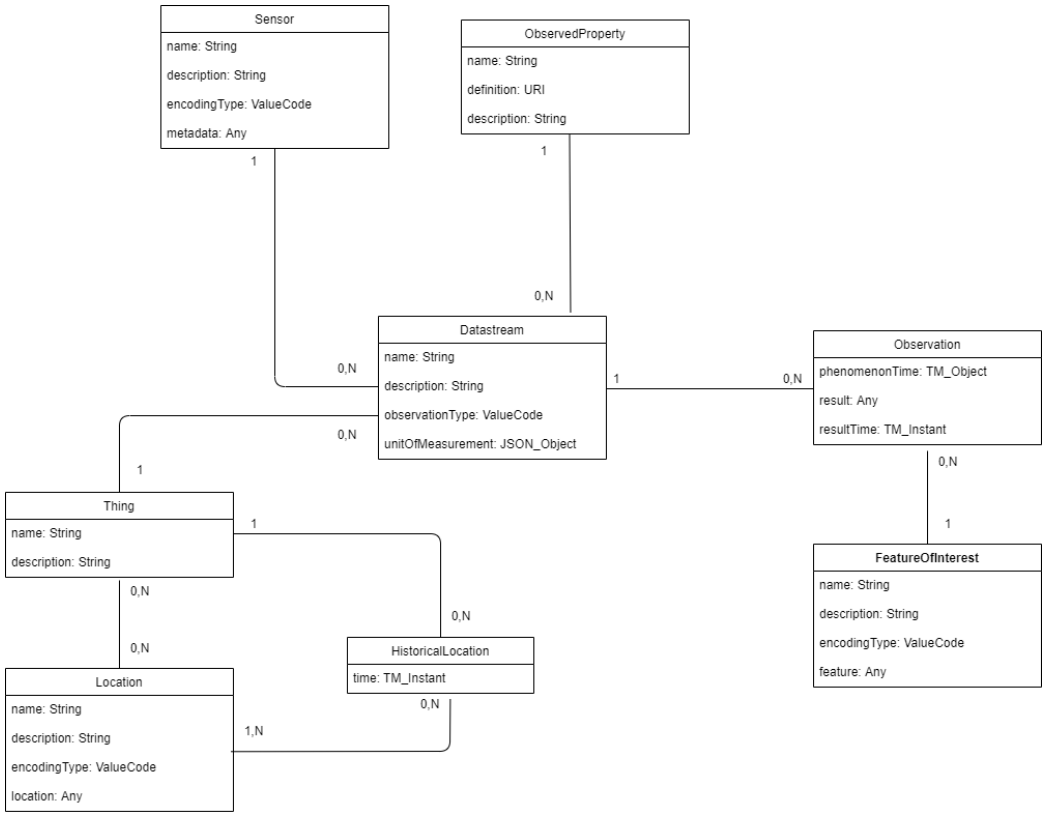


Figura 4.1 Schema relazionale

cui si trova il termostato intelligente. Tuttavia, la posizione di interesse finale di una Cosa non è sempre la posizione della Cosa. In questi casi d'uso, il contenuto della posizione di una cosa è diverso dal contenuto della caratteristica di interesse delle osservazioni della cosa. Proprietà obbligatorie:

1. *Nome: String*
2. *Descrizione: String*
3. *Tipo di encoding del campo Posizione: ValueCode (codice valore da tabella)*
4. *Posizione: Qualsiasi (coerente col tipo di coding selezionato)*

- **HistoriclLocation**: Il set di entità HistoricalLocation fornisce il timestamp della posizione corrente (ovvero l'ultima nota) e di quelle precedenti dell'entità Cosa associata. Proprietà obbligatorie:

1. *Timestamp: TM_istant*

- **Datastream**: Un `DATASTREAM` raggruppa una raccolta di osservazioni che misurano la stessa proprietà osservata e prodotte dallo stesso sensore. Proprietà obbligatorie:

  1. *Nome: String*
  2. *Descrizione: String*
  3. *Tipo di osservazione: ValueCode*
  4. *Unità di misura: Oggetto JSON contenente le proprietà dell'unità di misura*

- **Sensor**: Un sensore è uno strumento che osserva una proprietà o un fenomeno con l'obiettivo di produrre una stima del valore della proprietà. Proprietà obbligatorie:

  1. *Nome: String*
  2. *Descrizione: String*
  3. *Tipo di encoding del campo metadati: ValueCode*
  4. *Metadati: Qualsiasi (coerente col tipo di encoding selezionato)*

- **ObservedeProperty**: Un `OBSERVED PROPERTY` specifica il fenomeno di un'osservazione. Proprietà obbligatorie:

  1. *Nome: String*
  2. *Definizione: URI*
  3. *Descrizione: String*

- **Observation**: Un'osservazione è l'atto di misurare o altrimenti determinare il valore di una proprietà. Proprietà obbligatorie:

  1. *Timestamp del fenomeno osservato: TM_Object (ISO 8601 o Stringa di intervallo temporale) (corrisponde all'intervallo di tempo in cui si evolve il fenomeno osservato)*
  2. *Timestap della misurazione: TM_Istant (momento preciso in cui viene effettuata la misurazione)*
  3. *Valore: Qualsiasi*

- **FeatureOfInterest**: Un'osservazione si traduce in un valore assegnato a un fenomeno. Il fenomeno è una proprietà di un elemento, quest'ultimo è il `FEATURE OF INTEREST` dell'osservazione. Nel contesto dell'IoT, la `fFEATURE OF INTEREST` di molte osservazioni può essere la posizione della cosa. Ad esempio, la `fFEATURE OF INTEREST` di un termostato con connessione Wi-Fi può essere la posizione del termostato (ovvero il soggiorno in cui si trova il termostato). Nel caso del telerilevamento, `fFEATURE OF INTEREST` può essere l'area geografica o il volume che viene rilevato.

  1. *Nome: String*
  2. *Descrizione: String*
  3. *Tipo di encoding del campo Feature: ValueCode*
  4. *Feature: Qualsiasi (coerente col tipo di endoding selezionato)*

Interfaccia di Servizio

Le API di SensorThings implementano sia HTTP sia MQTT come protocolli per l'interfaccia al servizio. Il servizio API OGC SensorThings raggruppa gli stessi tipi di entità in set di entità, ciascuno dei quali ha un identificatore univoco. Nel caso in cui un'entità abbia una relazione con un set di entità a cui non appartiene, questa relazione è espressa grazie alle proprietà navigationLink e associationLink.
Pertanto, al fine di eseguire azioni CRUD sulle risorse, il primo passaggio è indirizzare le risorse di destinazione tramite URI. L'URI si compone di 3 elementi principali: il service root URI, il resource path e la query option.
Collegando il resource path dopo il service root URI, si possono indirizzare i diversi tipi di risorse, come ad esempio un set di entità, un'entità, una proprietà o una proprietà di navigazione. Infine, si possono elaborare ulteriormente le risorse indirizzate collegando le query option dopo il percorso della risorsa, andando così, per esempio, a filtrare o ad applicare un ordinamento ai dati ricevuti.

Richiesta dati

Attraverso il metodo `GET` del protocollo HTTP è possibile richiedere dati al server che verranno restituiti in formato JSON. Per indirizzare le diverse entità all'interno del database è sufficiente modificare l'URI utilizzato dal metodo `GET` nel seguente modo:

- **nessun path di risorsa**
  Es: *HTTP://example.org/v1.0/*
  Risposta: Un oggetto JSON con una proprietà denominata value che contiene un array in formato JSON contenente un elemento per ogni set di entità del servizio.
  Ogni elemento contiene almeno due valori, il primo è il nome del set di entità (ad esempio: `THING`,`DATASTREAM`,`OBSERVATION`, etc) mentre il secondo è un URI che indirizza il set all'interno del database.

- **indirizzo a una collezione di entità**
  Es: *HTTP://example.org/v1.0/ObservedProperties*
  Risposta: Un elenco di tutte le entità (con tutte le proprietà) nel set di entità specificato. La risposta è array JSON contenente una proprietà di nome value. Ogni elemento dell'array è un oggetto JSON e contiene la rappresentazione di ogni entità del set.

- **indirizzo a un'entità specifica**
  Es: *HTTP://example.org/v1.0/Things(1)*
  Risposta: Un oggetto JSON dell'entità con tutte le sue proprietà il cui id è quello specificato nell'URI.

- **indirizzo a una proprietà di un'entità specifica**
  Es: *HTTP://example.org/v1.0/Observations(1)/resultTime*
  Risposta: un oggetto in formato JSON contenente il nome della proprietà e il valore ad essa associato.

- **indirizzo al valore di una proprietà di un'entità specifica**
  Es: *HTTP://example.org/v1.0/Observations(1)/resultTime/$value*
  Risposta: una stringa contenente il valore associato alla proprietà.

- **indirizzo a una proprietà di navigazione (navigationLink)**
  Le entità in diversi set possono essere relazionate tra loro, queste relazioni possono essere richieste indirizzandosi a una proprietà di navigazione di un'entità.
  Es: *HTTP://example.org/v1.0/Datastreams(1)/Observations*
  Risposta: Un oggetto JSON di un'entità o un array JSON di molte entità del set indicato nella parte finale dell'URI che hanno una relazione con l'entità specificata.

- **indirizzo ad un associationLink**
  Un associationLink può essere utilizzato per recuperare un riferimento a un'entità o un set di entità relazionato all'entità corrente.
  Es: *HTTP://example.org/v1.0/Datastreams(1)/Observations/$ref*

Risposta: Un oggetto JSON con una proprietà value. Il valore della proprietà value è un array JSON contenente un elemento per ogni associationLink. Ogni elemento è un oggetto JSON con una proprietà selfLink a cui è associato l'URI di una delle entità che si relazionano.

- **resource path annidato**
  I vari URI analizzati possono essere innestati tra loro per indirizzare l'oggetto che si vuole ricevere.
  Es: *HTTP://example.org/v1.0/Datastreams(1)/Observations(1)*
  Risposta: si ottiene una delle risposte viste in precedenza in base all'elemento indirizzato nella parte finale dell'URI.

Operazioni CRUD

Le operazioni di CRUD (Create, Update, Delete) utilizzano i metodi `POST,PATCH,PUT,DELETE` di HTTP in modo simile alle operazioni viste in precedenza, con la differenza che alcune chiamate a questi metodi incorporano nel body le informazioni necessarie a completare l'operazione.

- **CREATE**
  Per creare un'entità in un set, il client deve inviare una richiesta `POST` all'URL di quel set. Il body della richiesta `POST` deve contenere un'unica rappresentazione di un'entità valida in un oggetto JSON. La creazione dell'entità deve sottostare ai vincoli relazionali visti nel diagramma della figura.
  In caso l'entità venga creata con successo, il server risponde fornendo l'autoLink dell'entità creata.
  Inoltre, il collegamento tra le entità deve essere stabilito al momento della creazione di un'entità. Devono essere considerati due casi d'uso: (1) collegamento a entità esistenti durante la creazione di un'entità e (2) creazione di entità correlate durante la creazione di un'entità. Le richieste per questi due casi d'uso sono descritte nella sottosezione seguente.
  Quando un client crea una risorsa in un servizio SensorThings, deve seguire i vincoli di integrità elencati nella Tabella 4.1. Ad esempio, un'entità `DATASTREAM` deve collegarsi a un'entità `THING`. Quando un client desidera creare un'entità `DATASTREAM`, deve (1) creare un'entità `THING` collegata nella stessa richiesta o (2) collegarsi a un'entità `THING` già creata.

Scenario	**Vincoli di integrità**
Creazione di un'entità `THING`	Nessun vincolo
Creazione di un'entità `LOCATION`	Nessun vincolo
Creazione di un'entità `DATASTREAM`	• Deve essere collegato ad una entità `THING` • Deve essere collegato ad una entità `SENSOR` • Deve essere collegato ad una entità `OBSERVED PROPERTY`
Creazione di un'entità `SENSOR`	Nessun vincolo
Creazione di un'entità `OBSERVED PROPERTY`	Nessun vincolo
Creazione di un'entità `OBSERVATION`	• Deve essere collegata ad una entità `DATASTREAM` • Deve essere collegata ad una entità `FEATURE OF INTEREST`. Nel caso in cui non sia presente, il sistema centrale provvederà a crearne una derivandola dalla entità `LOCATION`
Creazione di un'entità `FEATURE OF INTEREST`	Nessun vincolo

Tabella 4.1

- **UPDATE**
  Per aggiornare le proprietà di un'entità all'interno del database il client invia una richiesta con metodo `PATCH/PUT` all'URI che indirizza l'entità con un body contenente le proprietà che devono essere aggiornate con i rispettivi nuovi valori in formato JSON.
  Il metodo `PATCH` è preferibile rispetto al metodo `PUT` poiché aggiorna solo le proprietà specificate nel body lasciando invariate le rimanenti, mentre il metodo `PUT` sostituisce l'intera entità con quella specificata nel body, potendo causare una possibile perdita di informazioni.

- **DELETE**
  L'eliminazione di un'entità avviene attraverso una richiesta con metodo `DELETE` all'URI che indirizza l'entità che si vuole eliminare. Il body della richiesta è vuoto.
  Per garantire l'integrità del database, all'eliminazione di un'entità seguono l'eliminazione, in modo implicito o esplicito, di tutte le istanze delle altre entità ad essa collegata secondo i vincoli forniti dalla tabella 4.2.

Scenario	**Vincoli di integrità**
Rimozione di un'entità `THING`	Rimozione di tutte le entità `DATASTREAM` collegate a `THING`
Rimozione di un'entità `LOCATION`	Rimozione di tutte le entità HistoricalLocation collegate a `LOCATION`
Rimozione di un'entità `DATASTREAM`	Rimozione di tutte le entità `OBSERVATION` collegate a `DATASTREAM`
Rimozione di un'entità `SENSOR`	Rimozione di tutte le entità `DATASTREAM` collegate a `SENSOR`
Rimozione di un'entità `OBSERVED PROPERTY`	Rimozione di tutte le entità `DATASTREAM` collegate a `OBSERVED PROPERTY`
Rimozione di un'entità `OBSERVATION`	Nessun vincolo
Rimozione di un'entità `FEATURE OF INTEREST`	Rimozione di tutte le entità `OBSERVATION` collegate a `FEATURE OF INTEREST`
Rimozione di un'entità `HISTORICAL LOCATION`	Nesun vincolo

Tabella 4.2

5 | Progetto

In questo capitolo viene proposta una possibile architettura che prevede l'utilizzo dello standard OGC SensorThings per raccogliere i dati non standardizzati provenienti dai sensori. Per convertire tra i diversi formati il progetto prevede l'introduzione di strumenti aggiuntivi che si pongono come intermediario tra i sensori e il database di raccolta dati. Il punto focale del progetto è quindi la predisposizione di un dispositivo, che d'ora in avanti chiameremo "connettore", che raccolga i dati inviati dai vari gateway, li converta secondo le specifiche dello standard e li invii al database centrale. Al database finale potrà infine essere collegato un client a scelta con cui visualizzare ed elaborare i dati.

5.1 Requisiti funzionali

Sono stati individuati i requisiti che l'architettura di sistema deve rispettare. Alcuni di essi vengono soddisfatti grazie all'utilizzo dello standard scelto, mentre altri sono intrinsechi della tecnologia utilizzata. L'architettura proposta mira ad essere la più "adattibile" possibile ai tipi di esigenze in cui si può incombere nell'ambito dell'agricoltura di precisione.

- Assenza di vincoli di modello nell'impiego di sensori
- Supporto a vari protocolli di comunicazione
- Garanzia omogeneità dati
- Facilmente scalabile
- Possibilità di archiviazione dati
- Possibilità di visualizzazione dati secondo filtri dettati dall'utente
- Monitoraggio dello stato dei sensori

- Semplicità d'utilizzo
- Utilizzo di software open source
- Accessibilità da dispositivi mobili
- Autonomo con manutenzione ordinaria ridotta al minimo

5.2 Architettura generale

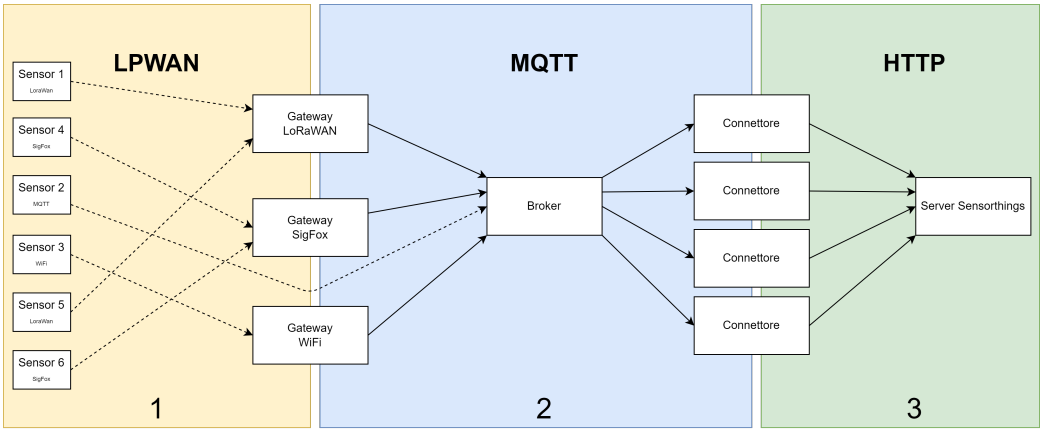


Figura 5.1

L'architettura proposta è sintetizzata nella figura 5.1. Lo schema mostra il flusso comunicativo tra i dispositivi e mette in risalto i protocolli utilizzati nelle varie sezioni di cui è composto.
Le sezioni di cui è composto il sistema sono 3: la prima sezione rappresenta i vari sensori posizionati nei terreni agricoli che periodicamente inviano i dati ai gateway, la seconda comprende i gateway che comunicano con i connettori che fungono da "traduttori" per i dati che vengono inviati alla terza sezione che comprende il server e il database centrale.
Per quanto riguarda la prima sezione, abbiamo supposto che i sensori siano di tipo eterogeneo tra loro, prodotti da aziende diverse e con diversi metodi di comunicazione. I vari dispositivi comunicano con il gateway associato attraverso tecnologie differenti, che possono appartenere alla categoria LPWAN (low-power wide-area network), come, ad esempio tra quelle più utilizzate in commercio, LoRaWAN o SigFox, oppure utilizzare un collegamento wireless alla rete internet.

Nella seconda parte, i gateway fungono da ponte tra i protocolli non IP, con cui comunicano con i sensori, e MQTT, protocollo con cui comunica con i connettori. I gateway, che, contrariamente ai sensori, non hanno particolari problematiche di autonomia, sono collegati alla rete internet ed il loro compito è di inoltrare i dati ricevuti dai sensori al broker che a sua volta pubblicherà i messaggi sui rispettivi topic.
I connettori vengono utilizzati per raccogliere i dati eterogenei provenienti dai vari gateway, convertirli in formato OGC SensorThings e reindirizzarli al server attraverso il protocollo RESTful.
Nella rete che si vuole andare a creare viene predisposto un connettore ad hoc per ogni modello di sensore collegato alla rete. Nella rete MQTT i gateway sono sostanzialmente i "publisher" mentre i connettori ricoprono il ruolo di "subscriber". Un gateway pubblica nel "topic" corrispettivo al modello del sensore da cui ha ricevuto il payload. Un connettore si "mette in ascolto" solo sul topic su cui vengono pubblicati i payload che può convertire. L'utilizzo di MQTT permette di mantenere la dimensione dei payload ridotta e consente ad alcuni sensori di comunicare direttamente con il connettore "scavalcando" il gateway, se dotati di connessione ad internet autonoma. Grazie a questa configurazione non è necessario un controllo dei dati da parte del connettore, poiché è garantito l'arrivo di sole informazioni traducibili da parte di quest'ultimo.
Un ulteriore vantaggio di questa configurazione è la facilità della scalabilità del sistema: infatti, nel caso in cui si vogliano aggiungere ulteriori modelli di sensori alla rete in un secondo momento, basterà aggiungere un ulteriore "topic" alla rete MQTT ed associare ad esso un connettore, senza andare a modificare le configurazioni preesistenti.
Nella terza sezione il server implementa uno dei software opensource disponibili per la gestione di un database OGC SensorThings. Successivamente verranno analizzate le alternative disponibili allo stato attuale e si sceglierà quella più adatta al sistema proposto. Queste implementazioni utilizzano un database SQL per l'immagazzinamento delle informazioni con supporto alla gestione delle feature geografiche spaziali. Al server centrale può essere infine collegato un qualsiasi client che possa richiedere, elaborare e visualizzare i dati contenuti nel database.

Sensori e gateway

Il punto di partenza del sistema coincide con i sensori posizionati nei terreni agricoli in punti strategici in modo da ottenere un campione di dati che sia il più possibile significativo per gli utilizzi e le analisi che si effettueranno in seguito. Questi sensori, come descritto in precedenza, possono avere diversi

metodi per la raccolta dei dati ed utilizzare protocolli diversi tra loro per la comunicazione. L'utente è libero di poter scegliere e aggiungere al sistema i sensori che ritiene più adatti allo scopo richiesto. In fase di installazione i sensori vengono collegati al gateway associato in base al loro protocollo di comunicazione, nel caso in cui un sensore sia dotato di collegamento a internet autonomo invece viene messo in comunicazione direttamente col il broker. Non potendo limitare i tipi di protocolli utilizzati altrimenti verrebbe meno lo scopo ultimo del sistema, all'interno di esso viene predisposto un gateway per ogni tipo di protocollo di comunicazione utilizzato dai sensori.
In commercio esistono numerose tipologie di gateway che possono essere configurati a piacimento per inoltrare i pacchetti ricevuti lungo un canale predefinito come abbiamo analizzato nei capitoli precedenti. Il gateway necessario al nostro sistema deve avere la possibilità di comunicare con un broker MQTT. Lo scopo del gateway è ricevere i dati dai sensori collegati, individuare da quale sensore ha ricevuto i dati e pubblicarli nel "topic" associato. Come accennato in precedenza, esiste un "topic" per ogni tipologia di sensore presente nel sistema. Un gateway può pubblicare su diversi "topic" poiché ad esso possono essere collegati sensori di aziende produttrici diverse.
Non è compito del gateway gestire i duplicati di pacchetti o la corretta ricezione delle informazioni. Questi compiti sono delegati al server in base al livello di QoS che si vuole mantenere all'interno del sistema.

Connettori

Il punto principale del progetto si articola nell'introduzione e lo sviluppo dei connettori. Il connettore è un software che rende possibile la comunicazione tra i vari sensori posizionati nel mondo reale e l'utente finale. Si predispongono nel sistema tanti connettori quanti sono le diverse tipologie di dato da convertire: per esempio, se nel sistema sono presenti 5 tipologie di sensori prodotti da aziende diverse con diverse formattazioni dei payload verranno predisposti 5 connettori.
Predisporre connettori multipli porta dei vantaggi al sistema rispetto alla configurazione a singolo connettore:

- Allegerimento del carico di lavoro sul connettore: il connettore è in ascolto in un canale in cui ha la sicurezza che i dati ricevuti siano da esso convertibili, quindi non c'è bisogno di verifica dell'origine dei dati.
- Scalabilità del sistema: se si presenta la necessità di aggiungere o sostituire sensori al sistema, basterà operare solamente con i connettori coinvolti, aggiungendone di nuovi se necessario, senza andare a toccare gli altri connettori.

- Manutenzione software: in caso di bug o aggiornamento di firmware dei sensori, è sufficiente aggiornare il software del singolo connettore associato.

Come introdotto in precedenza, lo scopo di un connettore è quello di ricevere i dati, convertirli e poi inviarli al database centrale. Il suo funzionamento consiste in 4 fasi:

1. **Attesa dei dati**: il connettore rimane in attesa di ricevere dati sul "topic" al quale è stato abbonato. È compito del broker fare da tramite tra gateway e connettori. Ogni connettore è abbonato ad uno ed un solo "topic" per i vantaggi visti in precedenza.

2. **Controllo dei dati**: All'arrivo dei dati il connettore controlla che quest'ultimi siano integri e che non ci siano errori nel payload. In caso contrario, i dati ricevuti vengono scartati.

3. **Conversione dei dati**: Successivamente al controllo di integrità dei dati, in caso di successo, avviene la conversione. Nella pratica il connettore legge i dati, grazie ad uno schema predefinito associato alla tipologia li converte in valori alfanumerici adatti alle specifiche dello standard OGC SensorThings e li raccoglie in un oggetto JSON da inviare al database centrale.

4. **Invio dei dati**: L'oggetto JSON viene inviato al server OGC SensorThings alla corrispondente entità con i metodi già descritti nei paragrafi precedenti.

Il funzionamento del connettore è riassunto nel diagramma delle attività nella figura 5.2. Il diagramma mostra tutti i passaggi descritti fino a qui e i punti in cui il connettore deve effettuare delle scelte riguardo ai dati in arrivo. Nella figura 5.3 è rappresentato il diagramma di sequenza della comunicazione che avviene a partire dal sensore fino ad arrivare al database.

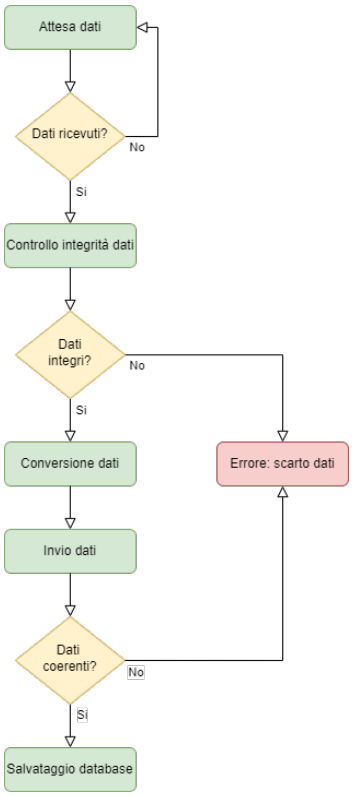


Figura 5.2

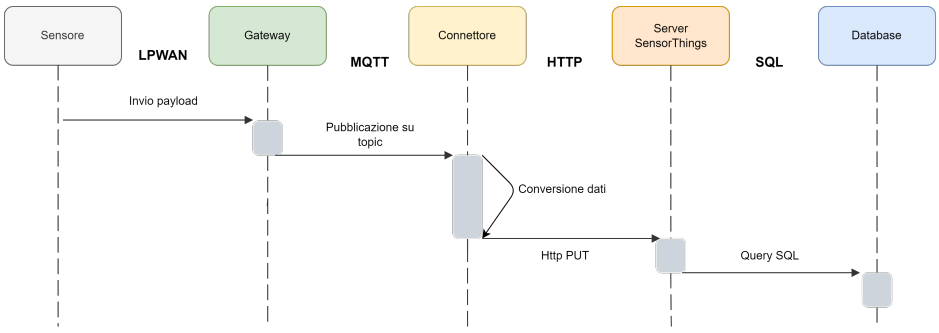


Figura 5.3 Sequence diagram comunicazione dal sensore al database

Server e client

La parte finale del sistema si articola nel server centrale che si occupa di gestire lo scambio di informazioni tra database e i client richiedenti.
Il server centrale deve rispondere principalmente a due tipologie di eventi esterni:

- **Richiesta di dati**: a prescindere dalla provenienza della richiesta, il server controlla che la richiesta sia formulata correttamente secondo le specifiche dettate dallo standard, in caso affermativo converte la richesta HTTP in una query SQL da sottoporre al database per estrarre i dati richiesti ed inoltrarli nel formato specificato dallo standard all'originr della richiesta. Nel caso in cui la richiesta sia formulata in modo errato o i dati richiesti non siano presenti nel database, il server interrompe la comunicazione notificando l'errore.

- **Pubblicazione di dati**: il server controlla che la richiesta e la formattazione dei dati siano corretti, in caso affermativo consulta il database per verificare che i dati inviati siano coerenti con i dati all'interno di esso. Il server comunica l'esito positivo se tutte le operazioni vanno a buon fine, in caso contrario, come in precedenza, il server interrompe la comunicazione notificando l'errore.

Un grande vantaggio che l'API SensorThings offre ai client è la potente funzionalità di query e ricerca che incorpora. Anche scoprire che tipo di dati viene servito da un'istanza STA è banale. Nella maggior parte dei casi è sufficiente un semplice browser web, poiché tutte le entità possono essere raggiunte semplicemente seguendo i collegamenti di navigazione che l'API offre. Attraverso il client possono essere eseguite le richieste dei dati seguendo le specifiche delle API messe a disposizione dallo standard SensorThings. Lo scopo del client è quello di fornire uno strumento per l'estrazione e l'analisi dei dati.
Per la visualizzazione dei dati geospaziali attraverso una cartina geografica, sarà compito del client estrarre le coordinate dai dati ricevuti e richiedere al server centrale (se esso mette a disposizione questo servizio implementando software per la gestione dei dati geospaziali come, ad esempio, GeoServer) la mappa relativa.

5.3 Analisi software

Uno dei requisiti funzionali del sistema è l'utilizzo di software open source. Allo stato attuale esistono poche alternative open source per l'implementazio-

ne del server OGC SensorThings. Molte di esse non vengono più supportate, sono incomplete o abbandonate da tempo. Tuttavia alcuni progetti sono ancora supportati e disponibili al pubblico senza licenze.
Per il progetto sono state selezionate ed analizzate le implementazioni più complete offerte al momento attuale:

- **Whiskers**: *Whiskers* è un framework API OGC SensorThings. È composto da un client JavaScript e un server leggero per dispositivi gateway IoT (ad esempio, *Raspberry Pi*). *Whiskers* mira a promuovere un ecosistema IoT sano e aperto, al contrario di quello dominato dai protocolli informativi proprietari.

Pro	**Contro**
– Mette a disposizione un client opensource per la comunicazione con la parte server – Supporto a dispositivi con risorse limitate (*Raspberry Pi*, *BeagleBone*)	– Supporto solamente parziale alle entità dello standard

- **FROST**: *FROST-Server* è un'implementazione server Open Source dell'OGC SensorThings API. il software è composto di 2 parti: la prima, *FROST-Server*, implementa l'intera specifica descritta dallo standard, comprese tutte le estensioni aggiunte in seguito. È scritto in Java e può essere eseguito attraverso *Tomcat* o *Wildfly* ed è disponibile come immagine *Docker*. Tra le sue numerose funzionalità c'è la possibilità di utilizzare ID entità basati su String o UUID. La seconda invece, chiamata *FROST-Client*, è una libreria client Java per la comunicazione con un server compatibile con l'API SensorThings.

Pro	Contro
– Fornisce un pacchetto completo server+client – Supporto ad MQTT e MQTTp – Installazione facile e veloce attraverso Docker – Ben documentato – Supporto all'autenticazione integrato	– Client non supporta il protocollo MQTT – Client non suppporta le "batch requests" dello standard

- **Mozilla STA**: *Mozilla* ha sviluppato un'implementazione del nodo dell'API OGC SensorThings.

Pro	Contro
– Implementa tutte le funzionalità di base previste dallo standard	– Nessun client – Progetto abbandonato da alcuni anni

- **52°North STA**: *52N SensorThingsAPI* è un'implementazione open source dell'OGC SensorThings API. Le sue caratteristiche principali sono l'interoperabilità con il *52N SOS* che implementa il servizio di osservazione dei sensori OGC, mappature di database personalizzabili e numerose estensioni. Può essere distribuito come container *Docker*, all'interno di un *Apache, Tomcat* o come applicazione autonoma.

Pro	Contro
– Supporto a Docker – Supporto a MQTT – Buona documentazione – interoperabilità con il servizio 52N SOS	– Nessun client – Non supporta l'estensione MultiDatastream

Per lo scopo del progetto si è scelto di utilizzare il software *FROST Server*. Rispetto ai suoi concorrenti, il software offerto dall'istituto Fraunhofer è più maturo e completo, ed implementa tutte le specifiche descritte nello standard OGC SensorThings. Inoltre mette a disposizione delle librerie con cui interfacciarsi ad esso che semplificano lo sviluppo di applicazioni che utilizzano lo standard. Le librerie aiutano a concentrarsi sullo sviluppo dell'applicativo mascherando con chiamate semplificate tutte le operazioni CRUD analizzate in precedenza.

5.4 Use case

In questa sezione si analizza il comportamento tipico di un utente nel sistema porposto. La figura 5.4 mostra lo "use case schema" dove sono rappresentati i vari utenti, l'utilizzo che fanno del sistema e i loro permessi.

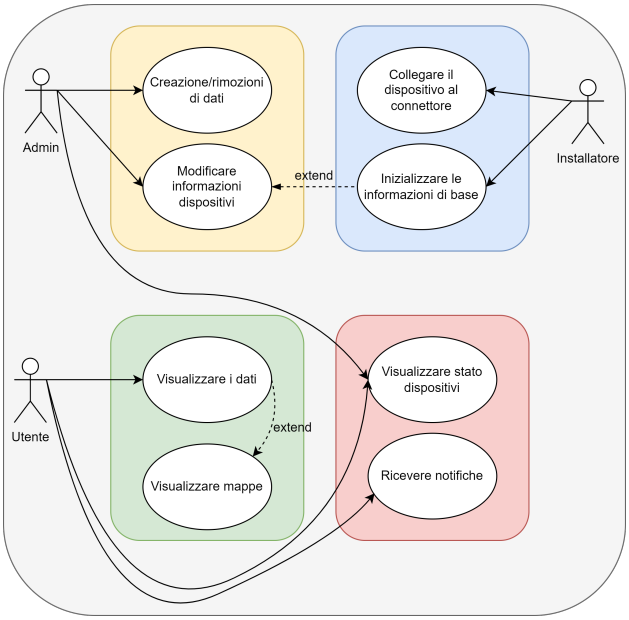


Figura 5.4 Use case schema del sistema

- **Utente**: Un utente del sistema è un individuo che si affaccia al sistema quotidianamente per un utilizzo standard. L'utente ha la possibilità di accedere ai dati raccolti dai sensori attraverso un client collegato al server centrale. I dati, oltre ad essere elencati e filtrati in base alle esigenze dell'utente, possono essere supportati da un'interfaccia grafica che ne aiuti la visualizzazione attraverso l'utilizzo di strumenti quali mappe, grafici, etc.

- **Amministratore**: L'amministratore è colui che gestisce e monitora il sistema L'amministratore si occupa della gestione del sistema, si assicura che l'integrità del database venga mantenuta e interviene se è necessario modificare le informazioni contenute all'interno di esso. È l'unica figura con il privilegio di poter eliminare e/o modificare i dati contenuti nel sistema.

- **Installatore**: L'installatore è la figura che si occupa dell'installazione di nuovi sensori al sistema. Si occupa del collegamento tra sensori e connettori e ha il privilegio di poter creare nuove entità nel database centrale da associare ai nuovi sensori.

I modi più comuni per interfacciarsi al sistema da parte degli utenti sono due: la richiesta e visualizzazione dei dati e l'invio di dati al database.
La figura 5.5 mostra il diagramma delle attività per l'operazione di ricezione dei dati descrivendo i vari passaggi che avvengono nel sistema alla richiesta formulata dall'utente. In maniera analoga la figura 5.6 mostra il diagramma delle attività per l'invio dei dati da parte dell'utente autorizzato.
Nelle figure 5.7 e 5.8 sono presentati i diagrammi di sequenza per le due operazioni di invio e richiesta dati che descrivono come le varie parti comunicano nello svolgimento delle due operazioni.

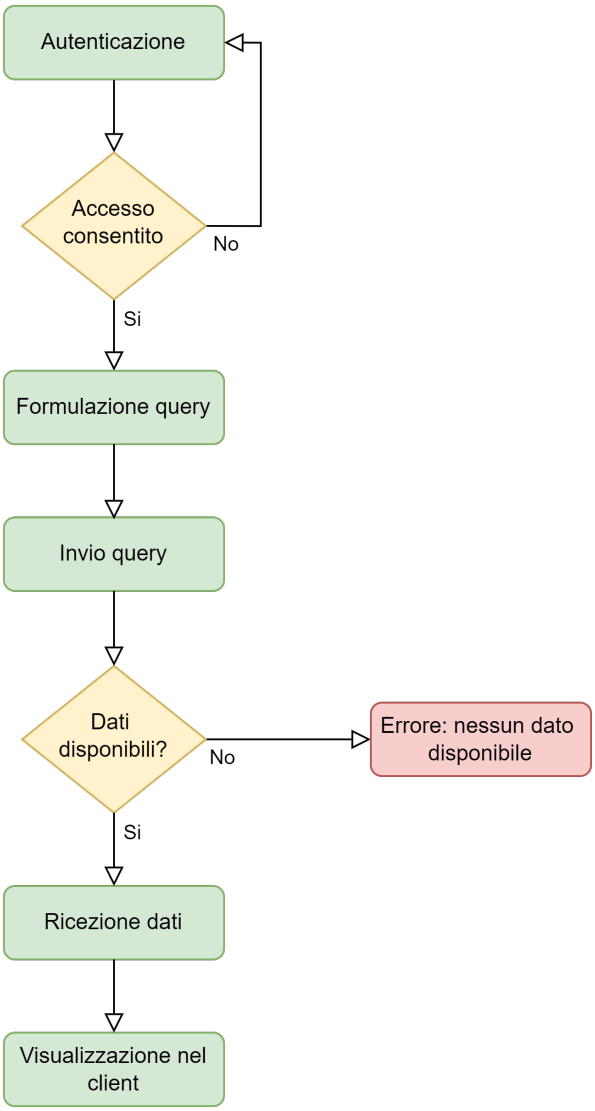


Figura 5.5 L'utente esegue l'accesso, formula la query che viene tradotta dal client in standard OGC e rimane in attesa dell'esito da parte del server. In caso di esito positivo vedrà visualizzati i dati nel client.

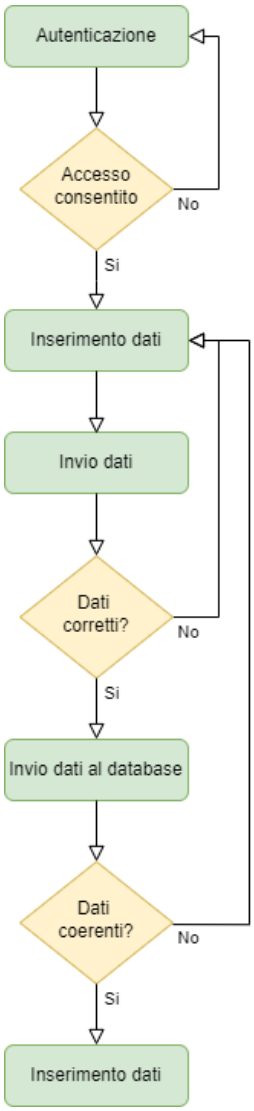


Figura 5.6 L'utente esegue l'accesso con privilegi per l'invio ed invia al server i dati, i quali verranno controllati di essere in formato corretto e coerenti con il database prima di essere salvati su quest'ultimo.

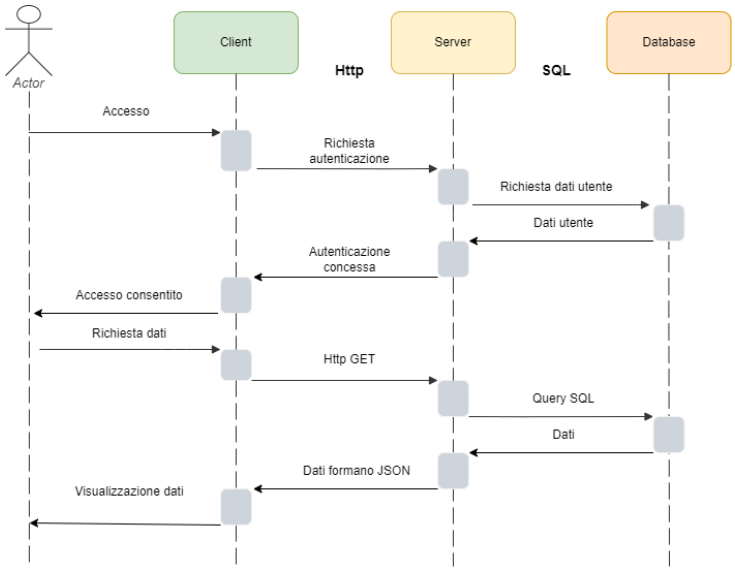


Figura 5.7 Sequence diagram richiesta dati da parte dell'utente

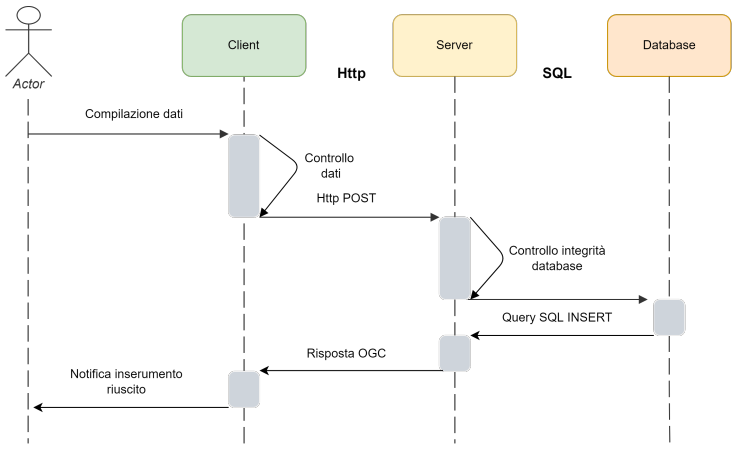


Figura 5.8 Sequence diagram invio dati da parte dell'utente

Esempio di use case con standard OGC SensorThing

Utente: l'utente riceve e visualizza i dati attraverso il client. Il client comunica con il server attraverso il protocollo HTTP. Le richieste verso il server vengono effettuate dal client attraverso l'interfaccia di servizio dello standard SensorThing in modo trasparente rispetto all'utente.
Per esempio, quando l'utente vorrà visualizzare i primi 10 risultati dell'en-

tità “Observations” ordinati per ID, il client effettuerà una richiesta `GET` all'indirizzo del server:

```
GET HTTP://example.org/v1.0/Observations?$orderby=ID&$top=10
```

Installatore: l'installatore crea le nuove entità necessarie all'interno del database affinchè il sensore possa essere collegato al sistema. Per fare ciò utilizza le operazioni di CRUD messe a disposizione dall'interfaccia di servizio di SensorThing.
Attraverso il metodo `POST` del protocollo HTTP può inviare al server le varie istanze delle entità necessarie all'inizializzazione del sensore all'interno del sistema.
Per esempio, se l'installatore deve installare un sensore con le seguenti specifiche:

- Nome: Sens-IA12
- Posizione: [-114.06,51.05]
- Tipo sensore: misuratore pressione atmosferica

invierà il seguente oggetto JSON attraverso il metodo `POST` all'indirizzo

`HTTP://example.org/v1.0/Things`

con body

```
{
     name:  "Sens-IA12",
     description:  "Sensore per la misurazione della pressione atmo-
sferica",
}
```

e aggiornerà la sua posizione inviando una nuova entità `LOCATION` all'indirizzo

`HTTP://example.org/v1.0/Locations`

con body

```
{
      encodingType :  "application/vnd.geo+json",
      name :  "pmPd5",
      description :  "Postazione metereologica di Padova n.  5",
      location :  {
            type :  "Feature",
            geometry :{
            type :  "Point", "coordinates":  [-114.06,51.05]
            }
      }
}
```

Admin: l'amministratore del sistema interviene quando è necessario modificare e/o eliminare dati all'interno del database. Per queste operazioni si avvale dei metodi `PATCH/PUT` e `DELETE`. Per esempio: l'amministratore vuole modificare un'istanza dell'entità `THING`, in particolare vuole modificare la proprietà `description`. Invia il seguente oggetto `JSON` attraverso il metodo `PATCH/PUT` all'indirizzo

`HTTP://example.org/v1.0/Things(1)`

con body

```
{
      description :"Sensore per la misurazione della temperatura."
}
```

La proprietà "description" dell'istanza viene sovrascritta e sostituita con con quella inviata dall'amministratore.

6 | Prototipo

È stato creato un prototipo che simula una rete formata da sensori di produttori differenti comunicanti con il loro rispettivo connettore e un server per la raccolta dei dati.
Il prototipo è costituito da:

- Un'istanza di *FROST-Server* che simula le funzioni del server di raccolta dati
- Un'istanza di *Mosquitto* come brocker MQTT
- Un applicativo java che simula l'invio di payload randomici sostituendosi ai sensori reali
- Un'applicativo java per la simulazione del comportamento dei connettori

Per la parte della simulazione dei sensori sono state utilizzate le librerie `PAHO` di *Eclipse* per la comunicazione MQTT. Le classi generano dei payload randomici, ma con i vincoli dettati dalla sintassi imposta dal produttore, ed utilizzano la tecnologia MQTT per comunicare con il broker, per il quale in questo caso è stata scelta un'istanza di `Mosquitto` per la sua facilità d'uso.
Per la seconda parte, quella dei connettori, sono state utilizzate le librerie `FROST-client` per la creazione delle entità e la comunicazione con il server. Il vantaggio dell'utilizzo di queste librerie è il mascheramento delle operazioni di `POST` del protocollo HTTP che viene affidato interamente alla libreria, semplificando molto sia la creazione di nuove entità sia la loro ricerca all'interno del server.
Infine per quanto riguarda il server, si è scelto di utilizzare un'istanza di `FROST-server` attraverso la distribuzione *Docker* fornita direttamente dagli autori.

6.1 Modello generale

Come espresso in precedenza, nel prototipo è presente una classe che periodicamente pubblica nei topic predefiniti i payload dei sensori relativi. Questa classe simula il comportamento dei gateway che nella rete reale si posizionerebbero come "ponte" comunicativo tra il sensore stesso e il broker. È compito dei gateway infatti raccogliere le informazioni provenienti dai sensori e incapsularle nei pacchetti MQTT da pubblicare.
Prima di analizzare il funzionamento del prototipo, è necessario elencare i requisiti necessari ad un suo corretto funzionamento:

- Ogni entità `DATASTREAM` associata ad un dispositivo deve essere identificata da un nome univoco.
- All'interno del database OGC SensorThings devono essere precaricate le entità `THING` associate ai dispositivi. Nel caso in cui non fossero presenti, le informazioni disponibili alla loro creazione devono essere precaricate nel database interno del connettore.
- All'interno del database OGC SensorThings devono essere precaricate le entità `OBSERVED PROPERTY` associate ai `DATASTREAM`. Nel caso in cui non fossero presenti, le informazioni disponibili alla loro creazione devono essere precaricate nel database interno del connettore.
- All'interno del database OGC SensorThings devono essere precaricate le entità `SENSOR` associate ai `DATASTREAM`. Nel caso in cui non fossero presenti, le informazioni disponibili alla loro creazione devono essere precaricate nel database interno del connettore.
- All'interno del database OGC SensorThings devono essere precaricate le entità `FEATURE OF INTEREST` associate al `DATASTREAM` specifico del dispositivo. Nel caso in cui non fossero presenti, il sistema centrale provvederà alla loro creazione derivandole dalle entità `LOCATION`

L'utilizzo di database interni al dispositivo ha il vantaggio di poter aggiungere e aggiornare le informazioni che lo riguardano senza dover modificare il codice del connettore. All'interno del prototipo il database interno è stato realizzato con `SQLite`, che permette di creare database di dimensioni ridotte utili al nostro scopo ed inoltre mette a disposizione delle librerie con cui interfacciarsi.
La prima operazione che compie il client una volta ricevuto il messaggio è identificare da quale dispositivo è stato inviato. Per far ciò, il client deve conoscere l'ID unico associato al dispositivo. Questo ID può essere inviato

dal sensore stesso oppure essere aggiunto dal gateway nel caso in cui il modello di sensore non incorporasse l'informazione di default. Successivamente il connettore interroga il proprio database interno per scoprire a quale entità del database è associato il dispositivo. L'entità associata al dispositivo è un'istanza dell'entità `THING` del modello OGC SenosrThings. È necessario conoscere l'entità `THING` a cui è associato il dispositivo poiché attraverso di essa possono essere ricavati i `DATASTREAM` relativi e quindi pubblicare le osservazioni relative dei sensori.
In seguito si passa alla fase di "traduzione" dell'informazione: questa operazione è specifica per ogni modello di sensore ed è quella che va a caratterizzare ogni connettore. In seguito verranno presentati degli esempi che tratteranno più nello specifico questa operazione, per ora ci limiteremo a descrivere in termini generali le operazioni compiute da ogni connettore:

1. Il connettore estrae i dati relativi alle misurazioni dal payload ricevuto
2. li interpreta secondo le linee guida dettate dal produttore
3. ricava la tipologia di misurazione ed i suoi valori
4. cerca tra i `DATASTREAM` associati all'entità `THING` quello associato alla misurazione
5. se non presente, crea una nuova entità `DATASTREAM` con le informazioni presenti all'interno del suo database: per prima cosa estrae dal database interno le entità `SENSOR` e `OBSERVED PROPERTY` relative al `DATASTREAM` da creare, successivamente estrae i dati relativi alle proprietà di quest'ultimo e ne completa la creazione. Nell'eventualità in cui una o più entità `SENSOR` o `OBSERVED PROPERTY` non siano disponibili, il connettore si occuperà di crearne di nuove con i dati in possesso nel suo database.
6. pubblica una nuova `OBSERVATION` collegata al `DATASTREAM` relativo contente i valori ricevuti e le associa la `FEATURE OF INTEREST` relativa.

I passaggi relativi alla creazione e collegamento delle entità `SENSOR` e `OBSERVED PROPERTY` vengono illustrati solo nel primo esempio per essere successivamente omessi per chiarezza di spiegazione.
Nel caso in cui la rilevazione inviata sia la posizione geografica del sensore invece, al posto di pubblicare una `OBSERVATION` verrà creata una nuova istanza dell'entità `LOCATION` da associare all'istanza `THING` del sensore.

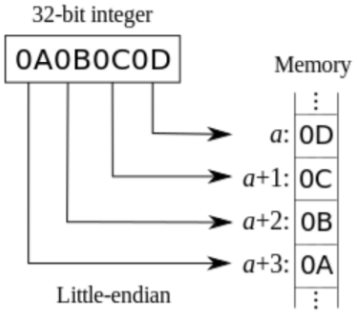
Ogni misurazione inviata dai sensori è formattata secondo le caratteristiche tecniche fornite dai produttori. Il prototipo realizzato si concentra sulla traduzione dei payload inviati dai sensori dei produttori *SensorData* e *Libellium*.

6.2 SensorData

Figura 6.1 SensorData DigitalMatter

Partendo dai sensori *SensorData* prodotti da *Digital Matter*, illustreremo come sono configurati nel dettaglio i payload inviati dai dispositivi.
Il payload è formato da una stringa esadecimale binaria i cui dati vengono passati attraverso un modello chiave-valore: ad un byte viene assegnato una chiave, la quale serve ad identificare la tipologia di misurazione che rappresenta, mentre i suoi n byte successivi (n variabile a seconda del tipo di dato) rappresentano il valore della misurazione. Viene evitato l'invio di dati correlati tra loro su più di un singolo messaggio a causa della complessità che avrebbe il riassemblaggio lato server e delle limitazioni dovute al tipo di trasmissione. Valori correlati sono sempre contenuti in un unico messaggio mentre più valori non correlati tra loro possono apparire in un singolo messaggio. I dati iniziano dal byte 0 del payload, perciò non è presente un header, mentre la chiave della prima misurazione è fornita dal numero di porta su cui viene inoltrato il messaggio (questo stratagemma aiuta a ridurre al minimo la dimensione del payload). La dimensione del payload varia a seconda del protocollo comunicativo utilizzato: nel caso si utilizzasse la tecnologia SigFox, il payload avrebbe una lunghezza predefinita di 12 byte, mentre se si utilizzasse LoRaWAN, la lunghezza del payload varierebbe in base alla regione e allo "spreading factor" utilizzato, con lunghezza minima 11 byte nel caso in cui si lavorasse nella regione EU868 con SF impostato a

12. I byte che vanno a comporre i valori delle misurazioni seguono l'ordine "Little-Endians" e quindi dovranno essere riconvertiti di conseguenza. Infine viene specificato che tutti i numeri con segno (quindi i numeri che possono essere sia positivi che negativi) sono rappresentati in complemento a 2.



Esempio di ordine Little-Endians.

Sintassi payload

Dopo le premesse sulla struttura del payload il costruttore fornisce le specifiche su come leggere i dati inviati contenuti in esso.
Ogni dato che il sensore misura e/o trasmette è denominato "campo dati". Ciascun campo dati è identificato da un ID (noto anche come "chiave") e contiene una misurazione (valore). Il software utilizzato per decodificare le informazioni ha bisogno di conoscere gli ID che gli vengono inviati e quali sono le loro lunghezze corrispondenti. Per questo motivo la casa produttrice mette a disposizione una tabella contentente gli ID che possono essere inviati dai sensori, il loro "campo dati" (a quale dato si riferiscono), la dimensione dei valori inviati ed il tipo di dato (ad esempio, UINT8, BYTE, etc). Nella tabella in figura 6.1 si possono notare alcuni "campo dati" su cui vale la pena fare un'analisi più dettagliata poiché importanti per il sistema che si vuole andare a creare: i campi che riguardano la posizione satellitare del dispositivo e i campi relativi alle misurazioni effettuate da quest'ultimo.

- **10 GPS Position**: La posizione GPS è associata all'ID identificativo di valore 10, i dati associati hanno lunghezza 6 byte.
  Nelle specifiche viene precisato come viene comunicata la posizione del dispositivo.

Offset	Type	Name	Units
0	INT24	Latitude	Deg x 10^7 / 256
3	INT24	Longitude	Deg x 10^7 / 256

ID	Name	Size	Units
0	Reserved		
1	System Firmware version (reset message)	4	Struct
2	Debug Statistics	TBD	Struct
3	Reserved for Downlink Acknowledgement		
10	GPS Position	6	Struct
20	Battery Voltage	2	UINT16 (mV)
21	Analog In 1	2	UINT16 (mV)
22	Analog In 2	2	UINT16 (mV)
23	Analog In 3	2	UINT16 (mV)
30	Digital Input State	1	Bitfield
31	Input 1 Pulse Count	2	UINT16
32	Input 2 Pulse Count	2	UINT16
33	Input 3 Pulse Count	2	UINT16
39	Digital Input Alert	2 + 2*N	Struct
40	Internal Temperature	2	INT16 (x100 degC)
41	Digital Matter I2C Temperature Probe 1 (Red)	2	INT16 (x100 degC)
42	Digital Matter I2C Temperature Probe 2 (Blue)	2	INT16 (x100 degC)
43	Digital Matter I2C Temperature & Relative Humidity	3	Struct
50	Battery Energy Used Since Power Up. 65535 = Unknown	2	UINT16 (mAh)
51	Estimated Battery % Remaining. 255 = Unknown	1	BYTE (0.5%)
128	SDI-12 Measurement 1	N	Struct
129	SDI-12 Measurement 1 – Part 2	N	Struct
130	SDI-12 Measurement 2	N	Struct
131	SDI-12 Measurement 2 – Part 2	N	Struct
132	SDI-12 Measurement 3	N	Struct
133	SDI-12 Measurement 3 – Part 2	N	Struct
134	SDI-12 Measurement 4	N	Struct
135	SDI-12 Measurement 4 – Part 2	N	Struct
136	SDI-12 Measurement 5	N	Struct
137	SDI-12 Measurement 5 – Part 2	N	Struct
223	Reserved		

Figura 6.2 Tabella dei campi dati *SensorData*

I valori associati alla misurazione sono espressi come due numeri in formato INT24, quindi i primi tre byte del campo valore rappresentano la latitudine del sensore mentre gli ultimi tre byte la longitudine. Sia latitudine sia longitudine hanno come unità di misura gradi moltiplicati per $10^7/256$ che portano ad una precisione sulla posizione di 4 metri.

- **SDI-12 Measurement n**: Le misurazioni prodotte dal sensore sono contrassegnate nella tabella dal nome "SDI-12 Measurement" seguito da un numero compreso da 1 a 5. Ogni sensore per limiti di fabbrica può inviare fino a 5 diversi tipi di rilevazione. Ogni rilevazione può essere composta da due parti nel caso in cui i valori ad essa associata superino il limite di byte del frame con cui si comunica (ad esempio, come spiegato in precedenza, nell'ipotesi peggiore il frame del protocollo LoRaWAN è composto da soli 11 byte). La dimensione in byte di una misurazione è determinata dalla misurazione stessa (quanti valori devono essere inviati) e dalla tipologia di dato associato al valore.
  Nella tabella fornita dal produttore viene spiegato come deve essere letto il "campo dati" relativo alle misurazioni.

Offset	Type	Name	Units
0.0-0.3	UINT4	Number of data points	
0.4-0.7	UINT4	Data type for this measurement 0 = soil moisture UINT8 1 = temperature UINT8 2 = INT16 (x100) 3 = INT32 (x1000) 4 = INT12	The data type determines the data size, scale and offset to use (see below)
1	-	Data for the measurement Length depends on the data type and the number of data points	

I primi 4 bit del primo byte comunicano il numero di valori di quella misurazione (massimo 10 valori per ogni misurazione) mentre gli ultimi 4 bit comunicano il tipo di dato dei valori seguendo la codifica della tabella. I valori contenuti dal secondo byte in poi corrispondono ai valori effettivi delle misurazioni.
Le cinque tipologie di dati dei valori sono:

  - **Soil moisture UINT8**: "Unsigned integer" ad 8 bit è la tipologia di dato più piccola e viene utilizzata per poter "impacchettare" più informazioni possibili in un solo frame. Nel caso in cui si utilizzi questa tipologia di dato il numero massimo di valori per frame è 10.

Questa tipologia di dato è stata pensata per le misurazioni del suolo, ma non si limita solo ad esse. Il valore contenuto nel campo deve essere letto nel seguente modo:

Valore=(Valore nel campo/2)-5

avendo quindi un range che va da -5 a 122.5 con precisione 0.5.

- **SDI-12 Temperature UINT8**: anche in questo caso vale lo stesso discorso fatto in precedenza per gli "unsigned integer" ad 8 bit. Questa tipologia di dato è stata pensata per la misurazione della temperatura ed il valore nel suo campo deve essere letto nel seguente modo:

  Valore=(Valore nel campo/2)-40

  che porta il range in cui possono essere contenuti i valori da -40 a 87.5 con 0.5 di precisione.

- **SDI-12 Generic INT16 * 100**: Questa tipologia di dato utilizza un "signed integer" a 16 bit. Un frame può contenere al massimo 5 valori che utilizzano questa dato.
  Il valore contenuto nel campo deve essere moltiplicato per 100 per ottenere il valore reale inviato dal sensore. Il range per questa tipologia di dato varia da -327.67 a 327.67 con precisione 0.1.

- **SDI-12 Generic INT32 * 1000**: Questa tipologia di dato utilizza un "signed integer" a 32 bit e un singolo frame può contenere al massimo 2 valori con questa tipologia.
  Il valore contenuto nel campo deve essere moltiplicato per 1000 prima di essere letto e il suo range varia da -2,147,483.647 a 2,147,483.647 con precisione 0.001.

- **SDI-12 Generic INT12**: Questa tipologia di dato utilizza un "unsigned integer" a 12 bit. Un frame può contenere al massimo 6 valori che utilizzano questo dato.
  Il valore reale è ottenuto con la formula:

  Valore=(valore nel campo/20)-50

  con range che varia da -50 a 154.7 con precisione 0.05.
  poiché 12 bit occupano un byte e mezzo, ogni valore sfocia nel byte successivo e deve essere letto secondo lo schema seguente che mostra come un valore possa iniziare all'inizio di un byte e finire a metà del byte successivo o rispettivamente iniziare a metà di un byte e finire alla fine del byte successivo

Byte 1	Byte 2		Byte 3
Sample 1 MSB	Sample 1 LSB	Sample 2 MSB	Sample 2 LSB

mentre la tabella sottostante mette in correlazione il numero di valori con il numero di byte utilizzati

Number of Samples	1	2	3	4	5	6
Length (bytes)	2	3	5	6	8	9

È importante notare che ogni altra misurazione prodotta dal sensore (temperatura interna, livello di carica della batteria, etc) si comporta in modo similare alle misurazione esterne del sensore e vengono trattate dal sistema allo stesso modo, cioè con osservazioni che vengono collegate ad un `DATASTREAM`.

Requisiti aggiunti

Oltre ai quelli elencati nel modello generale, il connettore *SensorData* necessita di ulteriori requisiti:

- Il payload ricevuto dal connettore deve contenere l'ID identificativo del dispositivo da cui è stato inviato il messaggio.
- Il payload inviato da un gateway LoRaWAN deve incorporare il numero di porta con cui il gateway ha ricevuto il messaggio, poiché identifica l'ID della prima misurazione.
- le entità `DATASTREAM` riguardanti le misurazioni esterne devono essere precaricate nel sistema OGC SensorThings poiché il connettore non ha dati a sufficienza per poterle creare.

Esempio

In seguito illustreremo un esempio con il quale viene spiegato in modo dettagliato il comportamento di un connettore *SensorData* all'arrivo di un payload. Dopo aver effettuato alcune misurazioni, il sensore trasmette il valore di esse attraverso un payload contenente la stringa esadecimale

"AF4D3276CE5F3334801260618231006A18"

al gateway con cui comunica. Il gateway LoRaWAN riceve il payload attraverso la porta 10 ed a sua volta lo inoltra al brocker del sistema in un messaggio che incorpora l'id univoco che identifica il sensore, in questo caso posto per esempio a "001". Il brocker, ricevuto il pacchetto, lo pubblica nel rispettivo topic *SensorData* dove il connettore associato ai dispositivi *SensorData* è in ascolto.

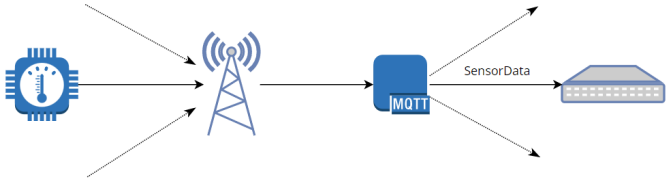


Figura 6.3

Questo è il messaggio ricevuto dal connettore:

```
{
      ID: 001,
      PORT: 10,
      PAYLOAD:AF4D3276CE5F3334801260618231006A18
}
```

La prima operazione che compie il connettore è estrarre dal suo database interno l'entità `THING` presente nel sistema SensorThing del server centrale correlata all'ID ricevuto.

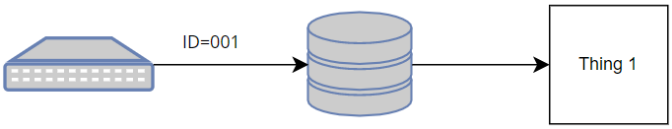


Figura 6.4

L'entità `THING` associata all'ID è la seguente
Thing

```
 {
      Name:  "Sens-IA12",
      Description:  "Sensore SensorData 1"
}
```

Il connettore ora conosce quale dispositivo ha inviato il messaggio e può procedere ad aggiornare i dati presenti nel server in base alle informazioni ricevute.
Come da specifiche, il numero della porta LoRaWAN coincide con l'ID del primo "campo dati" inviato, in questo caso 10 corrisponde al "campo dati" GPS, mentre il payload inizia dal byte 0 con i valori correlati al "campo dati" relativo alla porta.
Il connettore procede ad analizzare il payload ricevuto secondo le specifiche illustrate in precedenza:

- **AF4D3276CE5F**: dal numero della porta si intuisce che i primi 6 byte del payload contengono il valore della posizione del dispositivo.
  I primi 3 byte(AF4D32) corrispondono alla latitudine del dispositivo mentre gli utlimi 3 byte(76CE5F) corrisponodno alla longitudine, perciò in questo caso, dopo le opportune conversioni, si ottengono i valori delle coordinate *Latitudine=84.3951872 Longitudine=160.736614*.
  Con questi valori il connettore crea una nuova entità `LOCATION` da associare al dispositivo nel sistema.
  Location

```
{
    encodingType:application/vnd.geo+json,
    location:  {
        type:Feature,
        geometry:{
            type:  Point,
            coordinates:  [84.3951872,160.736614]
        }
    }
}
```

- **3334**: dal byte "33" si ricava l'ID 51 relativo all'indicatore della batteria residua mentre dal byte "34" si ricava il valore misurato, cioè *52*. Il connettore cerca tra i `DATASTREAM` collegati all'entità `THING` del dispositivo quello relativo alla misurazione della batteria rimanente.

Datastream

```
 {
      name:  "Battery remain",
      description:  "Remaining battery of the device",
      unitOfMeasurement:
      {
            name:"Percentage",
            symbol:"%",
            definition:  ""
      },
      observationType:"OM_Observation",
}
```

Se il DATASTREAM non è presente nel sistema OGC SensorThings, il connettore estrae le proprietà e le entità ad esso collegate dal suo database interno e ne crea uno nuovo.

Name	Description	UM	ObsProperty	Sensor
Battery remain	Remaining battery of the device	Percentage	Battery remain	SensorData
Soil temperature	Temperature 20cm under the soil	Celsius	Soil temperature	SDI-12
Nitrogen concentration	Nitrogen concentration in the soil, mg on K	Milligrams to kilogram	Nitrogen concentration	SDI-12
...	...	...	...	...

Tabella 6.1 Estratto della tabella contenente le informazioni riguardo alle entità DATASTREAM

Name	Symbol	Definition
Percentage	%	""
Celsius	°	HTTPs://en.wikipedia.org/wiki/Celsius
Milligrams to kilogram	mg/Kg	HTTPs://en.wikipedia.org/wiki/Kilogram
...	...	...

Tabella 6.2 Estratto di tabella delle unità di misura

La stessa operazione è effettuata con le entità SENSOR e OBSERVED PROPERTY in caso di loro assenza nel sistema centrale.

Observed Property

```
 {
      name:"Battery remain",
      definition:"HTTPs://en.wikipedia.org/wiki/Electric_battery
      description:  "The remaining energy level of an electric
battery"
}
```

Name	Definition	Description
Battery remain	HTTPs://en.wikipedia.org/wiki/Electric_battery	The remaining energy level of an electric battery
Soil temperature	HTTPs://en.wikipedia.org/wiki/Temperature	Soil temperature at a depth of 20cm from the surface
Nitrogen concentration	HTTPs://en.wikipedia.org/wiki/Concentration	Concentration of nitrogen in the soil calculated on one cubic meter
...	...	...

Tabella 6.3 Estratto della tabella delle entità OBSERVED PROPERTY

Sensor

```
 {
      name:"SensorData",
      description:  "SensorData LoRaWAN is a battery or line-
powered data communicator that interfaces to a range of sen-
sors, GPS, inputs and outputs, and uploads data via LoRaWAN
networks"
      encoding type:  "application/pdf",
      metadata:"HTTPs://digmat.freshdesk.com/helpdesk/attachments
      /16073699325"
}
```

Name	Description	EncodingType	Metadata
SDI-12	An asynchronous serial communications protocol for intelligent sensors that monitor environment data	application/pdf	HTTPs://sdi-12.org/current_specification/SDI-12_version-1_4-Jan-30-2021.pdf
SensorData	SensorData LoRaWAN is a battery or line-powered data communicator that interfaces to a range of sensors, GPS, inputs and outputs, and uploads data via LoRaWAN networks	application/pdf	HTTPs://digmat.freshdesk.com/helpdesk/attachments/16073699325
...	...	...	...

Tabella 6.4 Estratto della tabella delle entità SENSOR

Infine pubblica la misurazione collegandole l'entità FEATURE OF INTEREST relativa.

Observation

```
{
    result:  52
}
```

Feature Of Interest

```
{
    name:  "Battery",
    description:  "Device battery",
    encodingType:"application/vnd.geo+json",
    feature:""
}
```

In modo analogo ai precedenti esiste una tabella all'interno del connettore contente le informazioni riguardanti le entità "FEATURE OF INTEREST".

Datastream	ThingID	FeaureOfInterest
Battery remain	1	Battery
Soil temperature	1	Soil
Nitrogen concentration	1	Nitrogen Concentration
...	...	

Tabella 6.5 Tabella che mette in relazione le entità DATASTREAM alle entità FEATURE OF INTEREST

Si noti come il database interno metta in relazione le FEATURE OF INTEREST con i DATASTREAM e l'ID univoco che identifica il dispositivo: mentre le entità precedenti(OBSERVED PROPERTY e SENSOR) sono trasversali a tutti i dispositivi dello stesso mdello, le FEATURE OF INTEREST sono specifiche per ogni singolo dispositivo, poiché le misurazioni possono essere effettuate su spazi con dimensioni e/o posizioni differenti. Nel caso in cui non sia presente una FEATURE OF INTEREST da collegare all'osservazione, il server centrale ne deriverà una dalle entità LOCATION con la quale essere collegata.
I passaggi visti finora riguardanti la creazione delle entità OBSERVED PROPERTY, SENSOR e FEATURE OF INTEREST saranno omesse nei prossimi esempi per rendere più chiara la spiegazione.

- **80126061**:Il primo byte ha valore decimale 128 che corrisponde al "campo dati" SDI-12 Measurement 1. Il primo valore del secondo byte (1) corrisponde alla tipologia di dato con cui vengono espressi i valori, in questo caso "Temperature UINT8", mentre il secondo valore (2) il numero di osservazioni, in questo caso 2. Gli ultimi 2 byte corrispondono ai valori delle misurazioni *misurazione1= 8* e *misurazione2= 8.5* corrispondenti a due misurazioni della temperatura del suolo.
  Il connettore non ha abbastanza informazioni sulla natura delle misurazioni di tipo "SDI-12 Measurement" per poter creare un nuovo `DATASTREAM` relativo ad esse, perciò può solo cercare tra i `DATASTREAM` associati se ne esiste uno caricato in precedenza. Una volta ottenuto il `DATASTREAM`
  Datastream

```
{
     name:  "Soil temperature",
     description:  "Temperature 20cm under the soil",
     unitOfMeasurement:
     {
          name:"Gradi Celsius",
          symbol:"°C",
          definition:  ""
     },
     observationType:"OM_Measurement",
}
```

  il connettore pubblica due nuove `OBSERVATION`, una per misurazione ricevuta
  Observation

```
{
     result:  8
}
{
     result:  8.5
}
```

- **8231006A18**:in modo analogo al "campo dati" precedente, si ricava l'ID dal primo byte(82) che equivale a 130, SDI-12 Measurement 2, mentre dal secondo byte(31) si ottiene la misurazione di tipo INT32 composta da un'unico valore. Dopo la conversione il valore che si ottiene è *1600*, corrispondente alla misurazione del livello di azoto presente nel suolo.

Il connettore cerca il `DATASTREAM` relativo
Datastream

```
 {
      name:  "Nitrogen concentration",
      description:  "Nitrogen concentration in the soil, mg on
Kg",
      unitOfMeasuremen":
      {
            name:"milligrams to kilogram",
            symbol:"mg/Kg",
            definition:  ""
      },
      observationType:"OM_Measurement",
      resultTime:"2020-11-03T13:00:00Z/2020-11-03T13:00:00Z"
}
```

e pubblica una nuova `OBSERVATION`
Observation

```
 {
      resultTime:"2020-11-03T13:00:00Z/2020-11-03T13:00:00Z",
      result":  1600
}
```

6.3 Libellium

Figura 6.5 Dispositivo Waspmote di *Libellium*

La seconda tipologia di payload analizzati sono quelli prodotti dai dispositivi *Waspmote* dell'azienda *Libellium*.
Questi dispositivi sono programmabili dall'utente e possono produrre payload in due formati: il primo presenta il payload come stringa formata da caratteri ASCII mentre il secondo formato prensenta la stringa come serie di byte.

- **Stringa ASCII**:Questo formato di frame dovrebbe facilitare la comprensione dei dati inviati a scapito della dimensione del pacchetto. L'utilizzo dei caratteri ASCII permette al sensore di inviare dati in modo "trasparente", cioè senza doverli convertire nella tipologia di dato associata alla specifica misurazione. Il lavoro del connettore è a sua volta facilitato poiché non deve conoscere tutte le tipologie di dati di ogni misurazione, ma può estrapolare i valori direttamente semplicemente leggendo il payload ricevuto. Lo svantaggio di utilizzare questo formato è che ogni singolo carattere occupa un byte all'interno del pacchetto, perciò difficile comprimere le informazioni e i frame prodotti in questo modo avranno dimensioni maggiori.

La struttura del frame prodotto è composta da un header fisso, contenente informazioni utili alla comunicazione con il dispositivo, e da un payload variabile contente i valori delle misurazioni effettuate.

HEADER							PAYLOAD			
<=>	Frame Type	Num Fields	Serial ID	Waspmote ID	#	Sequence	Sensor_1	Sensor_2	...	Sensor_n

- **Stringa Binaria**:Questo tipo di frame è stato progettato per creare frame più compressi a scapito della leggibilità. Il frame è composto da una serie di byte ognuno rappresentante il valore di un campo preciso. L'obiettivo principale di questo formato è salvare più byte possibili nel payload del frame per inviare quante più informazioni possibili. Per fa ciò, il sensore converte ogni misurazione nella rispettiva tipologia di dato dettato dalle specifiche fornite dal produttore. Il connettore a sua volta deve essere a conoscienza di queste specifiche per poter estrarre e "tradurre" i valori ricevuti prima di inviarli al server centrale.
  Anche in questo caso la struttura del frame è composta da un header e da un payload in modo analogo al precedente.

HEADER							PAYLOAD			
<=>	Frame Type	Num Fields	Serial ID	Waspmote ID	#	Sequence	Sensor_1	Sensor_2	...	Sensor_n

Sintassi del payload

La struttura e la sintassi dei due tipi di frame sono simili: come accennato in precedenza, entrambi sono composti da due parti, un header composto da campi fissi che contengono informazioni riguardanti il dispositivo(come, ad esempio, l'ID identificativo) e alla comunicazione (n° di pacchetto), e un payload composto da campi variabili contente i valori misurati dal sensore con modalità chiave-valore.
Una delle differenze principali tra frame ASCII e frame binario è che nel primo viene utilizzato il carattere "#" per delimitare ogni campo, nel secondo invece il carattere "#" viene utilizzato solo per determinare la fine di un campo a lunghezza variabile, mentre tutti gli altri campi hanno lunghezza fissa dettata dalle specifiche tecniche e non hanno bisogno di essere delimitati.
Analizziamo per prima la struttura e le informazioni contenute nell'header. Come esempio viene mostrato un header di un frame binario nel quale è stata associata una lettera ad ogni campo contraddistinto:

HEADER						
<=>	0x00	0x03	0x74F94515	NODE_001	#	0x00
A	B	C	E	F	D	G

- **A[3 byte]**:Questo campo è composto dai tre caratteri "<=>" ed è utilizzato per determinare l'inizio di un nuovo frame.
- **B[1 byte]**:Questo campo viene utilizzato per determinare il tipo di frame, Binario o ASCII, e lo scopo del frame (frame di servizio, frame contenente dati, frame di inizio e di fine trasmissione). In particolare, se il primo bit di questo byte è pari a 0, allora il frame sarà di tipo binario, nel caso in cui sia pari a 1 allora sarà un frame di tipo ASCII.
- **C[1 byte]**:Questo campo specifica il numero di campi misurazioni contenute nel frame.
- **D[1 byte]**: carattere "#", utilizzato nei frame binari per segnare la fine di un campo variabile e nei frame ASCII come delimitatore tra un campo e l'altro.
- **E[4 byte]**:Questo è un campo di 4 byte che identifica ogni dispositivo *Waspmote* in modo univoco. L'ID seriale è ottenuto da un chip specifico integrato in Waspmote che fornisce un identificatore diverso a ciascun dispositivo. Quindi, è solo leggibile e non può essere modificato. Si noti che anche l'ID seriale viene inviato come campo binario.
- **F[0-16 byte]**: Questa è una stringa definita dall'utente che può identificare ogni *Waspmote* all'interno della rete locale. La dimensione del campo è variabile [da 0 a 16 Byte]. Quando l'utente non vuole fornire alcun identificatore, il campo rimane vuoto indicato da un carattere "#" univoco.
- **G[1 byte]**:Questo campo indica il numero di frame inviati. Questo contatore è a 8 bit, quindi va da 0 a 255. Ogni volta che raggiunge il massimo viene reimpostato a 0. Questo numero di sequenza viene utilizzato per rilevare la perdita di frame.

Infine analizziamo la struttura e le informazioni contenute nel payload.
Le misurazioni contenute nel payload sono espresse come coppie chiave-valore. Il produttore fornisce una tabella con tutte le possibili misurazioni effettuabili dai sensori e le loro proprietà come ID binario, ID ASCII,

dimensione, unità di misura, precisione, etc. Queste informazioni sono fondamentali per il connettore per conoscere come estrarre e convertire i dati da inviare successivamente al server.

Sensor	Sensor Reference	Sensor TAG	SENSOR ID		Number Of Fields	Binary		ASCII	Units
			Binary	ASCII		Type of variable	Size per Field (Bytes)	Default Decimal Precision	
Solvent Vapors	9237	SENSOR_SV	8	SV	1	float	4	3	voltage
Nitrogen Dioxide	9238	SENSOR_NO2	9	NO2	1	float	4	3	voltage
Temperature Celsius	9203	SENSOR_TCA	12	TCA	1	float	4	2	°C
Temperature Fahrenheit	9203	SENSOR_TFA	13	TFA	1	float	4	2	°F
Humidity	9204	SENSOR_HUMA	14	HUMA	1	float	4	1	%RH
Battery	N/A	SENSOR_BAT	52	BAT	1	uint8_t	1	0	%
Global Positioning System	WGPS	SENSOR_GPS	53	GPS	2	float	4	6	degrees
RSSI	N/A	SENSOR_RSSI	54	RSSI	1	int	2	0	N/A

Figura 6.6 Un estratto dalla tabella dei campi delle misurazioni

Per poter analizzare al meglio il contenuto del payload è necessario distingure tra payload binario e payload ASCII poiché presentano caratteristiche in lettura leggermente diverse.
Partendo dalla versione binaria, viene mostrato un esempio di payload contenuto in un frame.

PAYLOAD								
ID	Byte 1	Byte 2	ID	Byte 1	Byte 2	ID	Byte 1	Byte 2
Sensor 1			Sensor 2			Sensor 3		

Ogni misurazione inizia con un ID di dimensione un byte rapresentante un numero che ne identifica la tipologia. Ad ogni ID sono associati uno o più byte a seconda delle specifiche mostrate in tabella. Non è necessario utilizzare un carattere di separazione tra le diverse misurazioni poiché i campi che contengono sono di lunghezza fissa predeterminata. I campi che contengono i valori possono essere composti da tre tipi di dati:

- **Simple data**:Questo campo è composto da un dato unico costituito da un numero di byte dettato dalla tabella dei campi. Ad esempio, la misurazione della temperatura è un numero float, quindi è un campo a 4 byte. Pertanto, il valore nel campo per 27^{o}C sarà impostato come segue:

ID (1 Byte)	Byte1	Byte2	Byte3	Byte4
SENSOR_TCA	0x00	0x00	0xD8	0x41

- **Complex data**:Questo è il formato utilizzato per inviare dati composti da più di un valore.I diversi valori vengono codificati utilizzando tanti byte quanti specificati nella tabella dei campi. Ad esempio, il campo GPS è composto dai numeri float corrispondenti a latitudine e longitudine, il che significa che sono necessari 8 byte per entrambi i valori float:

ID (1 Byte)	Byte1	Byte2	Byte3	Byte4	Byte1	Byte2	Byte3	Byte4
SENSOR_GPS	0x59	0x9D	0x26	0x42	0xE0	0x10	0x61	0xBF

- **String**:Questo è l'unico campo che si forma diversamente: il primo byte successivo all'ID definisce la lunghezza della stringa, e il resto dei byte appartiene alla stringa stessa secondo la lunghezza precedentemente definita. Ad esempio, la stringa "hello" è formattata come segue:

ID (1 Byte)	Length	Byte1 ('h')	Byte2 ('e')	Byte3 ('l')	Byte4 ('l')	Byte5 ('o')
SENSOR_STR	0x05	0x68	0x65	0x6C	0x6C	0x6F

In modo analogo analizziamo il payload nel caso in cui venga utilizzato un frame con codifica ASCII.
Di seguito viene fornito il payload in formato ASCII.

PAYLOAD					
Temp:35	#	GPS:31.200;42.100	#	DATE:12-01-01	#
sensor1	D	sensor2	D	sensor3	D

Ogni misurazione è formata dall'ID in formato stringa che la identifica nella tabella dei campi seguito da una serie di caratteri che ne specificano il valore, anch'essi espressi sottoforma di stringa.
In questo caso, come si vede dall'immagine, è necessario utilizzare il carattere "#" per dividere i campi delle diverse misurazioni poiché queste non hanno lunghezza fissa come quando viene utilizzata la codifica binaria, ma hanno dimensione variabile in base a quanti caratteri contengono. Il payload ASCII utilizza le tre tipologie di dati visti nel payload binario, ma con formattazione leggermente diversa:

- **Simple data**:Il campo del sensore è composto da un dato unico. Il formato è: "tag:valore" e un carattere separatore "#" è impostato alla fine del valore. Ad esempio, un campo di temperatura che indica 23ºC sarebbe:

  #TC:23#

- **Complex data**:Questo è il formato utilizzato per inviare dati composti da due o tre valori. Il formato è: "tag:valore;valore;valore" e un carattere separatore "#" è impostato alla fine dell'ultimo valore. Accelerometro e misurazioni GPS sono alcuni esempi:

  #ACC:996;-250;-100#
  #GPS:41.680616;-0.886233#

- **Special data**:La data e l'ora sono definite in un formato speciale.
  La data è definita come "aa-mm-gg" dove:
  - aa: anno
  - mm: mese
  - gg: giorno del mese
  Esempio:

  #DATA:13-01-01#

  L'ora è formattata come "hh-mm-ss+GMT" dove:
  - hh: ore
  - mm: minuti
  - ss: secondi
  - GMT: GMT viene aggiunto dopo hh-mm-ss. È possibile evitare questa informazione per salvare spazio nel frame.

  Esempio senza GMT: #TIME:12-24-16#
  Esempio con GMT: #TIME:12-24-16+1#

Requisiti aggiunti

Oltre ai quelli elencati nel modello generale, il connettore *Libellium* necessita di ulteriori requisiti:

- L'entità `THING` nel sistema OGC SensorThings deve essere associata all'ID seriale *Waspmote* unico.
- Il dispositivo trasmette solo frame semplici, cioè tutte le informazioni sono contenute in un frame unico e non suddivise in frame multipli.

Esempio

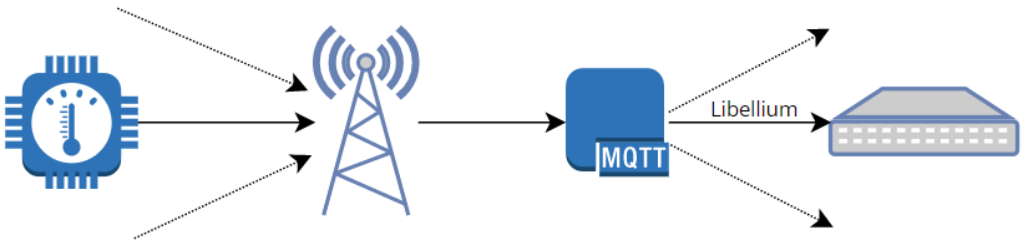
I sensori *Waspmote* di *Libellium* inviano tutte le informazioni necessarie al connettore all'interno dei propri frame e quindi, in questo caso specifico, non è necessario l'incapsulamento di informazioni aggiuntive da parte del gateway. Vengono presentati due esempi di traduzione effettuati dal connettore *Libellium*, uno per la codifica ASCII ed uno per la codifica binaria.

ASCII

Successivamente ad aver effettuato delle misurazioni, il sensore *Waspmote* trasmette il loro valore con un payload contenente la stringa esadecimale

"3C3D33800320233335363930323834202320 4E4F44455C5F30303123323134 235443413A3335234750533A33312E3230303B34322E3130234241543A3837"

inoltrandolo al connettore seguendo il percorso già visto in precedenza.



Alla ricezione, la prima operazione che il connettore effettua è il controllo della codifica del frame: il byte che trasporta le informazioni riguardanti il frame (0x80) ha il bit più significativo impostato a 1, perciò connettore interpreterà il frame come la stringa ASCII:

"<=> 0x80 0x03 # 35690284 # NODE_001 214 # TCA:35 # GPS:31.200;42.100 # BAT:87"

La seconda operazione che il connettore compie è identificare il sensore all'interno del sistema. Per far ciò, legge il campo del frame contenente l'ID unico identificativo del dispositivo, in questo caso 35690284, consulta il suo database interno ed estrae l'ID dell'entita `THING` presente nel server associata.

Thing

```
 {
     Name:  "Waspnote001",
     Description:  "Sensore Waspmote 1",
}
```

Infine passa all'analisi delle informazioni contenute nel payload.

- **TCA:35**: Questo campo corrisponde alla misurazione di una temperatura. Il connettore cerca tra i `DATASTREAM` associati al sensore quello corrispondente alla temperatura, se non presente ne crea uno nuovo.
  Datastream

  ```
   {
        name:  "Temperature Celsius",
        description:  "Temperatura",
        unitOfMeasurement:
        {
             name:"Celsius",
             symbol:"°C",
        },
        observationType:"OM_Measurement",
        resultTime :"2020-11-03T13:00:00Z/2020-11-03T13:00:00Z"
   }
  ```

  Identificato il `DATASTREAM`, crea un'entità `OBSERVATION` da collegargli contenente la misurazione.
  Observation

  ```
   {
        resultTime:"2020-11-03T13:00:00Z/2020-11-03T13:00:00Z",
        result:  35
   }
  ```

- **GPS:31.200;42.100**: in questo caso la misurazione corrisponde alla posizione GPS del dispositivo. Il primo valore corrisponde alla latitutine mentre il secondo valore alla longitudine. Il connettore crea una nuova entità `LOCATION` da associare all'entità `THING` del sensore.

Location

```
 {
      encodingType:"application/vnd.geo+json",
      location:  {
            type:Feature,
            geometry:{
            type:  "Point",
            coordinates:  [31.200,42.100]
      }
}
}
```

- **BAT:87**: Il dispositivo comunica la percentuale di batteria rimasta e il connettore tratta questa misurazione come una qualsiasi altra misurazione prodotta dal sensore, perciò cerca il `DATASTREAM` associato e in caso non lo trovasse ne crea uno nuovo.
  Datastream

```
 {
      name:  "Battery",
      description:  "Batteria residua del dispositivo.",
unitOfMeasurement:
      {
      name:"Percentuale",
            symbol:"%",
            definition:  ""
      },
      observationType:"OM_Observation",
      resultTime:"2020-11-03T13:00:00Z/2020-11-03T13:00:00Z"
}
```

  In seguito crea l'entità `OBSERVATION` relativa
  Observation

```
 {
      resultTime:"2020-11-03T13:00:00Z/2020-11-03T13:00:00Z",
      result:  87
}
```

Binario

In modo analogo ai casi precedenti, il sensore trasmette al connettore il payload

“3C3D3E 00 03 74F94515 53 2624 180000DA41 0FD7A3CA42”

Il secondo campo del pyload(0x00) contiene solo zeri quindi il connettore interpreterà il frame come binario mentre il quarto campo contiene l’ID unico del dispositivo con il quale si estrae dal database l’entità `THING` associata con ID pari a 003.
Thing

```
 {
     Name:  "Waspnote002",
     Description:  "Sensore Waspmote 2",
}
```

I successivi campi contengono i valori delle misurazioni.

- **2624**:Il primo byte corrisponde alla misurazione “Leaf Wetness” che ha come valore un “unsigned integer” pari a *36*.
  Datastream

  ```
   {
        name :  "Leaf Wetness",
        description :  "Umidità delle foglie",
        unitOfMeasurement :
        {
             name :"Percentuale",
             symbol :"%",
        },
        observationType :OM_CountObservationn,
        resultTime :"2020-11-03T13:00:00Z/2020-11-03T13:00:00Z"
  }
  ```

  Observation

  ```
   {
        resultTime :"2020-11-03T13:00:00Z/2020-11-03T13:00:00Z",
        result :  36
  }
  ```

- **180000DA41**:Il primo byte corrisponde alla misurazione “Temperature Celsius” che ha come valore un float, quindi il risultato è contenuto nei successivi 4 byte e va letto seguendo l’ordine “Little-Endians”.

Datastream

```
 {
      name :  "Temperature Celsius",
      description :  "Temperatura in gradi Celsius",
      unitOfMeasurement :
      {
            name :"Gradi Celsius",
            symbol :"°C",
      },
      observationType :OM_Measurement,
}
```

L'osservazione ha valore *27,25*

```
 {
      result :  27.25
}
```

- **0FD7A3CA42**:L'ultima misurazione corrisponde a "Pressure atmospheric" e anch'essa ha come valore un float
Datastream

```
 {
      name :  "Pressure atmospherics",
      description :  "Pression atmosfericas",
      unitOfMeasurement :
      {
            name :"Kilo Pascal",
            symbol :"kPA",
      },
      observationType :OM_Measurement,
}
```

di valore *101,32*

```
 {
      result:  101.32
}
```

7 | Conclusione

Grazie all'introduzione del concetto di connettore, l'architettura proposta si candida come soluzione al problema di interoperabilità tra dispositivi IoT. Questa soluzione non è l'unica possibile, ma presenta numerosi vantaggi rispetto ad altre alternative che la rendono facilmente implementabile ed adatta al settore dell'agricoltura di precisione.
Innanzitutto le risorse richieste da un connettore sono minime: le operazioni compiute non richiedono grandi capacità computazionali e la memoria utilizzata è molto limitata. Un connettore può essere quindi facilmente implementabile con un dispositivo embedded dotato di connessione a internet, che lo rende poco dispendioso da un punto di vista economico.
Ogni connettore collegato alla rete può essere considerato un modulo del sistema rendendo quest'ultimo adattabile a vari tipi di esigenze. Una rete ospiterà al suo interno solo moduli utilizzabili riducendo il numero di funzionalità non necessarie al funzionamento che potrebbero introdurre complessità e incompatibilità all'interno del sistema. Inoltre lavorando a moduli, nel caso in cui un connettore si guasti o il suo software debba essere aggiornato per rispettare le nuove linee guida fornite dai produttori, è sufficiente lavorare sul singolo modulo andandolo a sostituire o modificare senza dover riconfigurare l'intero sistema.
La modularità del sistema rende l'architettura adatta sia per contesti locali sia su larga scala e facilita la scalabilità di quest'ultima. Ogni connettore è un'unità indipendente dalle altre e può essere aggiunto o rimosso dalla rete senza intaccare l'integrità di essa. In questo modo è possibile adattare le dimensioni del sistema in base alle esigenze del contesto senza interruzioni al flusso di dati al suo interno.
La combinazione tra dispositivo connettore e standard utilizzato rende l'approccio al sistema semplice ed intuitivo. L'utente finale può concentrarsi sull'analisi dei dati raccolti senza che sia necessaria una comprensione approfondita dell'architettura con cui lavora. Lo sviluppo di applicativi terzi con cui interfacciarsi al server di raccolta dei dati richiede solamente la conoscenza delle specifiche dello standard, il quale è stato appositamente scelto

per la sua intuitività per rendere l'esperienza degli utenti finali la più lineare possibile.

Lavori futuri

In questa tesi lo sviluppo e l'analisi del software per i dispositivi connettori si è limitato ad alcune specifiche tipologie di sensori prodotti dalle aziende *Libellium* e *Digital Matter*. Per aggiungere il supporto ad ulteriori dispositivi provenienti da case di produzione diverse è necessario lo studio delle specifiche tecniche con cui essi comunicano i propri dati. Ogni implementazione di una nuova tipologia di connettore può essere sviluppata al bisogno ed aggiunta al sistema quando completata grazie alla modularità dell'architettura. L'obiettivo è sviluppare il maggior numero di tipologie di connettori in modo da poter supportare la maggior parte dei sensori più utilizzati nell'ambito dell'agricoltura offerti dal mercato.
Al momento l'architettura supporta solamente la ricezione dei dati dai dispositivi e non l'invio di messaggi ad essi. Per rendere il sistema completo, in futuro può essere implementato il supporto alla comunicazione verso i dispositivi. In particolare nel mondo IoT esistono dispositivi chiamati "attuatori" che necessitano di essere comandati da remoto per compiere diverse azioni nel mondo reale. La seconda parte di cui è composto lo standard OGC SensorThings propone un metodo di comunicazione con cui interfacciarsi con questi attuatori. Sarà compito del connettore quindi ricevere le informazioni inviate dagli utenti e "tradurle" nel linguaggio del dispositivo, sostanzialmente effettuando l'operazione inversa rispetto a quella esposta in questo elaborato.

Bibliografia

[1] Sunil Cheruvu, Anil Kumar, Ned Smith, David M. Wheeler, *Demystifying Internet of Things Security. Successful IoT Device/Edge and Platform Security Deployment.*, Apress Open, 2020

[2] Ioannis N. Athanasiadis, Steven P. Frysinger, Gerald Schimak, Willem Jan Knibbe, *Environmental Software Systems. Data Science in Action.*, Springer, 2020

[3] Milan Milenkovic, *Internet of Things: Concepts and System Design*, Springer, 2020

[4] M. Ayaz, M. Ammad-Uddin, Z. Sharif, A. Mansour and E. -H. M. Aggoune, *Internet-of-Things (IoT)-Based Smart Agriculture: Toward Making the Fields Talk*, in IEEE Access, vol. 7, pp. 129551-129583, 2019, doi: 10.1109/ACCESS.2019.2932609.

[8] D. Marsh-Hunn, S. Trilles, A. González-Pérez, J. Torres-Sospedra and F. Ramos, *A Comparative Study in the Standardization of IoT Devices Using Geospatial Web Standards*, in IEEE Sensors Journal, vol. 21, no. 4, pp. 5512-5528, 15 Feb.15, 2021, doi: 10.1109/JSEN.2020.3031315.

[8] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, *Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications*, in IEEE Communications Surveys & Tutorials, vol. 17, no. 4, pp. 2347-2376, Fourthquarter 2015, doi: 10.1109/COMST.2015.2444095.

[7] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui and T. Watteyne, *Understanding the Limits of LoRaWAN*, in IEEE Communications Magazine, vol. 55, no. 9, pp. 34-40, Sept. 2017, doi: 10.1109/MCOM.2017.1600613.

[4] G. Kortuem, F. Kawsar, V. Sundramoorthy and D. Fitton, *Smart objects as building blocks for the Internet of things*, in IEEE Internet Computing, vol. 14, no. 1, pp. 44-51, Jan.-Feb. 2010, doi: 10.1109/MIC.2009.143.

[9] P.P. Ray, *A survey on Internet of Things architectures*, Journal of King Saud University - Computer and Information Sciences, vol. 30, no. 3, pp 291-319, 2018, doi: 10.1016/j.jksuci.2016.10.003.

[10] M. S. Farooq, S. Riaz, A. Abid, K. Abid and M. A. Naeem, *A Survey on the Role of IoT in Agriculture for the Implementation of Smart Farming*, in IEEE Access, vol. 7, pp. 156237-156271, 2019, doi: 10.1109/ACCESS.2019.2949703.

[11] Liang, Steve & Huang, Chih-Yuan & Khalafbeigi, Tania. (2016). OGC SensorThings API Part I:Sensing.

[12] A. Kotsev et al., *Extending INSPIRE to the Internet of Things through SensorThings API*, Geosciences, vol. 8, no. 6, p. 221, Jun. 2018, doi: 10.3390/geosciences8060221.

[13] M. A. Ertürk, M. A. Aydın, M. T. Büyükakkaşlar, and H. Evirgen, *A Survey on LoRaWAN Architecture, Protocol and Technologies*, Future Internet, vol. 11, no. 10, p. 216, Oct. 2019, doi: 10.3390/fi11100216.

[14] V. Sundareswaran, *A survey on smart sensors in precision agricolture*, 2018

[15] Jararweh, Y., Al-Ayyoub, M., Darabseh, *SDIoT: a software defined based internet of things framework.*, J Ambient Intell Human Comput, vol. 6, pp 453–461, 2015, doi: 10.1007/s12652-015-0290-y.

[16] B. Moons, E. De Poorter, and J. Hoebeke, *Device Discovery and Context Registration in Static Context Header Compression Networks*, Information, vol. 12, no. 2, p. 83, Feb. 2021, doi: 10.3390/info12020083.

[17] C. Capodiferro, M. Mazzei, *An approach adopted for smart data generation and visualization problems*, ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, 2020, doi: 10.5194/isprs-annals-VI-4-W2-2020-41-2020.

[18] ISO/TC211,*ISO 19156: 2011-Geographic Information–Observations and Measurements-International Standard* International Organization for Standardization, 2011, Svizzera.

Siti consultati

[1] ISO/IEC 20922:2016, *Information Technology—Message Queuing Telemetry Transport (MQTT) v3.1.1*, https://www.iso.org/standard/69466.html.

[2] ISO/IEC 20802-1:2016, *Open Data Protocol (OData)*, https://www.iso.org/standard/69208.html.

[3] Project Haystac, *Haystack Documentation*, https://project-haystack.org/doc/docHaystack/index.

[4] ISO/IEC 30118-1:2021, *Information technology - Open Connectivity Foundation (OCF) Specification - Part 1: Core specification*, https://www.iso.org/standard/82127.html.

[5] OMA SpecWorks, *OMA LightweightM2M (LwM2M) Object and Resource Registry*, https://technical.openmobilealliance.org/OMNA/LwM2M /LwM2MRegistry.html

[6] OGC, *SensorThings API Part 1: Sensing*, http://docs.opengeospatial.org/is/15-078r6/15-078r6.html

[7] *OpenAPI Specification*, https://github.com/OAI/OpenAPI-Specification.

[8] *FROST-Server*, https://github.com/FraunhoferIOSB/FROST-Server

[9] Libellium, *Frame structure*, https://development.libelium.com /data-frame-programming-guide/frame-structure

[10] Digital Matter, *SensorData integration*, https://support.digitalmatter.com/helpdesk/attachments /16081161507