



MASTER DEGREE in EMBEDDED COMPUTING SYSTEMS

A.Y. 2017 - 2018

Relazione Progetto

Internet of Things

Professore
Enzo Mingozzi

Studenti
Silvio Bacci
Andrea Baldecchi

Indice

1	Introduzione	3
1.1	Monitoraggio dei corsi d'acqua	3
1.2	Panoramica sull'implementazione	3
2	Tecnologie di rete	5
2.1	Introduzione	5
2.2	OneM2M	5
2.3	Coap	5
2.4	JSON	5
2.5	Apache Tomcat Server	5
2.6	SSE	6
3	Sensori e Attuatori nella 6LoWPAN	7
3.1	Simulazione dei corsi d'acqua	7
3.2	Sensori e dighe	7
4	ADN lato MN	9
4.1	Introduzione	9
4.2	Raccolta dati	9
4.3	Gestione Middle Node	10
4.3.1	Subscription	12
4.4	Controllo	13
4.5	Controllo manuale dei mote	13
5	ADN lato IN	15
5.1	Introduzione	15
5.2	Copia su IN	15
6	Applicazione WEB	19
6.1	Interfaccia e Canvas	19
6.2	Interfaccia e Canvas	19
6.3	Tomcat server	23
7	Conclusioni	25
8	Manuale utente	26

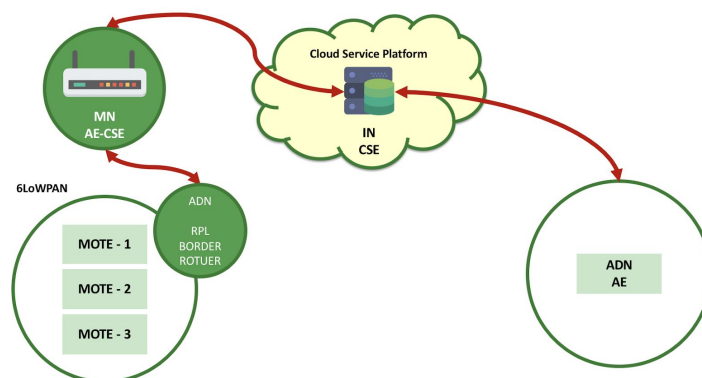
1 Introduzione

1.1 Monitoraggio dei corsi d'acqua

Il sistema da noi sviluppato è costituito un'applicazione distribuita dedicata al monitoraggio di flussi d'acqua naturali e artificiali. Lo scopo principale di questo sistema è quello di monitorare in tempo reale lo stato di corsi d'acqua che, in caso di esondazione, possono causare danni più o meno catastrofici ad ambienti urbani limitrofi o addirittura a persone fisiche. Il monitoraggio sarà effettuato per mezzo di appositi sensori in grado di rilevare in ogni momento il livello d'acqua presente. Oltre alla funzionalità di monitoraggio il sistema permette inoltre di comandare, in modo automatico o manuale, l'apertura di apposite dighe in modo tale da scongiurare esondazioni. In aggiunta, il sistema è in grado di fornire informazioni riguardanti i corsi d'acqua controllati grazie ad una pagina web dedicata in cui, oltre alle informazioni in tempo reale, vengono visualizzati anche alcuni storici utilizzabili per condurre analisi ambientali.

1.2 Panoramica sull'implementazione

Il sistema è stato realizzato sfruttando la piattaforma di rete oneM2M. In particolar modo l'idea è quella di riprendere la seguente classica struttura di un'applicazione sviluppata su questa piattaforma:



Come possiamo notare dalla figura il sistema può essere scomposto in tre parti geograficamente distribuite. La prima parte è composta da una rete 6LoWPAN nella quale, sensori idrici e dighe vengono interconnesse tra loro. La rete viene gestita da uno o più border router che, interfacciandosi verso l'esterno, rendono possibili le comunicazioni con i moduli software di controllo dedicati. Tali moduli, costituiscono la seconda parte del sistema. Questi si occupano di raccogliere i dati provenienti dai sensori idrici che verranno poi inoltrati in rete sfruttando la piattaforma di rete oneM2M. I moduli software implementano inoltre la funzionalità di controllo dei flussi in quanto una logica di controllo viene utilizzata per aprire e chiudere

automaticamente le dighe in funzione dello stato dei sensori ad esse associati. L'ultima parte è quella che si occupa della raccolta centralizzata dei dati prodotti, della loro elaborazione e visualizzazioni mediante pagina web. E' importante menzionare anche il fatto che la pagina web permette il controllo manuale, da parte dei soli utenti amministratori, delle dighe e dei parametri di configurazione dei vari sensori.

Nei prossimi capitoli andremo a descrivere in dettaglio le tre componenti appena introdotte in modo da apprezzare meglio le caratteristiche peculiari del sistema.

2 Tecnologie di rete

2.1 Introduzione

Prima di passare alla trattazione più specifica del sistema è opportuno menzionare le tecnologie, i linguaggi e i protocolli di rete che abbiamo deciso di utilizzare per lo sviluppo del sistema.

2.2 OneM2M

OneM2M è un'organizzazione globale che crea requisiti, architetture, specifiche API e altro ancora per sistemi machine-to-machine e IoT. In particolar modo abbiamo fatto uso dell'implementazione Java OM2M che abbiamo conosciuto durante le esercitazioni pratiche del corso.

2.3 Coap

Per ogni tipo di comunicazione necessaria nel nostro sistema (ad eccezione della connessione tra utente finale e server web) abbiamo scelto di utilizzare il protocollo CoAP anziché il protocollo HTTP (comunque supportato da Middle Node e Infrastructure Node di OM2M).

2.4 JSON

Per lo scambio di dati complessi e per il salvataggio di informazioni di configurazione utili abbiamo utilizzato il formato dati JSON che è ben supportato dalla maggior parte dei linguaggi di programmazione che abbiamo utilizzato per l'implementazione del sistema (c, Java, Javascript, ecc..).

2.5 Apache Tomcat Server

Per la realizzazione della parte server abbiamo scelto di utilizzare **Apache Tomcat** che è un web server, nella forma di contenitore servlet, open source sviluppato dalla Apache Software Foundation. Per poter implementare la parte server che offre sia la possibilità di creare vere e proprie pagine web (JSP) sia quella di creare i cosiddetti servlet, ovvero oggetti Java che possono essere “contattati” tramite le classiche operazioni HTTP. Il nostro utilizzo del server Tomcat si è limitato al caricamento del codice eseguito dal client (HTML, CSS, Javascript, ecc..) e alla creazione di appositi servlet utilizzati per la gestione dell'applicazione web. Come vedremo più avanti, Tomcat è stato inoltre utilizzato per eseguire il codice del ADN responsabile della gestione dei contenuti memorizzati sul Infrastructure Node di OM2M.

2.6 SSE

Per la gestione dinamica degli aggiornamenti della pagina web l'idea è stata quella di utilizzare SSE, ovvero una tecnologia che permette al browser di ricevere aggiornamenti direttamente dal server senza dover effettuare un polling inviando delle continue richieste GET.

3 Sensori e Attuatori nella 6LoWPAN

3.1 Simulazione dei corsi d'acqua

Non avendo a disposizione le strutture e i dispositivi fisici necessari per poter costruire un prototipo del nostro sistema, abbiamo optato per una simulazione di sensori, dighe ed in generale di tutta la rete 6LoWPAN utilizzando il software Cooja. I microcontrollori virtuali associati a dighe e sensori scelte sono gli Zolertia Z1. Queste unità elaborative sono alimentate da batterie dedicate, le capacità computazionali ed elaborative sono quindi relativamente limitate. Il sistema operativo utilizzato è Contiki OS, un sistema operativo open source appositamente progettato per microcontrollori a basso consumo.

3.2 Sensori e dighe

L'implementazione dei sensori e delle dighe è stata realizzata mediante linguaggio C. I sensori espongono due risorse Coap. Una risorsa, simulando un modulo GPS, contiene le informazioni relative alla locazione dei sensori. Le coordinate spaziali non sono ovviamente consistenti con il mondo reale e vengono utilizzate esclusivamente per due scopi:

1. Mappare i sensori simulati sugli elementi grafici della pagina web
2. Associare i sensori alle dighe per poter realizzare la logica di controllo

Per ottenere queste coordinate ci siamo serviti dello script editor di Cooja. In particolar modo un semplice script realizzato in JavaScript si occupa di fornire ai vari mote le proprie coordinate su richiesta, sfruttando una comunicazione seriale.

La seconda risorsa invece contiene le informazioni relative allo stato dei sensori. In particolare possiamo individuare:

- Livello idrico rilevato
- Parametri di configurazione:
 - Livello idrico minimo
 - Livello idrico massimo
 - Soglia idrica di allerta

E' inoltre presente un ulteriore parametro utilizzato per la simulazione dell'andamento del livello idrico. Infatti, non avendo a disposizione sensori fisici da poter installare l'evoluzione del corso d'acqua viene realizzata manipolando questo parametro chiamato *evolution*. La simulazione dei valori è molto semplice. Si sfrutta un timer di sistema per modificare periodicamente il livello idrico del sensore. L'incremento periodico viene generato in modo pseudo-casuale all'interno di un intervallo predefinito.

L'implementazione delle dighe è molto più semplice. Anche in questo caso vengono esposte due risorse, una per le coordinate spaziali e l'altra per lo stato della diga (chiusa/aperta). In questo caso non è necessario implementare alcuno tipo di simulazione, la risorsa associata allo stato della diga dovrà semplicemente rispondere alle richieste coap cambiando lo stato della diga stessa.

4 ADN lato MN

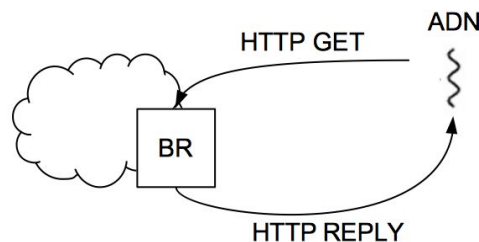
4.1 Introduzione

Questo module software è stato implementato in Java, utilizzando il framework CoAP Californium. Californium fornisce un set di API relative a servizi web di tipo RESTful, indispensabili per il funzionamento del modulo. Le funzioni svolte da questo modulo sono le seguenti:

1. Raccolta dei dati provenienti dai mote
2. Gestione del Middle Node associato
3. Controllo delle dighe
4. Controllo manuale dei mote

4.2 Raccolta dati

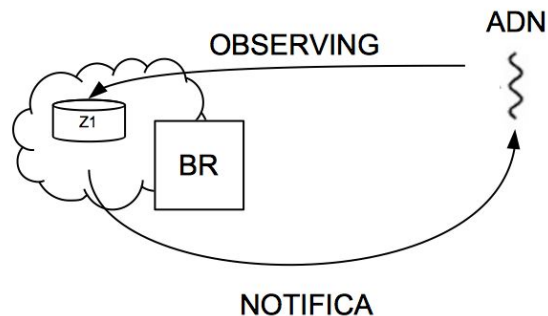
Questa funzione è stata implementata in un thread periodico che si occupa di comunicare con il border router della rete 6LoWPAN. Una volta ottenuti gli indirizzi dei moti gestiti dal border router (attraverso una richiesta HTTP), il modulo software memorizza al suo interno le informazioni provenienti dalle risorse CoAP esposte dai motes. In particolare, per ogni mote, viene effettuata una richiesta CoAP alla risorsa *well-known/core*. In questo modo, grazie agli attributi delle risorse il modulo è in grado di distinguere sensori e dighe avviando le opportune funzioni di inizializzazione.



Utilizzando l'oggetto *CoapHandler* del framework *Californium* il modulo effettua un "observing" delle risorse esposte dai motes. Sfruttando questo meccanismo il modulo è quindi in grado di:

1. Ricevere gli aggiornamenti sullo stato delle risorse esposte
2. Rilevare l'inattività dei motes

L'inattività di un mote infatti viene rilevata sfruttando le notifiche periodiche (previste dall'operazione di observing) che vengono inviate anche se lo stato interno delle risorse osservate rimane inalterato. Una volta rilevata l'inattività di un mote (rilevata quando non si ricevono più notifiche) il modulo elimina tutte le strutture interne associate ed ovviamente inoltra una notifica anche al Middle Node come illustrato nel paragrafo successivo.



4.3 Gestione Middle Node

In fase di avvio il modulo crea un Application Entity sul Middle Node associato (l'indirizzo del MN viene indicato in un apposito file di configurazione) al corso d'acqua monitorato. Questo conterrà la seguente struttura predefinita:

Logout
OM2M CSE Resource Tree
<http://127.0.0.1:8282/-/SWFM-mn-cse/CAE401961669>

```

- SWFM-mn-name
  - acp_admin
  - Pesca
    - DAMS
    - SENSORS
    - STATE
    - GPS

```

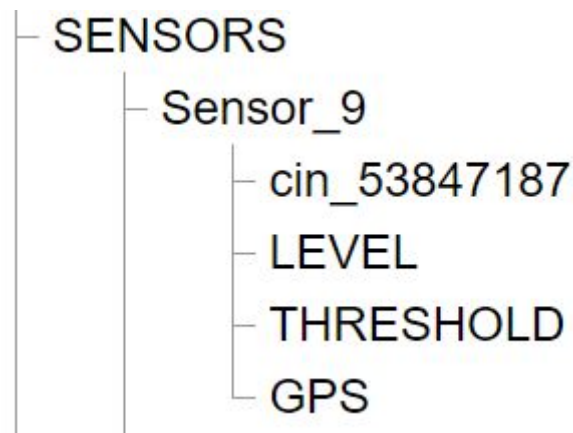


Attribute	Value
rn	Pescia
ty	2
ri	/SWFM-mn-cse/CAE401961669
pi	/SWFM-mn-cse
ct	20180421T190648
lt	20180421T190648
acpi	<div>AccessControlPolicyIDs</div> <div>/SWFM-mn-cse/acp-491663586</div>
et	20190421T190648
api	Pescia-ID
ael	CAE401961669
rr	true

L'Application Entity prende il nome del corso d'acqua monitorato. Questa informazione è contenuta anch'essa in un opportuno file di configurazione. All'interno di questo AE come si nota in figura troviamo diversi container:

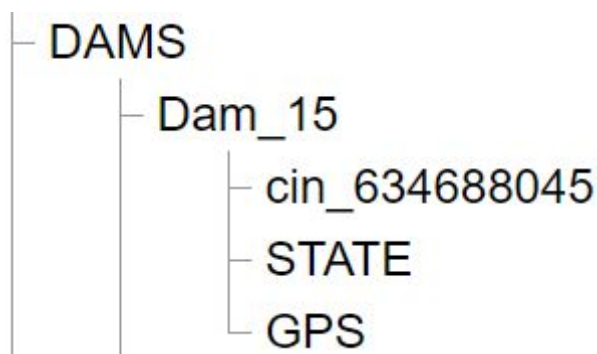
1. STATE: Contiene le informazioni generali relative alla situazione attuale del corso d'acqua.
2. GPS: Contiene le coordinate latitudinali e longitudinali della rete di monitoraggio
3. SENSORS: Contiene i container associati ai sensori rilevati
4. DAMS: Contiene i container associati alle dighe rilevate

Il contenitore associato ad un determinato sensore ha la seguente organizzazione interna.



Il contenitore GPS racchiude le informazioni relative alla locazione spaziale del sensore (nello spazio virtuale di Cooja). Il container THRESHOLD contiene invece le informazioni relative ai parametri di configurazione del sensore menzionate nei capitoli precedenti. Infine, il container LEVEL, raggrupperà le informazioni relative al livello idrico rilevato nel tempo.

Il contenitore associato ad una diga ha invece la seguente struttura:

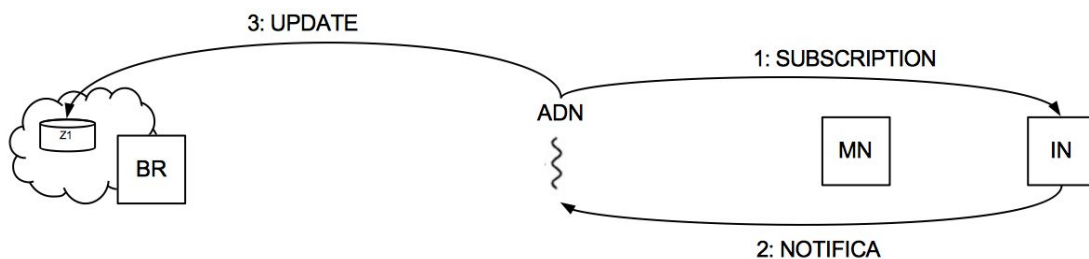


Il contenitore GPS è analogo al caso dei sensori. Il contenitore STATE invece contiene le informazioni relative allo stato della diga nel tempo.

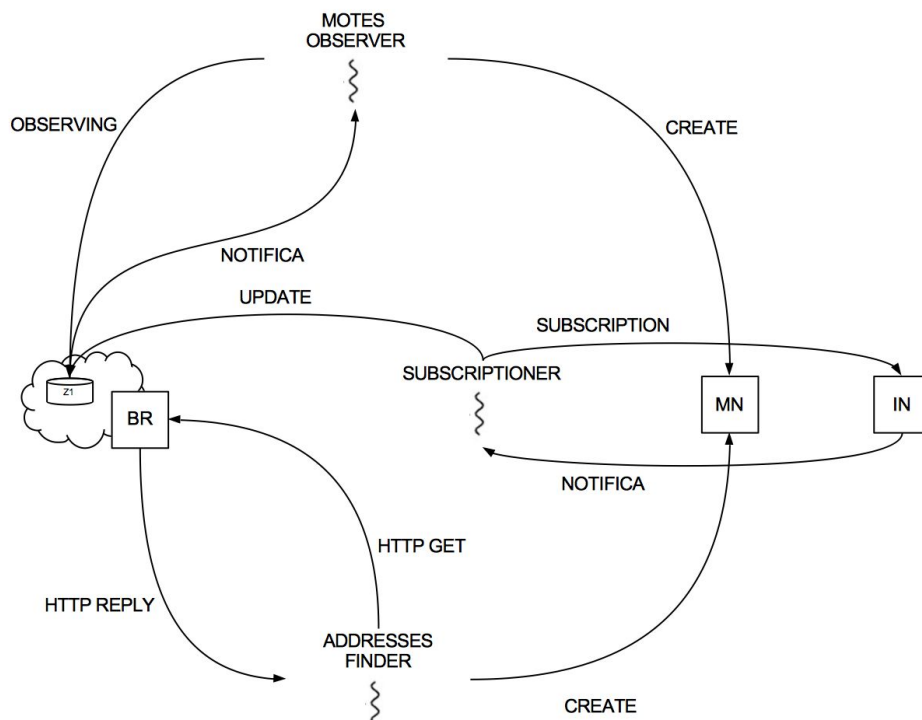
Infine è importante menzionare che all'interno del container associato ai motes verranno create delle content instance relative allo stato attuale del mote. Ogni volta che un mote cambia il suo stato, passando da attivo ad inattivo e viceversa verrà aggiunta una content instance contenente tale informazione.

4.3.1 Subscription

All'interno dei contenitori modificabili dal ADN lato IN (capitolo successivo) vengono create delle subscription in modo tale da ricevere una notifica ogni volta che un utente amministratore crea una nuova content instance in questi contenitori. In questa fase è necessario specificare, oltre che al nome della risorsa, l'indirizzo e la porta su cui oneM2M inoltrerà le notifiche. Per utilizzare questo meccanismo si sfruttano le classi CoapServer e CoapResource di Californium. In particolare viene inizializzato un CoapServer contenente un oggetto CoapResource. Il CoapServer si mette in ascolto sulla stessa porta e sullo stesso indirizzo specificati durante la creazione della subscription. All'arrivo di una notifica il modulo software rileva il contenitore per cui è arrivata la notifica (sensore o diga) e, in funzione di questo, effettua delle richieste POST CoAP direttamente ai mote virtualizzati in Cooja.



Per riassumere tutte le caratteristiche di questo modulo software possiamo considerare la seguente immagine in cui tutte le sue funzionalità sono state suddivise in tre differenti thread:



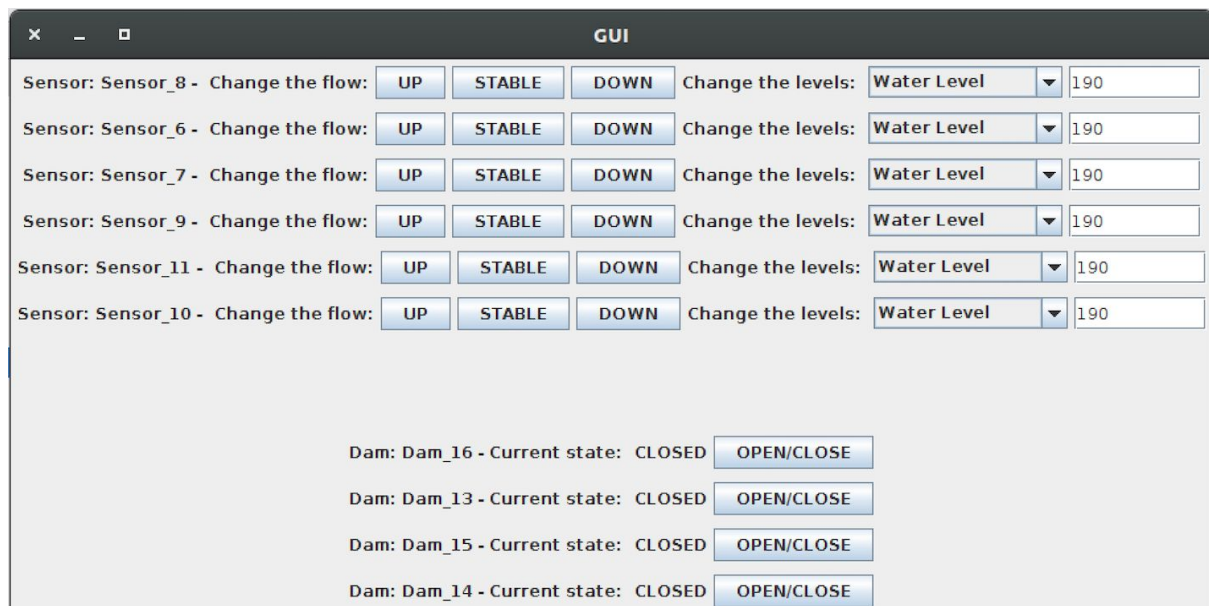
4.4 Controllo

Il thread periodico che si occupa del controllo implementa una logica relativamente semplice. Prima di attuare tale logica vengono calcolate le associazioni tra sensori e dighe, queste vengono realizzate sfruttando la coordinate spaziali in Cooja ed associando quindi i sensori alle dighe a loro più vicine. A questo punto il thread sarà in grado di attuare la logica di controllo. Ci serviamo del seguente elenco puntato per illustrare la logica implementata.

1. Considera ogni diga
 - a. Se la diga è chiusa
 - i. Considera ogni sensore ad essa associato
 1. Se almeno un sensore è in overflow:
 - a. Apri la diga
 - b. Fai scendere il livello del idrico del sensore
 - b. Se la diga è aperta
 - i. Considera ogni sensore ad essa associato
 1. Se nessun sensore è in overflow:
 - a. Chiudi la diga
 - b. Stabilizza i livello idrici

4.5 Controllo manuale dei mote

L'ADN espone inoltre un interfaccia grafica molto semplice dalla quale è possibile gestire tutti i parametri dei mote presenti nella rete 6LoWPAN. Riportiamo qui sotto la GUI.



Da notare che il cambiamento dei valori nella GUI dipende esclusivamente dalla risposta dei motes da Cooja. Questo significa si può verificare un po' di ritardo tra il cambiamento dei valori della GUI e il cambiamento dei valori sulla pagina web, che avviene soltanto al caricamento dei dati aggiornati su OneM2M. Quest'ultimo caricamento è piuttosto oneroso per la GUI e quindi il ritardo tende ad essere piuttosto evidente.

5 ADN lato IN

5.1 Introduzione

Questo modulo software è il componente complementare rispetto a quello descritto nel capitolo precedente ma con alcune differenze importanti. In particolar modo questo software ha il compito di eseguire due principali funzioni. La prima funzione è quella di copiare il contenuto dei vari Middle Node direttamente sul Infrastructure Node (la motivazione per cui questa copia viene eseguita verrà dettagliatamente argomentata nel prossimo paragrafo). La seconda funzione è quella di notificare eventuali cambiamenti di stato delle risorse ai servlet presenti sul web server, in modo da inviare aggiornamenti dinamici (tramite SSE) ai vari browser connessi.

5.2 Copia su IN

Un problema che ci siamo posti in fase di progettazione del nostro sistema è la gestione del sistema in caso di una (anche momentanea) mancanza di connessione tra IN e MN. La nostra prima idea era quella di memorizzare tutte le informazioni relative alle varie reti 6LoWPAN soltanto sui vari MN. Ovviamente in questo caso per mostrare i contenuti sulla pagina web, saremmo costretti a contattare direttamente il MN utilizzando le risorse remoteCSE presenti sul IN. In caso di mancata connessione però, non avremmo nessun dato da poter mostrare sulla pagina web.

L'idea alla base della risoluzione di questo problema è stata quella di creare una copia locale del contenuto dello specifico MN su IN e interrogare successivamente l'IN stesso per ottenere le informazioni desiderate.

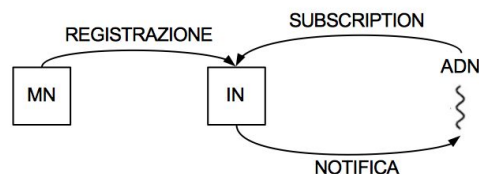
La copia effettiva non è un processo semplice da eseguire e per poterla compiere al meglio, abbiamo sfruttato tutte le caratteristiche messe a disposizione da OM2M. In particolar modo abbiamo fatto largo uso delle *subscription* in modo da ottenere aggiornamenti continui sulla creazione di risorse sui vari nodi. Per poter spiegare al meglio come viene raggiunta la copia è necessario trattare inizialmente il caso base in cui sono presenti soltanto un IN e il nostro modulo software.



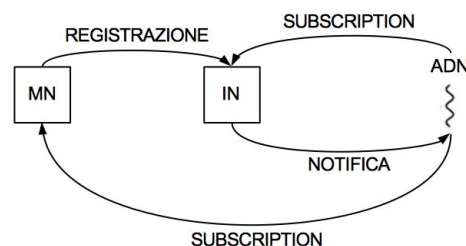
In questo caso il thread creerà una subscription sull'IN in modo tale da ricevere aggiornamenti non appena un MN si registrerà presso l'IN.



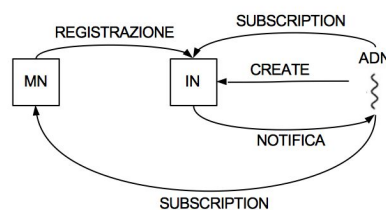
Non appena un nodo si registra presso l'IN, questo crea una risorsa *remoteCSE*. Contestualmente alla creazione della risorsa viene inviata una notifica della suddetta creazione al nostro thread. Adesso la situazione vede presenti IN, un unico MN e il nostro thread.



Non appena il nostro thread riceverà la notifica effettuerà tre operazioni distinte. La prima operazione che viene effettuata è la creazione di una subscription sul MN in modo che il nostro thread venga notificato non appena una risorsa viene creata sul MN (in modo da poterla copiare tempestivamente).



La seconda operazione che viene effettuata è la creazione di un AE direttamente sul IN con lo stesso nome dell'ID del *remoteCSE* appena notificato (in questo modo siamo sicuri che il nome di questo AE sarà unico all'interno del IN). Proprio all'interno di questo AE verrà memorizzata la copia di tutte le risorse presenti sul relativo MN.




Di seguito possiamo notare la struttura che viene quindi a crearsi sull'IN al termine di questa operazione:

[Logout](#)

OM2M CSE Resource Tree

<http://127.0.0.1:8080/-/SWFM-in-cse/csr-259315043>

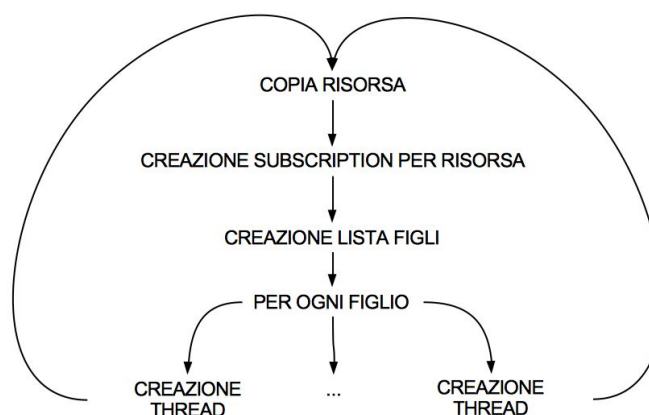
- SWFM-in-name
 - acp_admin
 - csr-259315043
 - SWFM-mn-name-SILVIO
 - subscription_SWFM-in-cse



Attribute	Value
rn	SWFM-mn-name-SILVIO
ty	16
ri	/SWFM-in-cse/csr-259315043
pi	/SWFM-in-cse
ct	20180421T22104Z
lt	20180421T22104Z
acpi	<div style="background-color: #e6f2ff; padding: 2px; text-align: center;">AccessControlPolicyIDs</div> <div style="border: 1px solid #ccc; padding: 2px;">/SWFM-in-cse/acp-927588171</div>
poa	<div style="background-color: #e6f2ff; padding: 2px; text-align: center;">Point Of Access</div> <div style="border: 1px solid #ccc; padding: 2px;">http://192.168.1.71:8282/</div>
cb	//om2m.org/SWFM-mn-cse-SILVIO
csi	/SWFM-mn-cse-SILVIO
rr	true

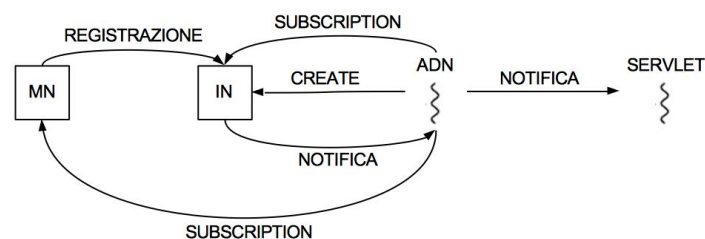
La terza operazione consiste nella copia vera e propria. Il thread infatti si occupa di esplorare l'albero del MN presente in quell'istante e di copiarlo direttamente sul IN. Ovviamente questa copia deve essere fatta con cura in quanto c'è la possibilità che qualche applicazione carichi dinamicamente contenuti sul MN proprio durante la nostra operazione di copia. Per ovviare a questo problema l'idea è quella di creare un pool di thread che hanno lo scopo di effettuare la copia senza perdita di dati. Ogni thread esegue le seguenti operazioni:

- Creazione copia della risorsa da MN a IN
- Creazione di una subscription per la risorsa appena copiata (ad eccezione delle content instance)
- Creazione della lista delle risorse figlie da creare
- Attivazione di un nuovo thread per la creazione di ogni risorsa figlia



L'idea è di esplorare così tutto l'albero per copiare tutto ciò che viene trovato. Se al momento dell'esplorazione una risorsa non era presente perché questa è stata caricata successivamente, la stessa verrà copiata al momento dell'arrivo della notifica. Infatti è da sottolineare come la subscription venga creata prima di chiedere l'elenco dei figli. In questo modo se una risorsa viene creata dopo la richiesta dei figli, la subscription già attiva permetterà di ricevere una notifica e all'arrivo della notifica verrà creato un thread che eseguirà le operazioni già descritte.

In realtà il singolo thread non esegue soltanto queste operazioni, infatti in caso di copia di una content instance si preoccupa di inviare una notifica ai servlet presenti sul server Tomcat per poter inviare eventuali aggiornamenti.



Infine per una gestione ancora più sicura della copia stessa, è stato previsto un thread che verrà lanciato periodicamente ma con un periodo molto lungo che si occuperà di controllare l'effettiva copia di tutte risorse.

6 Applicazione WEB

6.1 Interfaccia e Canvas

Per lo sviluppo dell'applicazione WEB sono stati utilizzati i classici HTML 5, CSS 3 e Javascript per il lato client e Java (nella forma dei servlet) per quanto riguarda la parte server.

6.2 Interfaccia e Canvas

La pagina web si presenta attraverso l'uso di alcune immagini e alcune frasi chiave che permettono di capire lo scopo del nostro sistema e le informazioni relative agli sviluppatori del sistema stesso. Per poter accedere all'applicazione è necessario registrarsi indicando le classiche informazioni personali ed è possibile decidere se creare un account amministratore o no (utile in fase di prototipazione). Una volta eseguito l'accesso l'applicazione risulta piuttosto semplice in quanto presenta soltanto una mappa Google sulla quale sarà possibile trovare alcuni marker in corrispondenza degli AE sparsi sul territorio.

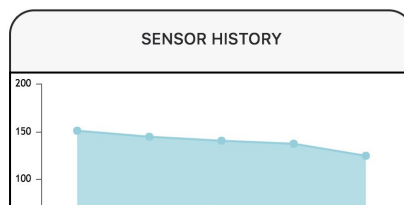
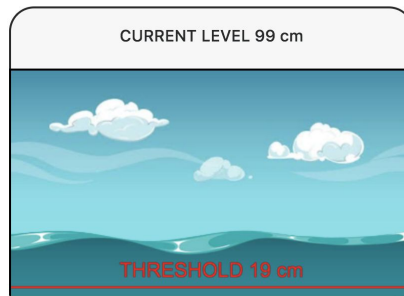
Una volta fatto click sul marker desiderato si aprirà una pagina in sovrimpressione in cui saranno raccolte tutte le informazioni disponibili, ovvero:

- Mappa del territorio con sensori e dighe presenti



- Dati relativi al sensore selezionato
 - Stato del sensore
 - Ultimo aggiornamento
 - Storico dei livelli di acqua
 - Possibilità di cambiare la soglia di pericolosità (solo per utenti amministratori)

Selected sensor: Sensor_8
Last update: 22/4/2018, 14:00:04



- Dati relativi alla diga selezionata
 - Stato della diga
 - Ultimo aggiornamento
 - Possibilità di cambiare aprire/chudere la diga (solo per utenti amministratori)

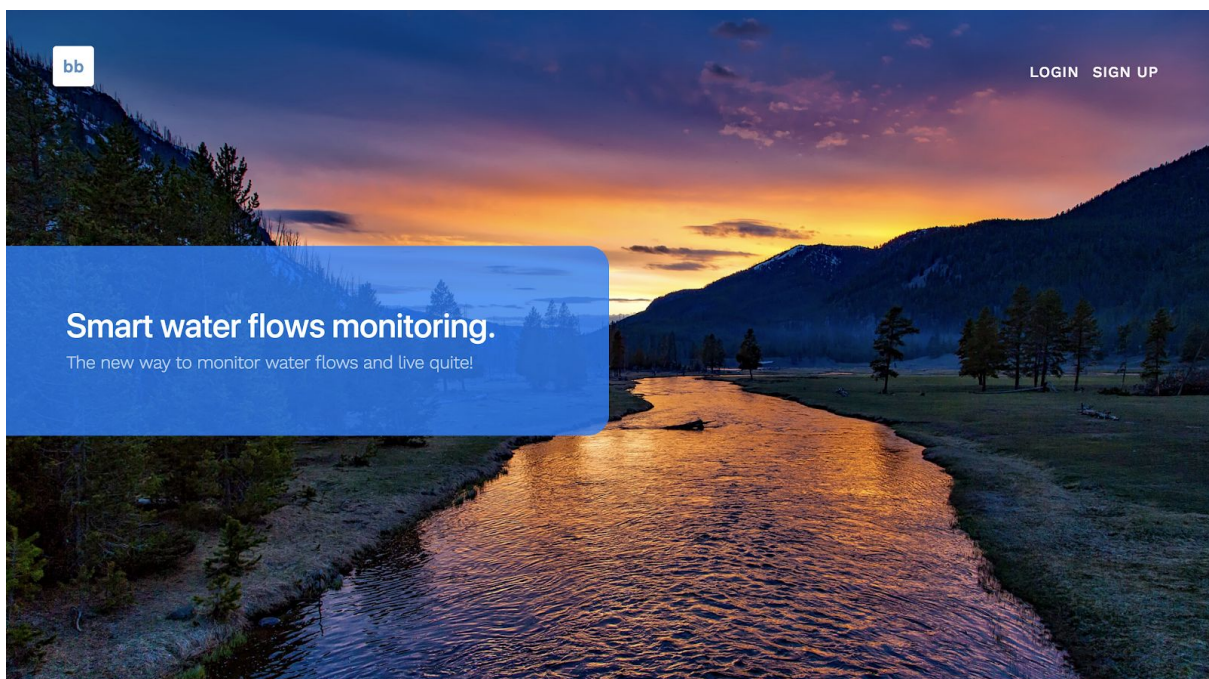
Selected dam: Dam_16
Last update: 22/4/2018, 14:03:08
Last state: open


CURRENT STATE: OPEN

CHANGE

Per quanto riguarda la mappa abbiamo deciso di optare per una rappresentazione non reale della realtà. In particolar modo avevamo bisogno di un modo semplice, intuitivo ma allo stesso tempo abbastanza reale per rappresentare lo stato del corso d'acqua. L'idea è stata quella di utilizzare alcune texture 16 pixel x 16 pixel prese dal popolare videogioco Pokemon. La grafica del videogioco scelto è perfetta in quanto unisce la semplicità di una rappresentazione 2D (ogni oggetto può trovarsi e/o muoversi solo in verticale o orizzontale) con la "realtà" di una rappresentazione tridimensionale. L'immagine finale è stata ottenuta andando a posizionare precise texture in precisi punti di una canvas. In particolar modo è stata creata una canvas 528 pixel x 528 pixel, che corrisponde ad un canvas con 33 x 33 textures. Per la generazione dell'immagine è stato utilizzato un trucco spesso utilizzato in fase di sviluppo di videogiochi con cui si crea un file di testo nel quale si inseriscono una serie di caratteri (33 righe composte da 33 caratteri ciascuna) che saranno associati a texture diverse.

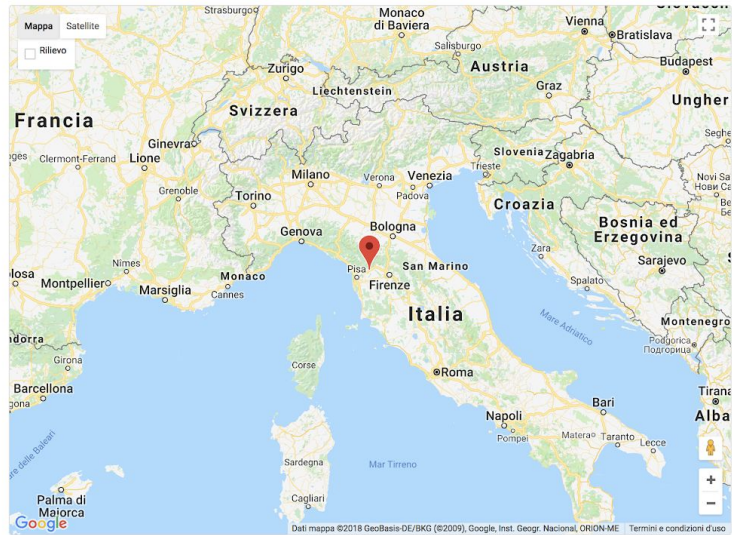
Nel suo complesso la pagina si presente attraverso le seguenti schermate:



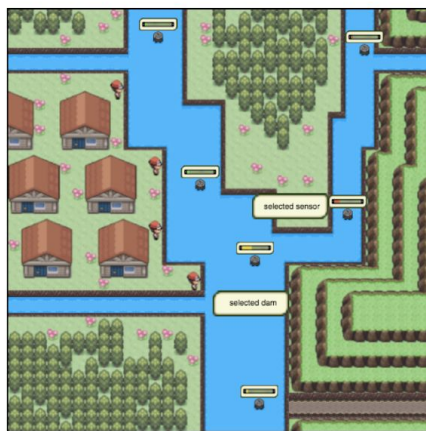


Silvio Bacci

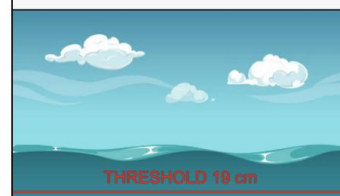
You are an administrator. You can act directly on our dams in order control the water flows.



csr-479396734 - Pesca_CAE185937199 (cnt-344809309)

**Alarm!** Critical situation. Please check Sensor_8**Selected sensor:** Sensor_8
Last update: 22/4/2018, 14:00:04

CURRENT LEVEL 99 cm



SENSOR HISTORY

200

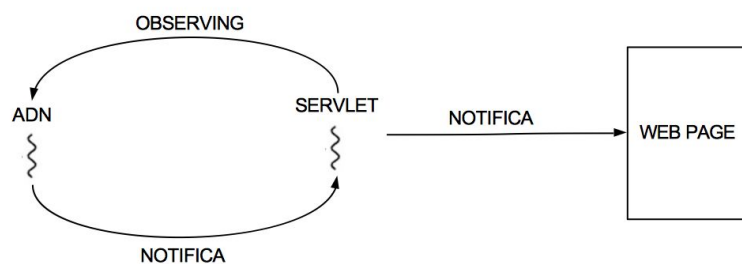
6.3 Tomcat server

Come già indicato la parte server è stata creata attraverso l'uso di alcuni servlet che permettono:

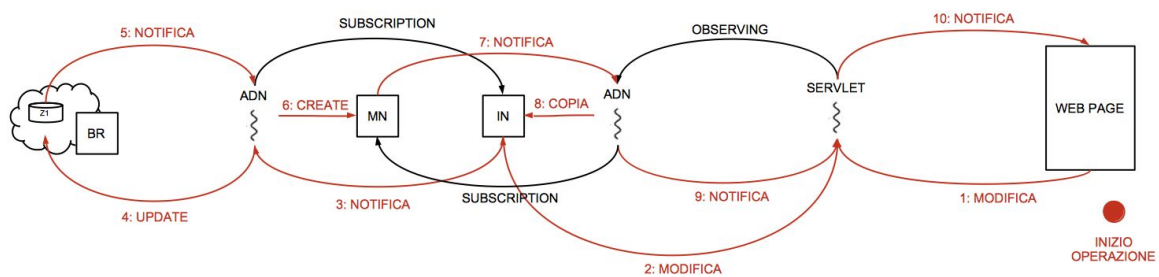
- Login
- Logout
- Registrazione
- Acquisizione dati per la mappa
- Acquisizione dati dei sensori
- Acquisizione dati delle dighe
- Modifica soglie sensori
- Modifica apertura/chiusura dighe

Per il supporto alla registrazione degli utenti si è fatto uso di un database MySQL in cui vengono salvati solo e soltanto le informazioni relative agli utenti. Tutte le informazioni relative ai sensori e agli attuatori saranno ovviamente memorizzate su IN e MN.

Ad eccezioni dei servlet legati al login dell'utente e a quelli legati alla modifica di risorse, gli altri servlet supportano SSE per poter inviare aggiornamenti dinamici. In questo caso l'idea è quella di avere una lista di oggetti Observer che osservano alcuni oggetti Observable controllati dai thread responsabili della copia su IN. In questo modo ogni volta che un utente si collega all'applicazione verrà creato un Observer che si aggiungerà alla lista degli observer per la risorsa richiesta. Non appena arriverà una nuova risorsa una notifica sarà inviata all'oggetto Observer che potrà inviare un aggiornamento al browser.

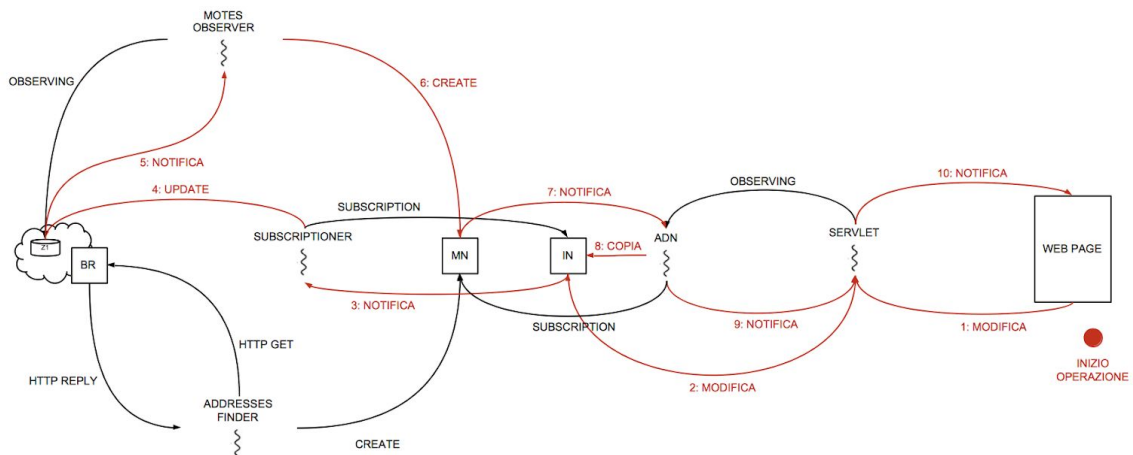


Per poter modificare una risorsa l'idea è quella di utilizzare alcune risorse specifiche chiamate CONTROL alle quali i vari AE si devono sottoscrivere. Così facendo ogni volta che un utente amministratore vuole modificare lo stato di una risorsa scriverà in questo contenitore. In questo modo una notifica arriverà all'ADN lato MN che invierà la modifica al sensore opportuno. Quando il sensore avrà cambiato il suo stato una notifica verrà inviata al thread che stava osservando il sensore (operazione di observing CoAP). Alla fine di questo ciclo l'observer CoAP farà un aggiornamento sul MN e questo aggiornamento sarà copiato sul IN. Nel momento in cui l'aggiornamento arriva all'IN si può dire che l'operazione di modifica di stato della risorsa è completato.



7 Conclusioni

Per concludere la struttura globale della nostra applicazione, nel caso specifico di una modifica da parte di un utente amministratore tramite l'applicazione web, sarà:



8 Manuale utente

Di seguito riportiamo un semplice manuale passo passo utilizzabile per testare il sistema da noi sviluppato. I seguenti passi fanno riferimento alla situazione in cui l'ambiente di simulazione Cooja, l'ADN lato Middle Node ed il Middle Node stesso risiedono su una macchina diversa dalla quella che ospita l'ADN lato Infrastructure Node e il Tomcat Server. Tra i prerequisiti software necessari per lanciare l'applicazione ci sono:

- Cooja
- Tomcat server

Da notare che i due progetti maven creati sono:

- Client (all'interno della cartella ADN_MN) che eseguirà il codice dell'ADN lato MN
- SWFM (all'interno della cartella ADN_IN) che eseguirà il codice dell'ADN lato IN

I passi da effettuare per poter eseguire l'applicazione sono:

1. Aprire Cooja
 - a. Aprire la simulazione *Water_Flow_Simulation*
 - b. Assicurarsi che l'opzione *run* dello script editor sia spuntata
2. Avviare TunSlip
3. Impostare i parametri corretti nei file di configurazione
 - a. config.ini del Middle Node
 - b. config.ini dell' Infrastructure Node
 - c. Conf.txt dell'ADN lato MN
 - d. config.txt dell'ADN lato IN
4. Avviare l'Infrastructure Node sulla seconda macchina
5. Avviare il Middle Node
6. Importare i progetti già esistenti (SWFM e Client) nella propria workspace di Eclipse
7. Eseguire i servlet presenti dentro il progetto SWFM su Tomcat Server
8. Eseguire il file GUI.java all'interno del progetto Client
9. Effettuare l'accesso all'indirizzo:

indirizzo_ip_tomcat_server:porta_tomcat/SWFM