ART4SOS - TERMINALE REMOTO PER IL SISTEMA DI OSSERVAZIONE DEI SENSORI

L. La Gattuta, A. Langiu, P. Storniolo, M. Sprovieri

Indice

1 Introduzione **2**

2 Prerequisiti **3**

2.1 Sensor Observation Service 3

2.1.1 Panoramica 3

2.1.2 Modello di dati 4

2.1.3 Inserimento dati 5

2.2 Raspberry PI 7

2.2.1 Panoramica del sistema e descrizione tecnica 7

2.2.2 Sistema operativo e capacità di programmazione 9

2.2.3 Connettività del sensore 10

3 Sistema di acquisizione dati da sensori **12**

3.1 Software Solution 14

3.2 Configurazione 15

3.2.1 Acquisizione dei dati 16

3.2.2 Formato dei dati 18

3.2.3 Trasmissione dati 18

3.3 Funzionalità secondarie 20

3.4 Installazione 21

3.5 Configurazione 22

3.6 Analisi della Soluzione 24

4 Conclusioni **27**

Capitolo 1

Introduzione

Nell'era dell'Internet degli oggetti (IoT), dove ogni giorno si diffonde la disponibilità della connettività dati e dei dispositivi intelligenti, la capacità di acquisire i dati dei sensori attraverso una soluzione semplice, economica ed efficiente in grado di utilizzare lo standard affermato per l'osservazione dei sensori, è un fattore di promozione per la creazione collaborativa di una serie di dati secondo la filosofia Open Data. Questa impostazione è particolarmente adatta al monitoraggio ambientale attraverso la misurazione ripetitiva di parametri fisici e chimici. Questo rapporto tecnico presenta una nuova soluzione software per l'acquisizione dei dati basata su un terminale Raspberry PI[1] dotato di un sensore termico e connettività dati a livello di sistema. Sfrutta le funzionalità standard del Sensor Observation Service (SOS)[2] come il modello di dati, i formati di comunicazione e assume la disponibilità di un server SOS per archiviare i dati, e un implementazione open source da 52° North[3] come servizio remoto.

[1] Sito ufficiale di Piattaforma Raspberry PI - https://www.raspberrypi.org/

[2] OGC ® Standard di interfaccia del Sensor Observation Service. Open Geospatial Consortium, 20 aprile 2012, versione: 2.0 - https://www.opengeospatial.org/standards/sos

[3] 52° North SOS Server web-site - https://52north.org/software/

Capitolo 2

Prerequisiti

2.1 Sensor Observation Service

2.1.1 Panoramica

Il SOS (Sensor Observation System) è uno standard che indirizza ed estende lo scenario comune di acquisizione dei dati dai sensori, come stazioni meteorologiche, sonde chimiche e sensori dei parametri fisici, fornendo un modello di dati, una serie di funzionalità per l'inserimento, l'interrogazione e il recupero di informazioni che un dato in origine deve fornire, così come le specifiche di un protocollo di comunicazione ricco e completo.

SOS è applicabile nel caso in cui i dati dei sensori devono essere gestiti in modo interoperativo. Lo standard definisce un'interfaccia del servizio Web che consente di inserire e interrogare osservazioni, metadati del sensore e rappresentazioni delle caratteristiche osservate. Inoltre, questo standard specifica come registrare un nuovo sensore e come rimuovere quelli esistenti.

Molte delle funzionalità di cui sopra sono definite in un modo indipendente dal formato dei dati; nell'ambito di questo lavoro ci concentriamo su due formati di dati, il JavaScript Object Notation[1] (JSON) e il Simple Object Access Protocol[2] (SOAP) per lo scambio di dati strutturati.

[1] Sito Web ufficiale Notazione oggetti JavaScript - https://www.json.org/

[2] SOAP (Simple Object Access Protocol) 1.1, W3C Note, 08 maggio 2000

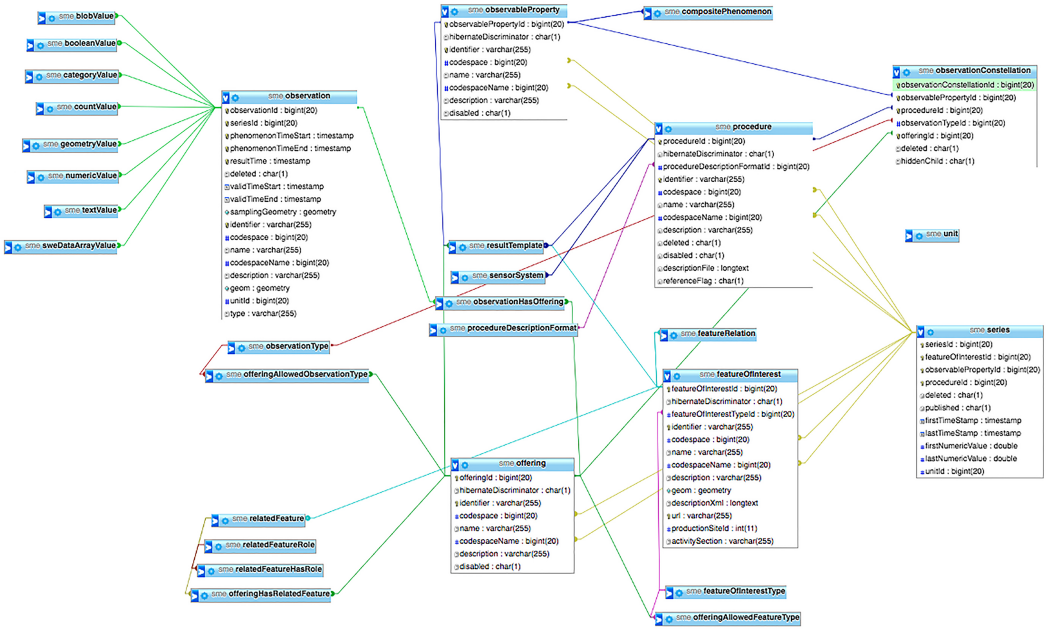

Figura 2.1: Schema dei componenti principali del modello di dati SOS.

2.1.2 Modello di dati

Il modello di dati SOS si basa sul concetto di *osservazione*. Un'osservazione è l'atto di osservare un *fenomeno*, che potrebbe essere di natura fisica o potrebbe essere un'operazione virtuale, derivata o complessa. L'atto base dell'osservazione è la misurazione di un parametro. Diversi tipi di osservazioni vengono eseguite mediante una appropriata *procedura*. L'atto di misurare un fenomeno tramite una sonda fisica produce un valore *numerico*. La *caratteristica di interesse* è la presenza di un fenomeno in un luogo reale. La ripetizione nel tempo della lettura sensoriale riferita allo stesso fenomeno, luogo di interesse e procedura, unitamente all'appropriata *unità di misura*, sono collegate per formare delle *serie temporali*.

Nella figura 2.1 è riportato uno schema dei componenti principali del modello di dati SOS che si riflette nell'implementazione del database. Per una descrizione più dettagliata dello standard SOS si faccia riferimento alle documentazioni ufficiali SOS.

Il protocollo di comunicazione definisce una serie di funzionalità che devono essere supportate da un'origine dati compatibile con il formato di query, risposta e dati basato

principalmente su linguaggi derivati XML (eXtensible Markup Language)[3]. Le principali operazioni sono:

- GetCapabilities - identifica i dati, i relativi sensori e i parametri associati ad essi;
- GetObservation - accesso ai valori del sensore, nella codifica O&M;
- DescribeSensor - ottiene la descrizione dei diversi sensori, nella codifica SensorML.

Le richieste più specifiche sono le seguenti.

- GetFeatureOfInterest - ottiene la descrizione della caratteristica di interesse associata alla misurazione;
- GetObservation - per richiedere osservazioni con diverse opzioni;
- GetResult - ottieni il risultato dell'osservazione;
- RegisterSensor - immettere nuovi valori per le misurazioni non inserite in precedenza.
- InsertSensor - definisce il nuovo sensore come procedura di osservazione.

Nella prossima sezione ci concentreremo sulle operazioni coinvolte nell'inserimento dei dati e del loro impatto sul modello di dati.

2.1.3 Inserimento dati

Secondo la versione delle specifiche SOS 2.0, il modo comune per inserire nuovi dati su un server SOS è usando il servizio *insertObservation* che richiede una chiamata post http con un pacchetto di dati, in genere in JSON o SOAP.

La figura 2.2 mostra un esempio della chiamata Insert Observation secondo lo standard SOS nel formato dati JSON. Il valore attuale viene passato come un attributo *risultato* del campo *osservazione* insieme all'unità di misura associata.

Poiché lo standard SOS definisce l'operazione *InsertObservation* come una delle cosiddette operazioni transazionali, il server può richiedere un'autenticazione dell'utente.

```
{
  "request": "InsertObservation",
  "service": "SOS",
  "version": "2.0.0",
  "offering": "temperature",
  "observation": {
    "type": "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement",
    "procedure": "temperature",
    "observedProperty": "air:temperature",
    "featureOfInterest": {
      "identifier": {
        "value": "roosvelt",
        "codespace": "http://www.opengis.net/def/nil/OGC/0/unknown"
      },
      "name": [
        {
          "value": "roosvelt",
          "codespace": "http://www.opengis.net/def/nil/OGC/0/unknown"
        }
      ],
      "geometry": {
        "type": "Point",
        "coordinates": [
          38.190234,
          13.350320
        ],
        "crs": {
          "type": "name",
          "properties": {
            "name": "EPSG:4326"
          }
        }
      }
    },
    "phenomenonTime": "2019-05-08T17:17:02+00:00",
    "resultTime": "2019-05-08T17:17:02+00:00",
    "result": {
      "uom": "DEG",
      "value": 22.44
    }
  }
}
```


Figura 2.2: Insert observation pacchetto JSON .

Per definire correttamente una *Procedura* ed un *Offering* da utilizzare in una chiamata di osservazione, è possibile utilizzare una chiamata *InsertSensor* al servizio SOS come mostrato in figura 2.3. Questo esempio della chiamata Insert Sensor segue lo standard SOS e utilizza il formato dati JSON. Contiene un campo per la descrizione della procedura e uno per il suo formato. Il formato di descrizione della procedura si riferisce alla specifica standard sensorML[4] di OpenGIS. Nella descrizione della procedura è definito il suo nome, l'offering, il sito di interesse, gli attributi spaziali del sito di interesse formato dalla latitudine, dalla longitudine, dall'altitudine e dalla proiezione. Contiene anche il nome della proprietà osservata, il tipo di osservazione, OM_Measurement in questo caso, e il tipo di sito del tipo di interesse, che nel nostro caso è un punto di campionamento. La figura 2.4 mostra una risposta positiva quando si inserisce il nuovo sensore.

2.2 Raspberry PI

2.2.1 Panoramica del sistema e descrizione tecnica

Raspberry Pi (RPI) è una piccola scheda di elaborazione con alcune porte integrate IO standard. Può essere definito un minicomputer a basso costo. L'RPI è disponibile in diversi modelli con diversa potenza di elaborazione e IO. Leggeri, piccoli e poco costosi sono le caratteristiche vincenti che rendono l'RPI il minicomputer più popolare sia nella risoluzione rapida di prototipo della ricerca, sia nello sviluppo di piccole soluzioni economicamente vantaggiose, specialmente nell'ambiente scientifico. Applicazioni molto popolari sono state sviluppate da un'enorme comunità liberamente collaborativa come la versione del sistema operativo configurato come media center. Anche se è economico e ha un basso consumo energetico, l'ultima versione di Raspberry risulta abbastanza potente da poter utilizzare una versione light di Windows 10.

Può avere, in base alla versione, la connessione per un monitor tramite una porta HDMI, per Internet via cavo o via Wi-Fi, 4 porte USB per collegare mouse, tastiera e

[3]XML (XML) 1.0, Raccomandazione W3C, 10 febbraio 1998
[4]OpenGIS ® Sensor Model Language (SensorML), 01 maggio 2005, ver. 1.0

```
{
    "request": "InsertSensor",
    "service": "SOS",
    "version": "2.0.0",
    "procedureDescriptionFormat": "http://www.opengis.net/sensorML/1.0.1",
    "procedureDescription": "<sml:SensorML xmlns:swes=\"http://www.opengis.net/swes/2.0\"
xmlns:sos=\"http://www.opengis.net/sos/2.0\"
xmlns:swe=\"http://www.opengis.net/swe/1.0.1\"
xmlns:sml=\"http://www.opengis.net/sensorML/1.0.1\"
xmlns:gml=\"http://www.opengis.net/gml\"
xmlns:xlink=\"http://www.w3.org/1999/xlink\"
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
version=\"1.0.1\"><sml:member><sml:System><sml:identification>
<sml:IdentifierList>  <sml:identifier name=\"uniqueID\">
<sml:Term definition=\"urn:ogc:def:identifier:OGC:1.0:uniqueID\">
<sml:value>temperature</sml:value></sml:Term></sml:identifier>
</sml:IdentifierList>  </sml:identification><sml:capabilities name=\"offerings\">
<swe:SimpleDataRecord>  <swe:field name=\"temperature\">
<swe:Text definition=\"urn:ogc:def:identifier:OGC:offeringID\">
<swe:value>temperature</swe:value></swe:Text></swe:field>
</swe:SimpleDataRecord></sml:capabilities>
<sml:capabilities name=\"parentProcedures\">
<swe:SimpleDataRecord><swe:field name=\"parentProcedure\"><swe:Text>
<swe:value>http://www.52north.org/test/procedure/1</swe:value></swe:Text>
</swe:field> </swe:SimpleDataRecord></sml:capabilities>
<sml:capabilities name=\"featuresOfInterest\"><swe:SimpleDataRecord>
<swe:field name=\"featureOfInterestID\"><swe:Text>
<swe:value>roosvelt</swe:value></swe:Text>
</swe:field></swe:SimpleDataRecord></sml:capabilities>
<sml:position name=\"sensorPosition\">
<swe:Position referenceFrame=\"urn:ogc:def:crs:EPSG::4326\"><swe:location>
<swe:Vector gml:id=\"STATION_LOCATION\"><swe:coordinate name=\"easting\">
<swe:Quantity axisID=\"x\"><swe:uom code=\"degree\"/>
<swe:value>13.350320</swe:value>  </swe:Quantity></swe:coordinate>
<swe:coordinate name=\"northing\"><swe:Quantity axisID=\"y\">
<swe:uom code=\"degree\"/><swe:value>38.190234</swe:value></swe:Quantity>
</swe:coordinate><swe:coordinate name=\"altitude\">
<swe:Quantity axisID=\"z\"><swe:uom code=\"m\"/>
<swe:value>2.0</swe:value></swe:Quantity></swe:coordinate></swe:Vector>
</swe:location></swe:Position></sml:position><sml:inputs><sml:InputList>
<sml:input name=\"air temperature\">
<swe:ObservableProperty definition=\"air temperature\"/></sml:input>
</sml:InputList></sml:inputs><sml:outputs><sml:OutputList>
<sml:output name=\"air temperature\">
<swe:Category definition=\"air temperature\">
<swe:codeSpace xlink:href=\"NOT_DEFINED\"/></swe:Category></sml:output>
</sml:OutputList> </sml:outputs></sml:System></sml:member></sml:SensorML>",
    "observableProperty": "air:temperature",
    "observationType": "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement",
    "featureOfInterestType": "http://www.opengis.net/def/samplingFeatureType/OGC-OM/2.0/SF_SamplingPoint"
}
```

Figura 2.3: Esempio di richiesta inserimento di un sensore.

```
{
  "request" : "InsertSensor",
  "version" : "2.0.0",
  "service" : "SOS",
  "assignedProcedure" : "air temperature",
  "assignedOffering" : "temperature air"
}
```

Figura 2.4: Risposta positiva alla richiesta di inserimento di un sensore.

in modo standard un disco rigido esterno o qualsiasi altra periferica commerciale del computer .

Non è dotato di interfacce hard disk native, come interfacce IDE o ATA, ma utilizza una scheda micro SD per l'archiviazione dei dati. Dispone di un connettore GPIO (General Purpose Input Output) in cui è possibile collegare dispositivi digitali. Ha un ingresso per la connessione della fotocamera e alcune altre interfacce comuni come il connettore jack audio e la connettività Bluetooth.

Le unità di elaborazione sono processori basati su ARM comunemente utilizzati in dispositivi di piccole dimensioni come tablet e smartphone. Grazie alla potenza di elaborazione data da un tempo di alta frequenza, che può essere di Giga Hertz, la configurazione multi core, fino a 4 core nella versione più avanzata, e la buona capacità di RAM, un Giga Byte, è in grado di supportare qualsiasi applicazione che non richiede elaborazione ad alte prestazioni.

2.2.2 Sistema operativo e capacità di programmazione

I sistemi operativi come Raspbian, Pidora, Raspbmc, Ubuntu Mate e Windows10 possono essere installati utilizzando immagini preconfigurate per la scheda SD. Il sistema operativo più comune per l'RPI è Raspbian[5], un sistema operativo derivato da Debian. Mantiene molte funzionalità comuni di Debian, come il sistema di gestione dei pacchetti, il kernel e la compatibilità con gli archivi software standard, quindi ci sono diverse applicazioni che possono essere facilmente scaricate e installate. RPI con Raspbian è un sistema abbastanza solido e flessibile che mantiene un consumo molto basso, in particolare in modalità server, che utilizza l'interfaccia a riga di comando anziché un'interfaccia grafica completa. Pertanto, quando viene utilizzato come dispositivo headless remoto con la porta HDMI disabilitata, il suo consumo energetico è di circa 2 Watt in IDLE e 5 Watt a pieno carico. Sul mercato sono presenti molti accessori dedicati che hanno supporto nativo tramite moduli del kernel standard e accessori per PC comuni con supporto nativo a causa dell'utilizzo del kernel Linux. Avere un kernel Linux con un sistema completo

[5]Sito ufficiale del sistema operativo Raspbian - https://www.raspbian.org/

offre la flessibilità di scrivere software in qualsiasi linguaggio, grazie alla presenza dei compilatori e degli ambienti di sviluppo.

2.2.3 Connettività del sensore

Tra le caratteristiche più interessanti dell'RPI, la presa GPIO, un'intestazione di 40 pin posizionata lungo il bordo superiore della scheda, offre la possibilità di connettersi a semplici dispositivi digitali di input e output, nonché a dispositivi con connessioni più avanzate come l'interfaccia di modulazione di larghezza d'impulso, interfaccia SPI, I2C e canale dati seriale.

Esistono molte schede di espansione compatibili come sensori, set di telecamere, giroscopi, moduli GPS, moduli GPRS o UMTS. Tutte queste interfacce hanno un supporto nativo ed una facile configurazione tramite un file di testo inserito nella partizione di avvio della scheda SD, che è anche facilmente accessibile mentre non è collegata all'RPI poiché utilizza il file system FAT32.

Ci concentriamo ora su un semplice caso di acquisizione dei dati dal sensore fisico. Mostriamo come viene configurato un sensore di temperatura sull'RPI e come recuperare i dati. Il sensore DS18B20 di Maxim Integrated Products Inc. è un termometro digitale a 1 filo con risoluzione programmabile. Il protocollo di comunicazione 1-Wire è un bus di comunicazione, noto anche come MicroLAN, di Dallas Semiconductor / Maxim che implementa un canale di trasmissione half duplex su due fili, il cavo dati e la terra. Forma una rete di dispositivi che utilizzano uno schema multi slave unico. L'RPI funge da master e il sensore termico come slave. È possibile collegare altri dispositivi slave in parallelo sullo stesso cavo dati. Sul master, l'RPI è in grado di gestire la comunicazione con ciascuno degli slave.

Il sensore DS18B20 è in grado di rilevare temperature tra $-55°$C e $+125°$C ($-67°$F a $+257°$F) con una precisione di $\pm 0,5°$C tra $-10°$C e $+85°$C. La sua alimentazione deve essere compresa tra 3 e $5,5$ V e comunica tramite il protocollo 1-Wire attraverso un singolo filo.

```
$ cat /sys/bus/w1/devices/28-0213192111aa/w1_slave
7c 01 4b 46 7f ff 0c 10 7f : crc=7f YES
7c 01 4b 46 7f ff 0c 10 7f t=23750
```

Figura 2.5: Una lettura dal file w1_slave su un sistema Raspbian.

Il primo passo nella configurazione del sensore è aggiungere la riga `dtoverlay = w1-gpio, gpiopin = N`, dove `N` è il numero di pin di comunicazione dati desiderato all'inizio del file`/boot/config.txt`. Dopo il riavvio, il comando `modprobe w1-gpio` e `modprobe w1-therm` caricheranno i moduli del kernel appropriati per gestire il GPIO e il protocollo di comunicazione 1-Wire. Una directory chiamata dopo il numero di serie del sensore appare nel percorso `/sys/bus/w1/devices` contenente il file `w1_slave` che darà accesso al valore corrente letto dal sensore. Una lettura grezza del file `w1_slave`, ad esempio tramite il comando `cat`, fornisce due righe di testo contenenti un codice CRC e il valore della temperatura corrente espresso in millesimi di gradi Celsius come mostrato in figura 2.5.

Capitolo 3

Sistema di acquisizione dati da sensori

Lo scopo di qualsiasi sistema di acquisizione dati è creare un set di informazioni generando un record per ogni osservazione. La disposizione comune di un tale sistema, in particolare nel caso di acquisizione di parametri chimici, è composta da sensori o sonde, un data logger o una piccola unità di elaborazione fisicamente interfacciato con i sensori, e il post processing dei dati, per creare un set di dati coerente compatibile con alcuni formati di dati. In molti casi, si utilizza un file di valori separati da virgola (csv) o una raccolta di file. In questo scenario, aspetti come la convalida dei dati, la conformità al formato, la disponibilità dei dati, l'accessibilità dei dati e persino le specifiche del formato sono aspetti che devono essere affrontati manualmente tramite una soluzione ad hoc.

Possiamo riassumere lo schema esposto in tre blocchi concettuali. 1. il livello fisico con il sensore e la sua interfaccia di comunicazione con il componente intelligente, cioè l'unità di elaborazione; 2. il livello dati con la convalida e la gestione della conversione dei dati da una lettura grezza del sensore a un valore in una giusta unità di misura, la formattazione di tali dati in un formato di archiviazione; 3. il livello di set di dati con la responsabilità di raccogliere i dati nel tempo e di offrire i dati per un successivo utilizzo.

Le soluzioni IoT offrono un facile accesso ai dati sensoriali grezzi attraverso una pletora di dispositivi e protocolli di comunicazione che possono essere facilmente collegati ad un dispositivo intelligente, come l'RPI. I vantaggi dell'utilizzo del Raspberry piuttosto che di costosi sistemi di acquisizione dati personalizzati sono la sua semplicità e conve-

nienza. Avendo un sistema operativo completo simile a Unix potente e flessibile, e avendo il kernel Linux che riconosce automaticamente molti dispositivi non si devono installare manualmente i relativi driver.

Di solito, quando viene utilizzata una soluzione di acquisizione dati personalizzata, la struttura dell'insieme di dati e la sua descrizione sono relegati in un file csv con alcuni commenti di intestazione o uno schema di database. In questo modo è difficile gestire il set di dati perché molte operazioni richiedono spesso un intervento manuale. L'accessibilità dei dati è in genere un problema sottovalutato e le soluzioni personalizzate comuni sono lontane dalla filosofia Open Data che sta ultimamente attirando l'attenzione del pubblico e delle istituzioni governative. La condivisione dei dati e la loro riusabilità è un'attività complessa quando vengono utilizzati formati di set di dati personalizzati, come file csv o database personalizzati, ed è solitamente un'attività che richiede molto tempo, poiché è necessario fornire, oltre ai dati, una descrizione del contenuto. Al contrario, utilizzando un modello di dati standard e una serie di operazioni standard, è facile accedere alle informazioni tramite mezzi automatici poiché i dati sono chiaramente descritti e viene fornito l'accesso online.

Nel caso delle osservazioni del sensore, una soluzione completa per il livello dell'insieme di dati è il servizio SOS che è in grado di gestire grandi quantità di dati e di fornire un accesso a tali dati in un modo facile e ben descritto, supportando anche la query spaziale, descritta nella sezione 2.1. Inoltre, ci sono diversi software compatibili per la visualizzazione e per la manipolazione dei dati. Infatti, molte piattaforme di Spacial Data Infrastructure e Geographic Information System riconoscono il servizio SOS come fonte di dati.

Presentiamo in questo capitolo una nuova soluzione semplice per l'acquisizione remota dei dati basata sull'RPI e lo standard SOS. Supponiamo di avere un RPI in grado di raccogliere dati grezzi da un sensore come, ad esempio, quello descritto nella sezione 2.2.3 dove viene utilizzato un RPI per raccogliere dati da un sensore termico digitale.

3.1 Software Solution

In questo capitolo presentiamo un nuovo sistema di acquisizione dati per le osservazioni dei sensori. L'obiettivo di questo sistema è quello di leggere i dati da un sensore collegato a un dispositivo remoto, come un RPI, e di memorizzare i dati su un server SOS online. L'intento è quello di creare un sistema facile da configurare e mantenere, in grado di gestire i dati in modo semplice e di offrire i dati nella filosofia di Open Data.

Il terminale remoto accessibile per il sistema di osservazione dei sensori (ART4SOS) è una soluzione software che funziona su sistemi Unix-like. Prende i dati del sensore e li memorizza su un server SOS come osservazioni. I dati provengono da un sensore fisico riconosciuto a livello di sistema. Per una descrizione dettagliata della configurazione dei sensori su RPI, rimandiamo il lettore alla sezione 2.2.3. L'attività principale consiste nel gestire la conversione dei dati da una lettura non elaborata di un sensore a un formato di dati compatibile SOS e per la successiva trasmissione a un server SOS. In secondo luogo, affronta problemi come la sincronizzazione dell'ora e le interruzioni di connettività come descritto più avanti in questo capitolo.

I punti chiave del design sono: *Efficienza energetica* da eseguire su dispositivi di piccole e basse potenze; *Flessibilità* da adattare a qualsiasi dispositivo IoT fisico con un basso sforzo manuale; *Affidabilità* e *Manutenibilità* da utilizzare in postazioni remote con un tasso di intervento manuale molto basso richiesto; *Interoperabilità* da collegare a sistemi standardizzati; *Accessibilità dei dati* per promuovere la condivisione dei dati e persino la filosofia dell'Open Data.

Supponiamo di avere accesso ad un server SOS remoto, ad esempio il server SOS 52°North, che è responsabile della creazione di un set di dati di lettura del sensore in base al modello di dati SOS. Per una descrizione dettagliata del modello di dati SOS, fare riferimento alla Sezione 2.1.2 del capitolo precedente. Ciò implica che il terminale remoto, l'RPI nel nostro caso, è connesso a Internet tramite un accesso Wi-Fi o tramite una rete mobile.

ART4SOS è stato suddiviso in tre aree, la gestione temporanea dei dati prima dell'in-

serimento, la convalida dei dati e quindi il formato dei dati e la trasmissione dei dati al server.

Abbiamo progettato ed implementato tre script in Python, che hanno soddisfatto i requisiti di progettazione in termini di semplicità del codice, prestazioni di esecuzione e modello di dati standard e conformità alla comunicazione. Realizzano i seguenti tre passaggi:

- Acquisizione
- Formattazione
- Trasmissione

Una volta disponibili i dati non elaborati, un pacchetto JSON viene preparato in modo conforme al servizio *Insert-Observation* di SOS. I dati sono riferiti a tempo e spazio e sono temporaneamente memorizzati nel terminale remoto in attesa di un accesso a Internet. Quindi i dati vengono trasferiti su un server SOS [1].

Nella seguente sezione forniamo una descrizione migliore di questi tre passaggi della soluzione ART4SOS, oltre a una descrizione dell'installazione del sistema, la configurazione, la descrizione di alcuni aspetti secondari e una panoramica delle prestazioni di esecuzione.

Il prototipo realizzato adotta un Raspberry PI 3+ con una distribuzione Raspbian preinstallata e un sensore di temperatura digitale con connessione a un filo. Il trasferimento dei dati sul server SOS utilizza la connessione Internet gestita a livello di sistema. Inoltre, supponiamo che ci sia un accesso a Internet via Wi-Fi con disponibilità inferiore al 100%.

3.2 Configurazione

Il file di configurazione ART4SOS è un file di testo utilizzato per la configurazione dei parametri di servizio tramite coppie di valori variabili. Comprende vari campi necessari come il nome e le coordinate della stazione di acquisizione, che è il sito di interesse

[1] Il server SOS utilizzato negli esperimenti è la versione SOS. 2.0 di 52° North. Una versione demo è disponibile su http://sensorweb.demo.52north.org/sensorwebtestbed/

```
[DEFAULT]
FEATUREOFINTEREST   = roosvelt
OBSERVABLEPROPERTY  = air:temperature
PROCEDURE           = temperature
OFFERING            = temperature
UNITOFMEASURE       = DEG
LATITUDE            = 38.190234
LONGITUDE           = 13.350320
ALTITUDE            = 2.0
PROJECTION          = EPSG:4326
VERBOSE             = True
PATH                = ./data
SERVER              = http://art4sos.local/sos/service
```

Figura 3.1: Esempio di file di configurazione ART4SOS.

secondo il modello SOS. La definizione spaziale segue lo standard del punto di campionamento ed è espressa in valori di latitudine, longitudine, altitudine e proiezione. Questa definizione spaziale statica è necessaria perché questa versione del sistema non ha un GPS installato. Usiamo la variabile PATH come punto di montaggio di una partizione RAM, e quindi il luogo in cui i file prodotti vengono memorizzati localmente nel buffer. La configurazione include l'interfaccia al servizio SOS, una URL Internet accessibile, che fornisce il servizio per l'inserimento dei dati. Un esempio è riportato in figura 3.1.

Lo script di configurazione ART4SOS è uno script Python che prepara il server SOS a ricevere le osservazioni remote. Usa il servizio *InsertObservation* come mostrato brevemente in figura 3.2. Prende i parametri locali dal suddetto file di configurazione per riempire un modulo JSON standard per il servizio *InsertObservation* come descritto in Sottosezione 2.1.3 ed effettua la chiamata effettiva al servizio SOS. Anche il pacchetto JSON usato viene archiviato localmente. Una volta eseguita questa configurazione, il servizio è pronto per l'immissione automatica dei dati.

3.2.1 Acquisizione dei dati

Questo modulo si occupa di interrogare il sensore con la frequenza temporale configurata, di effettuare la convalida sintattica dei dati, restituendo il controllo alla shell di sistema.

Come descritto nella sezione 2.2.3, si presume che un sensore collegato alla GPIO sia montato sul file system come file virtuale. Nelle nostre semplici impostazioni sperimentali, come mostrato in figura 3.3, questa attività viene eseguita tramite un semplice

```
import os, sys, from datetime import datetime, configparser
try:
   config = configparser.ConfigParser()
   config.read('./art4sos.conf')
   FEATUREOFINTEREST  = config['DEFAULT']['FEATUREOFINTEREST']
   OBSERVABLEPROPERTY = config['DEFAULT']['OBSERVABLEPROPERTY']
   PROCEDURE          = config['DEFAULT']['PROCEDURE']
   OFFERING           = config['DEFAULT']['OFFERING']
   UNITOFMEASURE      = config['DEFAULT']['UNITOFMEASURE']
   LATITUDE           = config['DEFAULT']['LATITUDE']
   LONGITUDE          = config['DEFAULT']['LONGITUDE']
   ALTITUDE           = config['DEFAULT']['ALTITUDE']
   PROJECTION         = config['DEFAULT']['PROJECTION']
   VERBOSE            = config['DEFAULT']['VERBOSE']
   SERVER             = config['DEFAULT']['SERVER']
   USER               = config['DEFAULT']['USER']
   PASSWORD           = config['DEFAULT']['PASSWORD']
except IOError as e:
   print("configuration file not found. quit")
   exit(10)
jsonschema = """{{
"request": "InsertSensor",
"service": "SOS",
"version": "2.0.0",
"procedureDescriptionFormat": "http://www.opengis.net/sensorML/1.0.1",
"procedureDescription": "<sml:SensorML xmlns:swes=\\\"http://www.opengis.net/swes/2.0\\\"
...
</sml:System></sml:member></sml:SensorML>",
"observableProperty": "{}",
"observationType": "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement",
"featureOfInterestType": "http://www.opengis.net/def/samplingFeatureType/OGC-OM/2.0/SF_SamplingPoint"
}}"""
now = datetime.now()
date = now.strftime("%Y-%m-%d")
time = now.strftime("%H:%M:%S")
try:
 jsondata = jsonschema.format(PROCEDURE,PROCEDURE,OFFERING,OFFERING,
 FEATUREOFINTEREST,PROJECTION,LATITUDE,LONGITUDE,ALTITUDE,
 OBSERVABLEPROPERTY)
except KeyError as e:
 print("parameter format error. quit.")
try:
   outputfilename = "insertsensor-{}-{}.json"
   outputfilename = outputfilename.format(date.replace('-',''),time.replace(':',''))
   print("creating file " + outputfilename)
   with open(outputfilename, 'w') as outputfile:
      outputfile.write(jsondata.replace('\n',''))
except KeyError as e:
   print("IO error. quit.")
curlline = "curl  -vs -H \"Content-Type: application/json\" -d \"@{}\" {}"
os.system(curlline.format(outputfilename,SERVER))
```

Figura 3.2: Parti principali del file di script di configurazione ART4SOS.

```
#!/usr/bin/env python
import w1thermsensor
sensor = w1thermsensor.W1ThermSensor()
temp = sensor.get_temperature()
print('{:.2f}'.format(temp))
```

Figura 3.3: Script di acquisizione ART4SOS in Python.

script Python che utilizza una libreria appropriata per gestire un sensore termico tramite MicroLAN one wire[2]. Questo modulo è lasciato volontariamente semplice per chiarezza poiché è semplice da adattare al dispositivo IoT comune tramite il supporto del sistema.

3.2.2 Formato dei dati

Nella figura 3.4 è rappresentato il formato ART4SOS, ovvero uno script Python che assume un valore numerico in INPUT e produce un pacchetto JSON compatibile con il servizio SOS InsertObservation. Usa il tempo di sistema e i valori di configurazione per comporre un pacchetto JSON a partire da un modello. Il file di dati JSON viene salvato sul disco.

3.2.3 Trasmissione dati

Lo script di trasmissione dati ART4SOS è incaricato di effettuare la chiamata effettiva al servizio di inserimento del server SOS selezionato, utilizzando i pacchetti JSON memorizzati sul sistema locale.

Poiché i servizi SOS sono accessibili tramite un'interfaccia Web, la trasmissione dei dati effettua una richiesta http con un pacchetto di dati POST collegato, utilizzando lo strumento `curl`, che è uno strumento a riga di comando per lo scambio di dati su Internet.

La trasmissione di dati ART4SOS è progettata per funzionare in modo asincrono rispetto all'acquisizione dei dati e agli script di formato dei dati, al fine di gestire problemi di connettività, come interruzione della rete o indisponibilità temporanea. Ad ogni esecuzione cerca i file di osservazione bufferizzati localmente e tenta di trasmettere ciascuno di essi al server. Se la trasmissione viene eseguita con successo, rimuove l'osservazione

[2] Alcuni passaggi dell'installazione descritta sono presi da https://github.com/timofurrer/w1thermsensor

```
import os, sys, datetime import datetime, configparser
try:
   config = configparser.ConfigParser()
   config.read('./art4sos.conf')
   FEATUREOFINTEREST  = config['DEFAULT']['FEATUREOFINTEREST']
   OBSERVABLEPROPERTY = config['DEFAULT']['OBSERVABLEPROPERTY']
   PROCEDURE          = config['DEFAULT']['PROCEDURE']
   OFFERING           = config['DEFAULT']['OFFERING']
   UNITOFMEASURE      = config['DEFAULT']['UNITOFMEASURE']
   LATITUDE           = config['DEFAULT']['LATITUDE']
   LONGITUDE          = config['DEFAULT']['LONGITUDE']
   PROJECTION         = config['DEFAULT']['PROJECTION']
   VERBOSE            = config['DEFAULT']['VERBOSE']
   PATH               = config['DEFAULT']['PATH']
   USER               = config['DEFAULT']['USER']
   PASSWORD           = config['DEFAULT']['PASSWORD']
try:
   value = float(sys.argv[1])
except e:
   print("parameter is not numeric. quit")
   exit(2)
jsonschema = """{{
  "request": "InsertObservation",
  "service": "SOS",
  "version": "2.0.0",
  "offering": "{}",
  "observation": {{
    "type": "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement",
    "procedure": "{}",
    "observedProperty": "{}",
    "featureOfInterest": ...,
    "phenomenonTime": "{}",
    "resultTime": "{}",
    "result": {{
      "uom": "{}",
      "value": {}
    }}
  }}
}}"""
now = datetime.now()
date = now.strftime("%Y-%m-%d")
time = now.strftime("%H:%M:%S")
completetime = now.strftime("%Y-%m-%dT%H:%M:%S+00:00")
try:
 jsondata = jsonschema.format(OFFERING, PROCEDURE, OBSERVABLEPROPERTY, FEATUREOFINTEREST, FEATUREOFINTEREST,
 LATITUDE, LONGITUDE, PROJECTION, completetime, completetime, UNITOFMEASURE, value)
except KeyError as e:
    print("parameter format error. quit")
try:
   outputfilename = PATH+'/'+"{}_{}_{}.json"
   outputfilename = outputfilename.format(PROCEDURE,date.replace('-',''),time.replace(':',''))
   with open(outputfilename, 'w') as outputfile:
      outputfile.write(jsondata)
except KeyError as e:
   print("output io error. quit")
```

Figura 3.4: Formato ART4SOS script in Python.

```
import os, subprocess, sys, fileinput, configparser
from datetime import datetime
try:
   config = configparser.ConfigParser()
   config.read('./art4sos.conf')
   SERVER             = config['DEFAULT']['SERVER']
   PATH               = config['DEFAULT']['PATH']
   VERBOSE            = config['DEFAULT']['VERBOSE']
   USER               = config['DEFAULT']['USER']
   PASSWORD           = config['DEFAULT']['PASSWORD']
except IOError as e:
   print("Configuration file not found. Quit.")
   exit(10)
curlline = "curl  -vs -u {}:{} -H \"Content-Type: application/json\" -d \"@{}\" {}"
files = os.listdir(PATH)
for name in files:
   observationFile = PATH+'/'+name
   (ret,output) = subprocess.getstatusoutput(curlline.format(USER,PASSWORD,observationFile,SERVER))
   if ret > 0:
      print("Connection error. "+ name + " not transmitted.")
      continue
   if output.find("exception") > -1:
      print("Service error. "+ name + " not transmitted.")
      continue
   if VERBOSE:
      print(name + "transmitted and locally removed.")
   os.system("rm -f {}".format(json))
```

Figura 3.5: Script in Python della trasmissione di dati ART4SOS.

trasmessa dal filesystem locale, altrimenti il file di osservazione viene lasciato, per essere successivamente ritrasmesso. Quindi, analizza la risposta del server, se ha come risposta `''http 200 OK"` e non contiene alcuna eccezione, la trasmissione viene eseguita correttamente e l'osservazione viene inserita correttamente nel set di dati. La figura 3.5 riporta il codice dello script Python di trasmissione dati ART4SOS.

Secondo le specifiche SOS, il servizio di insert observation è una delle cosiddette operazioni transazionali che di solito richiedono un'autorizzazione. L'autenticazione utente e password viene gestita utilizzando variabili configurabili.

3.3 Funzionalità secondarie

Poiché ART4SOS utilizza il tempo di sistema come tempo dell'osservazione, ovvero la lettura dal sensore, presuppone che l'ora del sistema sia gestita correttamente. Sarebbe facile estendere questo servizio con l'aggiunta del GPS, in modo che possa gestire il tempo in modo indipendente, e in questo modo si può anche aggiornare la georeferenziazione dei dati. Poiché l'RPI non ha un orologio predefinito, ART4SOS risolve il problema di

perdita dell'orario causato da un arresto del sistema, forzando l'ora del sistema da impostare al momento dell'avvio con una chiamata ad un server affidabile (ad es. Google) come descritto nello script di installazione alla sezione 3.4.

Piccole schede informatiche come l'RPI non dispongono di un disco di archiviazione, ma utilizzano invece dispositivi di archiviazione dati di memoria flash. Questi tipi di dispositivi anno cicli di scrittura limitati che potrebbero portare ad un errore del sistema in fase di acquisizione dati.

Il servizio ART4SOS presume di essere connesso ad Internet e non garantisce di salvare l'acquisizione dei dati in caso di perdita di potenza in base alla progettazione. Quindi, utilizza uno spazio disco molto piccolo per il file di configurazione, gli script e per il buffering dei dati acquisiti in attesa di trasmissione dei dati. Per salvare i cicli di scrittura del disco, ART4SOS utilizza un disco virtuale nella RAM per il buffering dei dati come descritto nella sezione 3.4. Un disco basato sulla memoria è un file system che utilizza una porzione di RAM come memoria. Questo tipo di archiviazione è ideale per le applicazioni che richiedono aree di dati ripetutamente ridotte per la memorizzazione nella cache o l'utilizzo come spazio temporaneo. In un'installazione base di Raspian sull'RPI ci sono circa 700 Mega Byte di RAM libera che possono essere utilizzati per la creazione di un disco RAM senza incidere troppo sulle prestazioni del sistema. Nell'impostazione tipica di ART4SOS, che è l'acquisizione di un valore con una frequenza di secondi o minuti, pochi megabyte di spazio sono sufficienti per gestire il buffering di pochi giorni di dati. Poiché ART4SOS produce un file di circa un kilobyte dopo ogni acquisizione, utilizzando l'intera capacità della RAM è possibile bufferizzare tutta la lettura di una settimana alla frequenza di un campionamento al secondo.

3.4 Installazione

L'installazione del servizio ART4SOS viene eseguita tramite uno script Bash come riportato in Figura 3.6. Si presume che il file di configurazione di ART4SOS sia stato impostato correttamente. Inizialmente si esegue lo script di configurazione ART4SOS

```
#!/bin/bash
python3 art4sosconfigure.py

cat << EOF >> /etc/fstab
tmpfs   $PWD/data tmpfs   nodev,nosuid,noexec,nodiratime,size=24M   0 0
EOF

mkdir $PWD/data
mount $PWD/data

(crontab -l && echo "@reboot sleep 30 && date -s \"
    $(wget -qSO- --max-redirect=0 google.com 2>&1 | grep Date: | cut -d' ' -f5-8)Z
    \"") | crontab -

(crontab -u pi -l && echo "* * * * * cd $PWD &&
    python3 $PWD/art4sosformat.py \$(python3 $PWD/art4sosacquisition.py)") | crontab -u pi -

(crontab -u pi -l && echo "* * * * * cd $PWD &&
    python3 $PWD/art4sostransmission.py") | crontab -u pi -
```

Figura 3.6: Script di installazione ART4SOS in Bash.

per definire gli attributi di osservazione sul server SOS. Poi si crea una directory per il buffering locale dei file di dati di osservazione. Questa directory viene utilizzata come punto di montaggio di un disco RAM definito a livello di sistema tramite il file `fstab`. Per motivi di sincronizzazione dell'orologio, a causa della mancanza di un sistema che dia l'orario a bordo dell'RPI, viene definito un comando da eseguire ad ogni avvio attraverso il CRON con privilegi di root. Una semplice chiamata a un sito Web stabile, ad es. Sito Web di Google, viene utilizzato per impostare l'ora del sistema come il tempo indicato nell'intestazione della risposta. Il sistema di gestione CRON viene anche utilizzato per impostare l'esecuzione periodica degli script di acquisizione dati e di formato dati ART4SOS, in una singola chiamata di sistema che combina il valore attraverso la pipe di sistema. Un'ulteriore chiamata di sistema viene utilizzata per eseguire periodicamente lo script di trasmissione ART4SOS.

3.5 Configurazione

Lo script di configurazione ART4SOS è uno script Python che prepara il server SOS a ricevere le osservazioni remote. Prende i parametri locali dal suddetto file di configurazione per riempire un modulo JSON standard per il servizio *InsertObservation* e effettua la chiamata effettiva al servizio SOS. Anche il pacchetto JSON usato viene archiviato lo-

calmente. Una volta eseguita questa configurazione il servizio è pronto per l'immissione automatica dei dati.

```
import os, sys, from datetime import datetime, configparser
try:
   config = configparser.ConfigParser()
   config.read('./art4sos.conf')
   FEATUREOFINTEREST  = config['DEFAULT']['FEATUREOFINTEREST']
   OBSERVABLEPROPERTY = config['DEFAULT']['OBSERVABLEPROPERTY']
   PROCEDURE          = config['DEFAULT']['PROCEDURE']
   OFFERING           = config['DEFAULT']['OFFERING']
   UNITOFMEASURE      = config['DEFAULT']['UNITOFMEASURE']
   LATITUDE           = config['DEFAULT']['LATITUDE']
   LONGITUDE          = config['DEFAULT']['LONGITUDE']
   ALTITUDE           = config['DEFAULT']['ALTITUDE']
   PROJECTION         = config['DEFAULT']['PROJECTION']
   VERBOSE            = config['DEFAULT']['VERBOSE']
   SERVER             = config['DEFAULT']['SERVER']
   USER               = config['DEFAULT']['USER']
   PASSWORD           = config['DEFAULT']['PASSWORD']
except IOError as e:
   print("configuration file not found. quit")
   exit(10)
jsonschema = """{{
"request": "InsertSensor",
"service": "SOS",
"version": "2.0.0",
"procedureDescriptionFormat": "http://www.opengis.net/sensorML/1.0.1",
"procedureDescription": "
<sml:SensorML xmlns:swes=\\\"http://www.opengis.net/swes/2.0\\\"
xmlns:sos=\\\"http://www.opengis.net/sos/2.0\\\"
xmlns:swe=\\\"http://www.opengis.net/swe/1.0.1\\\"
xmlns:sml=\\\"http://www.opengis.net/sensorML/1.0.1\\\"
xmlns:gml=\\\"http://www.opengis.net/gml\\\"
xmlns:xlink=\\\"http://www.w3.org/1999/xlink\\\"
xmlns:xsi=\\\"http://www.w3.org/2001/XMLSchema-instance\\\"
version=\\\"1.0.1\\\">

<sml:member><sml:System>
<sml:identification><sml:IdentifierList><sml:identifier name=\\\"{}\\\">
<sml:Term definition=\\\"urn:ogc:def:identifier:OGC:1.0:uniqueID\\\">
<sml:value>{}</sml:value></sml:Term>
</sml:identifier></sml:IdentifierList> </sml:identification>
<sml:capabilities name=\\\"offerings\\\"><swe:SimpleDataRecord>
<swe:field name=\\\"{}\\\"><swe:Text
definition=\\\"urn:ogc:def:identifier:OGC:offeringID\\\">
<swe:value>{}</swe:value></swe:Text></swe:field>
</swe:SimpleDataRecord></sml:capabilities>
<sml:capabilities name=\\\"parentProcedures\\\">
<swe:SimpleDataRecord><swe:field name=\\\"parentProcedure\\\">
<swe:Text> <swe:value>void</swe:value></swe:Text>
</swe:field> </swe:SimpleDataRecord></sml:capabilities>
<sml:capabilities name=\\\"featuresOfInterest\\\">
<swe:SimpleDataRecord>
<swe:field name=\\\"featureOfInterestID\\\">
<swe:Text>  <swe:value>{}</swe:value></swe:Text>
</swe:field></swe:SimpleDataRecord></sml:capabilities>

<sml:position name=\\\"sensorPosition\\\">
<swe:Position referenceFrame=\\\"urn:ogc:def:crs:{}\\\"><swe:location>
<swe:Vector gml:id=\\\"STATION_LOCATION\\\">
<swe:coordinate name=\\\"easting\\\">
<swe:Quantity axisID=\\\"x\\\"><swe:uom code=\\\"degree\\\"/>
<swe:value>{}</swe:value></swe:Quantity>
</swe:coordinate>
<swe:coordinate name=\\\"northing\\\">
<swe:Quantity axisID=\\\"y\\\"><swe:uom code=\\\"degree\\\"/><swe:value>{}</swe:value>
</swe:Quantity></swe:coordinate>
<swe:coordinate name=\\\"altitude\\\">
```

```
<swe:Quantity axisID=\\\"z\\\"><swe:uom code=\\\"m\\\"/>
<swe:value>{}</swe:value></swe:Quantity></swe:coordinate>
</swe:Vector></swe:location></swe:Position>
</sml:position>

</sml:System></sml:member></sml:SensorML>
",
"observableProperty": "{}",
"observationType": "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement",
"featureOfInterestType": "http://www.opengis.net/def/samplingFeatureType/OGC-OM/2.0/SF_SamplingPoint"
}}"""
now = datetime.now()
date = now.strftime("%Y-%m-%d")
time = now.strftime("%H:%M:%S")
try:
 jsondata = jsonschema.format(PROCEDURE,PROCEDURE,OFFERING,OFFERING,
 FEATUREOFINTEREST,PROJECTION,LATITUDE,LONGITUDE,ALTITUDE,
 OBSERVABLEPROPERTY)
except KeyError as e:
 print("parameter format error. quit.")
try:
   outputfilename = "insertsensor-{}-{}.json"
   outputfilename = outputfilename.format(date.replace('-',''),time.replace(':',''))
   print("creating file " + outputfilename)
   with open(outputfilename, 'w') as outputfile:
      outputfile.write(jsondata.replace('\n',''))
except KeyError as e:
   print("IO error. quit.")
curlline = "curl  -vs -H \"Content-Type: application/json\" -d \"@{}\" {}"
os.system(curlline.format(outputfilename,SERVER))
```

3.6 Analisi della Soluzione

Mediante la soluzione adottata si raggiunge l'accessibilità dei dati, la sicurezza dei dati, l'interoperabilità utilizzando i protocolli e i formati standard SOS e un server remoto SOS. Si garantisce manutenibilità, riutilizzabilità e affidabilità utilizzando strumenti Linux combinati con semplici script Python e Bash, nonché standard affidabili. La semplicità del codice contribuisce a ridurre eventuali errori dovuti all'intervento manuale. Può funzionare su dispositivi di piccole dimensioni basati su IoT comuni poiché utilizza risorse di sistema molto basse in termini di potenza di calcolo e utilizzo della memoria. Il codice è disponibile online per l'uso e la modifica gratuiti.

Nel nostro prototipo, abbiamo utilizzato il RPI 3 B + con un sistema operativo base, utilizzando meno del 1% di CPU con e senza l'esecuzione di ART4SOS, secondo quanto riportato dallo strumento da riga di comando superiore. Pertanto, il calcolo e l'impatto della potenza su questo dispositivo non è apprezzabile in scala macro. L'impronta di memoria di qualsiasi script in esecuzione di ART4SOS è inferiore a dieci megabyte a causa del framework Python. La larghezza di banda della trasmissione è proporzionale ai

Tabella 3.1: Risultato del tempo GNU dello script di acquisizione, formato e trasmissione, con utilizzo di Tempo, CPU e memoria.

Script	Time (sec.)	CPU (%)	Memory (KB)
Acquisition	0.21	22%	7924
Format	0.21	99%	7896
Trasmission	0.26	67%	8148

pacchetti JSON più un piccolo overhead dovuto al protocollo http. La riduzione del carico di rete può essere ottenuta utilizzando strategie comuni non trattate in questo contesto.

Un'indagine è stata condotta utilizzando lo strumento della riga di comando GNU `time` per ogni script e i risultati sono riportati in tabella 3.6. L'utilizzo della CPU è basso per l'acquisizione e la trasmissione mentre attendono risorse esterne come IO e rete. La trasmissione utilizza tutta la potenza di elaborazione solo per brevissimo tempo poiché utilizza solo IO sul disco RAM. Hanno quasi la stessa impronta di memoria dovuta all'uso del framework Python.

Mentre si utilizzano i servizi standard SOS è possibile la visualizzazione dei dati del sensore, si pensi all'interfaccia grafica open source Helgoland [3], che è uno strumento di visualizzazione dei dati del sensore basato su SOS compatibile con il web. La figura 3.7 riporta un grafico dei dati raccolti nel nostro prototipo di sviluppo, in cui viene mostrato un elenco e un diagramma dei valori del sensore.

[3]Esplorazione visiva e analisi dei dati web del sensore. Versione 1.0 - https://52north.org/software/software-projects/helgoland/

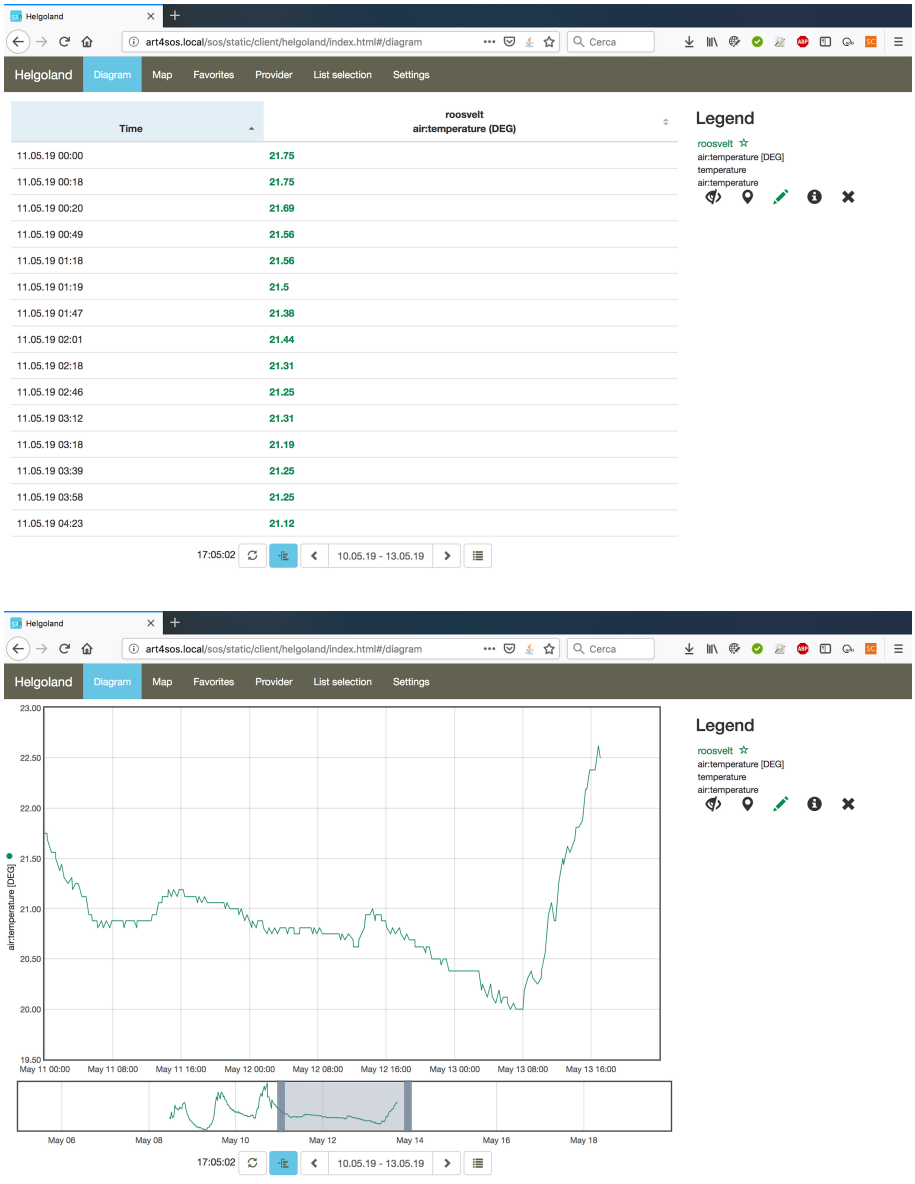

Figura 3.7: Schermate dell'interfaccia grafica che mostra i dati inviati da ART4SOS.

Capitolo 4

Conclusioni

Questo rapporto tecnico presenta una nuova soluzione software per l'acquisizione dei dati del sensore basata su una scheda di elaborazione a basso costo con sensore e connettività Internet a livello di sistema. Sfrutta le funzionalità standard del Sistema di osservazione del sensore come il modello di dati e i formati di comunicazione, presupponendo la disponibilità di un server SOS per l'archiviazione. Questa impostazione è particolarmente adatta al monitoraggio ambientale attraverso la misurazione ripetitiva di parametri fisici/chimici. La soluzione presentata garantisce l'accessibilità dei dati, la sicurezza dei dati, l'interoperabilità utilizzando lo standard SOS e un server remoto SOS e raggiunge la manutenibilità, la riusabilità e l'affidabilità utilizzando strumenti Linux combinati con semplici script Python e Bash. Può funzionare su dispositivi di piccole dimensioni basati su IoT comuni poiché utilizza risorse di sistema molto basse, in termini di potenza di calcolo e utilizzo della memoria. ART4SOS è disponibile su GitHub al seguente indirizzo: `https://github.com/lauralagattutacnr/art4sos`