UNIVERSITÀ DEGLI STUDI DI TRENTO

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea triennale in Informatica

Elaborato Finale

MONITORING DI RISORSE TRAMITE UNA WIRELESS SENSOR NETWORK: IL CASO DI STUDIO DI UNA SERVER FARM

Relatore: Dott. Paolo Giorgini Laureando: Matteo Chini

Anno Accademico 2006 2007

A tutti coloro che pensano
di aver contribuito ... grazie.

Indice

1 Introduzione

La temperature di celle frigo, la pressione di caldaie industriali o l'umidità dell'aria sono solo alcuni degli esempi per cui l'importanza di un buon metodo di rilevazione dei dati è un fattore importante ed in alcuni casi vitale. Inizialmente delegati all' uomo, che con strumenti e macchine campionava tali informazioni, le immagazzinava e successivamente le studiava, ora i sistemi di monitoraggio sono molteplici, automatizzati e possono cambiare a seconda dell'esigenza e delle risorse economiche disponibili. Grazie alla costante evoluzione delle componenti elettriche e informatiche, grazie all'abbattimento dei prezzi, la soluzione al problema del monitoring puoi contare su di una nuova arma: le Wireless Sensor Network (WSN). Questa particolare tecnologia è composta da piccoli nodi che eseguono tutte le operazioni di campionamento, processo e trasmissione dei dati. Monitoraggio di fattori ambientali, controllo dei mezzi pubblici o delle temperature di risorse a rischio sono alcuni dei campi in cui vengono create delle soluzioni mediante l'uso di reti sensoriali.

Le applicazioni, però, utilizzano degli standard sviluppati ad hoc e questo ne consegue una difficile integrazione tra hardware e software. Gli stessi campionamenti della rete, vengono trattati come un flusso grezzo di byte e non come un insieme di dati con un effettivo significato. Questo rende difficile l'accesso ad appplicazioni esterne e quindi la loro integrazione nel sistema che, inoltre, vengono spesso progettate per un ambito locale e non sono accessibili attraverso internet.

Scopo della tesi è quello di progettare e sviluppare un framework per l'accesso e la visualizzazione attraverso internet di dati prodotti da una Wireless Sensor Network. L'idea è quella di un framework attraverso il quale sia possibile progettare applicazioni facilmente integrabili ed estendibili con nuovi moduli che utilizzano i dati raccolti dalla WSN. Si vuole poter rendere sempre disponibili all'utente i dati campionati e quindi osservare lo stato della propria rete e, in caso venga rilevato un allarme, prendere le dovute contromisure.

In particolare ci si è focalizzati sul problema del monitoring delle temperature dei server installati su rack. L'idea è quella di tener sotto controllo la temperatura dei server in

maniera da evitare malfunzionamenti e il loro eventuale spegnimento.

La tesi è strutturata in quattro capitoli: nel **capitolo 1** vengono introdotte le diverse metodologie di monitoraggio, con i relativi vantaggi e svantaggi; vengono inoltre spiegate le Wireless Sensor Network, il loro generico funzionamento, la struttura del sensore ed alcuni dei sensori più comuni. Viene, poi, illustrato il sistema operativo TinyOs, uno standard nello svilupo delle reti di sensori, e le tipologie diverse di pacchetti che possono essere generate dai nodi. Infine viene descritto lo stato iniziale dello sviluppo del progetto, la rete sensoriale utilizzata e il metodo di interfacciamento all'esterno. Il **capitolo 2** disegna il quadro architetturale del sistema. Vengono descritte le parti coinvolte, i loro protocolli di comunicazione e la loro interazione. Innanzitutto viene descritta la WSN, l'interfacciamento con la FoxBoard e la creazione dei file XML. Si passa poi all'applicativo server e alla descrizione dei moduli che lo compongono. Si descrive poi il DataBase che crea lo storico dei dati. Infine la spiegazione dell'interfaccia utente mediante Web Server e l'utilizzo di pagine JSP e applet. Il **capitolo 3** illustra nei dettagli le scelte implementative di ogni singola componente: come vengono creati i file XML, come questi vengono gestiti dal SerialForwarderServer e come server, servlet e applet interagiscono tra loro. Infine, nel **capitolo 4**, viene mostrato il caso di studio implementato: il controllo della temperatura di una server farm. Viene analizzato nel dettaglio il problema e come l'architettura spiegata è stata implementata nello specifico. L'elaborato termina con le conclusioni dello studio eseguito, i risultati ottenuti, vantaggi e svantaggi dell'uso di tale tecnologia ed i possibili sviluppi futuri.

2 Monitoraggio di risorse tramite una WSN

Il seguente capitolo vuole illustrare l'evoluzione dei metodi di campionamento mostrando vantaggi e svantaggi di ognuno. Vengono introdotte tre metodologie tutt'oggi utilizzate. Viene poi descritta la tecnologia delle WSN, la struttura, i protocolli, l'hardware, i sistemi operativi più utilizzati e come questa soluzione si presta per il problema del monitoraggio di risorse. Successivamente viene illustrato il sistema operativo TinyOs e la struttura del pacchetto utilizzato. Infine viene descritto il progetto *VIGILIA*, punto di partenza del framework creato.

2.1 Le metodologie di rilevazione: brevi cenni storici

Con il termine monitoraggio si intende l'osservazione di valori rilevanti e variabili all'interno di un contesto specifico. Fin dall'antichità l'utilizzo delle pratiche di monitoraggio hanno avuto una grande importanza nella civiltà : venivano controllate le maree per il rischio di inondazioni, le precipitazioni in una determinata area per l'agricoltura. Gli stessi strumenti di misura seguirono uno sviluppo costante e tutt'oggi sono utilizzati quelli che possono essere considerati i figli dei metodi usati allora: basti pensare che nel 1597 Galileo Galilei creò il primo indicatore di temperatura, predecessore dei moderni termometri [1]. Una persona incaricata a raccogliere i dati eseguiva il lavoro con gli strumenti opportuni. Con l'evoluzione della civiltà, la conseguente evoluzione della tecnologia ha permesso l'installazione di sistemi adibiti alla raccolta delle rilevazioni: in campo ambientale sono state costruite installazioni per il monitoraggio come stazioni idriche per il controllo delle acque; a livello industriale l'installazione di termostati hanno permesso di verificare lo stato di caldaie e di prendere le dovute contromisure in caso la temperatura superasse il livello di guardia.

L'avvento dei sistemi informatici e delle reti ha portato numerosi cambiamenti anche al tipo di approccio al problema del monitoring: sistemi nati per rilevare e memorizzare i dati in modo locale, potevano essere non solo gestiti con la precisione di un calcolatore, ma i dati potevano essere disponibili a grandi distanze, permettendone la visualizzazione in un tempo relativamente basso.

L'evoluzione dell'informatica e dell'elettronica e l'analogo abbassamento di prezzo delle componenti elettroniche, ha portato all'adozione di sistemi di rilevazione sempre più piccoli [2], in grado di gestire rilevazione, elaborazione e trasmissione delle informazioni in modo autonomo. L'utilizzo di queste componenti, integrati all'uso di tecnologie radio per la trasmissione dei dati, ha portato i sistemi di monitoraggio alla loro naturale evoluzione nelle cosiddette Wireless Sensor Network [3].

2.2 Tipi di monitoraggio esistenti

Di seguito vengono descritte le tipologie più utilizzate per la rilevazione ed il campionamento, le quali hanno caratteristiche diverse e possono essere scelte in base a fattori come prezzo o problematica.

2.2.1 Manual data collection

Figura 1. Un addetto alla rilevazione manuale

Questa modalità di monitoraggio è la più antica e tutt'ora oggi ampliamente diffusa. E' caratterizzata da un operatore che con l'ausilio di apparecchiature specifiche anche digitali (es: termometro, barometro,etc.) raccoglie manualmente le informazioni, le controlla, le cataloga per poi successivamente analizzare i dati campionati (*Figura 1*). Sono numerosi i campi in cui viene tutt'ora utilizzata, poiché spesso non si ha la possibilità di installare apparecchiature costose, vuoi per i costi che possono essere proibitivi per le piccole aziende, vuoi per fattori legati all'ambiente di installazione. Un fattore importante in questo tipo di misurazioni è che non vi è nessun tipo di

collegamento al mondo esterno, se non quello dell'operatore. Non posso, quindi, da remoto poter conoscere lo stato delle risorse monitorate e solamente quando l'addetto alle misurazioni inserirà i dati in un sistema avrò questa possibilità. Un altro grave problema è dato dalla mancanza di una continua rilevazione. Il fatto che è l'uomo ad eseguire le misurazioni pone dei limiti fisici al refresh dei dati che altri tipi di apparecchi non hanno.

Vantaggi: facile da implementare, non necessita di manutenzione particolare.

Svantaggi: richiede ore di lavoro da parte di un operatore, si possono commettere errori di misurazione o trascrizione, non sempre vi è la costante presenza dell'operatore.

2.2.2 Chart Recorders

Figura 2. Monitoraggio di più parametri tramite Chart Recorder

Strumento di tipo elettromeccanico che permette di registrare un particolare input elettrico su di un nastro di carta tramite la scrittura per mezzo di un ago. Vi è la possibilità di monitorare anche più risorse contemporaneamente con l'ausilio di più aghi di colori differenti (*Figura 2*). A differenza del **Manual data collection**, la presenza del fattore umano è limitata solamente alla manutenzione: questo fatto permette di avere una misurazione continua nel tempo. Come ogni altro strumento meccanico, il chart recorder è soggetto a guasti, malfunzionamenti e manutenzione. Si possono creare così problemi nel caso questa non venga eseguita in tempo. Se, per esempio, il nastro dovesse finire, avverrebbe una perdita di dati. Un'altra complicazione legata all'uso di questi macchinari è il tipo di memorizzazione utilizzata, il nastro di carta, che occupa spazio fisico non indifferente e non permette un metodo veloce di accesso ai dati. Normalmente questa metodologia non viene connessa ad altro e necessita quindi di una

persona incaricata alla lettura dei dati. Un' altro difficoltà non trascurabile dovuta all'utilizzo di queste apparecchiature è il costo che, a seconda del numero di canali monitorati e del grado di precisione può arrivare a diverse migliaia di euro.

Vantaggi: abbastanza preciso, facilità di installazione.

Svantaggi: manutenzione del nastro, archiviazione dei dati inefficace,costoso, non connesso con l'esterno.

2.2.3 Centralized Monitoring Systems (CM)

Figura 3. Monitor per il monitoraggio delle temperature di celle frigo

Questi sistemi, più complessi di quelli visti finora, utilizzano un sistema centralizzato per l'osservazione. Una rete di sensori viene applicata sulle risorse o nell'intorno del fenomeno da monitorare che viene tenuto sotto costante osservazione (*Figura 3*). La macchina centrale è adibita alla raccolta delle informazioni rilevate e le memorizza sulla macchina. A differenza delle metodologie precedenti, questi sistemi vengono gestiti tramite una rete e grazie a questo l'operatore che risiede sulla macchina centrale può costantemente visualizzare lo stato di ogni sensore in tempo reale. Grazie a questo sistema è anche possibile costruire un sistema di allarmi, generati in caso i sensori rilevino un dato preoccupante o fuori norma. L'addetto verrà informato tramite un messaggio a video o un allarme sonoro e potrà prendere le contromisure del caso. Questi sistemi hanno però numerosi contro che rendono l'utilizzo dei CMS una scelta

impossibile in alcuni casi. Primo fra tutti è il fatto che l'installazione deve essere fatta da tecnici specializzati che devono cablare la rete dei costosi sensori, facendo salire il costo in modo esorbitante e impedendo un sistema esteso. Vi sono poi i problemi dovuti al fatto dell'utilizzo di un sistema centralizzato che, in caso di problemi legati alla rete o alla macchina centrale, non permetterebbe un buon monitoraggio.

Vantaggi: gestione in tempo reale di allarmi, stato in tempo reale delle macchine

Svantaggi: hardware molto costoso, il tipo di installazione è impossibile in ambienti esterni molto estesi.

2.3 Le WSN come soluzione per il monitoring

Il seguente paragrafo vuole illustrare la tecnologia delle Wireless Sensor Network e come possono essere utilizzate per risolvere il problema del monitoring di risorse. Dopo un'introduzione generale sulla struttura della rete e del singolo nodo che la compone si vuole entrare nel dettaglio di TinyOs: il primo ed il più utilizzato sistema operativo per reti sensoriali. In particolare si vuole illustare la struttura dei pacchetti generati dai sensori, a seconda dellìapplicazione utilizzata, ed il flusso dei bytes prodotti dai nodi.

2.3.1 Caratteristiche generali

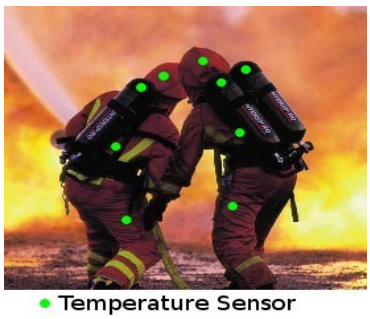

Figura 4. Sensori applicati agli abiti di due pompieri

Una WSN (Wireless Sensor Network) è una rete senza fili composta da elementi chiamati nodi. Questi apparecchi composti da una una parte adibita alla rilevazione (sensore), una all'elaborazione (microcontroller) ed un'altra alla trasmissione

(trasmittente), sono completamente indipendenti e per questo capaci autonomamente di rilevare, processare e trasmettere i dati. Nata nella seconda metà degli anni '90 da un consorzio di università americane, successivamente la DARPA (Defense Advanced Research Projects Agency**)** per la sorveglianza militare ha avuto fin da subito utilizzi nei campi più disparati sia civili che industriali [2]. Un vantaggio notevole che questa tecnologia offre rispetto ai metodi presentati precedentemente è l'utilizzo di metodi di trasmissione wireless che permettono l'installazione di sensori in posti prima impossibili poiché il cablaggio di una rete di sensori non era realizzabile. Un'esempio rilevante è quello del monitoring dell'eruzione del vulcano Tingurahua condotto dai ricercatori Geoffrey Wemer-Allen, Jeff Johnson, Mario Ruid, Jonathan Lees e Matt Welsh[1]. Grazie alla capillarità della rete utilizzata riuscirono ad avere una enorme quantità di dati da studiare, che prima non potevano essere ottenuti [4].

Grazie a questa tecnologia di rilevazione ora è possibile monitorare anche risorse non statiche. Prendiamo, per esempio il caso in cui il problema è quello del monitoraggio di una squadra di pompieri. Spesso vengono coinvolti in incendi e, a causa della temperatura elevata, sono sottoposti a stress fisico elevato. Per questo uno strumento per rilevare la temperatura corporea degli uomini e delle attrezzature vitali che indossano è necessario. Tramite una serie di nodi installati sui punti chiave, è possibile monitorare la situazione ed avere tutto sotto controllo (*Figura 4*).

A differenza delle altre metodologie, poi, questo sistema è facilmente estendibile [5] tramite l'inserimento di nuovi nodi e non necessitano, quindi, di ulteriori cablature o installazioni. Il costo, poi, è ridottissimo e alla portata anche di piccole imprese.

1 Tutti i dettagli dell'esperimento sono disponibili al sito http://www.eecs.harvard.edu/~mdw/proj/volcano/

2.3.2 Struttura e comunicazione in una WSN

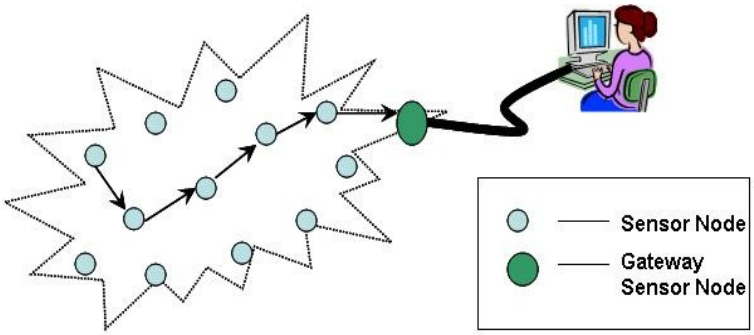

Figura 5. Struttura generale di una WSN

Una rete sensoriale è composta totalmente da Sensor Node (o Mote): elementi indipendenti capaci di eseguire campionamenti, processarli in forma digitale e trasmetterli all'interno della rete. Le tipologie di rete possono essere le più disparate: a stella, ad anello o di tipo mesh. Quest'ultima, la più utilizzata, è caratterizzata dal fatto che ogni nodo provvede all'instradamento dei pacchetti non diretti a lui. Esistono due tipi di nodi distinti: il primo è il Sensor Node, adibito alla rilevazione e all'invio dei dati; il secondo è il nodo Gateway: il ponte di comunicazione tra la WSN e il resto del mondo. Tutti i pacchetti che contengono le informazioni utili sui campionamenti, vengono instradati in modo da raggiungere il nodo Gateway e, quindi, essere disponibili all'esterno (*Figura 5*). Ogni nodo tiene in memoria una tabella che contiene le informazioni sui nodi che può raggiungere, la loro distanza dal nodo gateway e la qualità del collegamento. In base a queste informazioni ogni nodo ha sempre chiaro il nodo a cui dovrà instradare i pacchetti per raggiungere il nodo Gateway. Periodicamente, queste tabelle vengono aggiornate mediante lo scambio di informazioni tra i nodi.

Una struttura di questo tipo permette un ulteriore vantaggio: l'aggiunta di un nodo può essere eseguito in modo facile e veloce e non richiede nessun difficile e costoso cablaggio.

2.3.3 Hardware: il sensor node

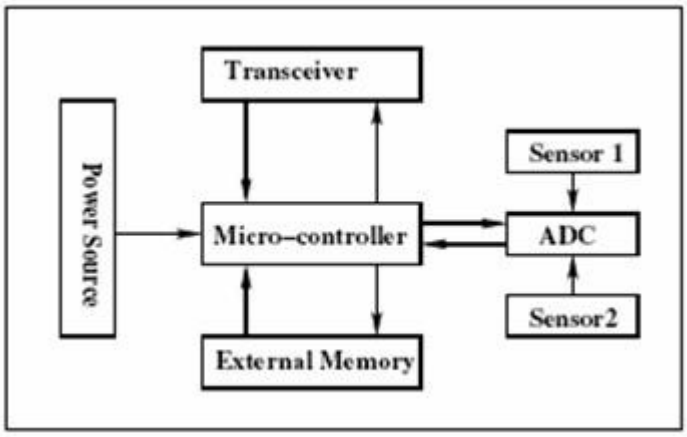

Figura 6. Struttura HW di un nodo sensoriale

Anche conosciuto con il nome di mote, la progettazione di un sensore ha come obiettivi principali quello di essere piccolo ed economico[6]. Con queste prerogative sono stati commercializzati numerosi tipi di sensori[2] diversi in fatto di grandezza, memoria o tipo di programmazione. La struttura generale di un sensore può essere schematizzata dalla *Figura 6.*

Le componenti principali di un nodo sono:

- **Microcontroller**: gestisce le operazioni del sensore, i dati ricevuti e controlla tutta la componentistica. E' la scelta migliore poiché questo componente ha la caratteristica di consumare relativamente poco
- **Transceiver**: gestisce la comunicazione radio. Esistono numerosi tipi che possono essere raggruppati in tre tipologie: radio, Laser e Infrarossi. Normalmente la scelta ricade sulla comunicazione via radio (standard IEEE 802.15), capace di passare anche attraverso ad ostacoli, cosa non permessa dai sistemi ottici.
- **External Memory**: ne esistono di due tipi: la prima sul chip del microcontroller, poca ma a basso consumo, la seconda invece di tipo FLASH memory, di maggior capienza ma normalmente poco utilizzata a causa del consumo elevato. La memoria è forse il fattore più rilevante per quanto riguarda le applicazioni che utilizzano le WSN e poiché determina i dati che il sensore può

2 Una lista dettagliata è disponibile sul sito http://www.btnode.ethz.ch/Projects/SensorNetworkMuseum

immagazzinare.

- **Power Source**: il consumo all'interno di un nodo è dovuto alle tre seguenti cause: campionamento, comunicazione e processamento dei dati. Il maggior consumo è dato dalla componentistica trasmissiva che, a seconda della sorgente e della programmazione del sensore, determina l'autonomia del sensore. L'utilizzo di batterie è la pratica più utilizzata sia ricaricabili che non ricaricabili. Attualmente vengono sviluppati anche sensori in grado di autoricarsi tramite energia solare, riducendo la manutenzione all'interno della rete.
- **Sensore:** è una periferica hardware che può risiedere direttamente sul nodo o può essere collegata dall'esterno. Rileva una misurazione fisica in un segnale elettrico misurabile da parte del microcontroller. I sensori più comuni sono quelli di temperatura, integrati spesso sul nodo, ma ne esistono numerasi altri a seconda del tipo di misurazione da effetuare: elettromagnetici, meccanici, acustici, di movimento sono solo alcuni esempi.

2.3.4 Sistemi operativi

Come già introdotto in precedenza, la vera difficoltà dell'utilizzo dei sensori è quella di disporre di risorse limitate: memoria, autonomia e un'esigua velocità di calcolo devono essere prese in considerazione per l'utilizzo di un sistema operativo per i nodi.Prerogativa asoluta per la costruzione di un SO per reti di sensori è quella di essere leggero e poco complesso. Il fatto che non vi è, però, nessun tipo di interfaccia e interattività dall'utente può comunque aiutare.
Numerosi sono i sistemi creati su queste basi: Contiki, Retos, BtNut e Mantis sono solo alcuni di SO creati per lavorare su hardware con limitate risorse come i nodi.
Il primo sistema operativo nato appositamente per i sensori è, però TinyOs.

2.3.5 Tiny Os e i pacchetti generati dalla WSN

Nato nel 1999 a Berkeley all'interno del progetto DARPA NEST (Defense Advanced Research Projects Agency Network Embedded Systems Technology) come sistema

operativo per sensori militari, cresce in parallelo allo sviluppo hardware dei sensori, questi progettati all'interno della stessa università (Berkeley Mote). Ha raggiunto ora la release 2.0.2 e viene costantemente aggiornato[3].

Il sistema [7] è ottimizzato per lavorare in condizioni di limitazione di memoria, fattore già descritto come fondamentale per l'esecuzione dei programmi, e è stato scritto per lavorare ad eventi. I programmi vengono scritti utilizzando il nesC, un dialetto del C per e possono essere installati su una grande varietà di sensori: BTNode, Imote, Mica e Telos solo per citarne alcuni.

Cambiando la programmazione del nodo, i pacchetti generati possono essere di diverse tipologie a seconda delle funzioni utilizzate. Il pacchetto può essere visto come una sorta di scatola che contiene differenti dati a seconda del protocollo e dell'applicazione che viene utilizzata dal sensore per eseguire le rilevazioni.

Figura 7. La struttura base del pacchetto Tiny Os

Come si può vedere dalla *Figura 7*, c'è un guscio esterno comune che definisce il pacchetto TinyOs, mentre la struttura di quello più interna è definita dal tipo di applicazione, la quale varierà a seconda della programmazione del nodo. La *Tabella 1* mostra la posizione dei campi presenti nel pacchetto Tiny Os, possibili valori e significato [8].

3 Il sito ufficiale del progetto è www.tinyos.net

Tabella 1		
#Byte	***Campo***	***Descrizione***
0	Data Length	La lunghezza in byte del Payload.
1-2	Frame Control Field	Frame Control Field (guardare standard 802.15.4)
3	Dat Sequence Number	Numero di sequenza di data link
4-5	Dest PAN Address	Identificatore di destinazione PAN
6-7	Message Address	Può assumere uno dei seguenti valori: • Broadcast (**0xFFFF**): messaggio a tutti i nodi. • UART (**0x007E**): messaggio da un nodo al gateway. Tutti i messaggi in entrata hanno questo valore. • Indirizzo del nodo: l' ID univoco del nodo che riceverà il messaggio.
8	Message Type	L'identificatore del tipo di messaggio contenuto in Payload Data. Ogni applicazione diversa avrà il suo identificatore univoco. Ecco aluni degli identificatori conosciuti : • XUART = **0x00** • MHOP DEBUG = **0x03** • SURGE MSG = **0x11** • XSENSOR = **0x32** • XMULTIHOP = **0x33** • MULTIHOP MSG = **0xFA** • **....**
9	Group ID	Identificatore univoco associato al gruppo di sensori. Il valore di default è 125 (0x7D). Solamente i sensori che appartengono allo stesso gruppo possono comunicare tra loro.

10..n-2	Payload Data	I messaggio vero e proprio specificato dal tipo nel secondo byte.
n-1, n	CRC	Due byte che assicurano l'integrità dell'intero pacchetto. Il CRC conmprende tutto il pacchetto ad eccezione dei byte di sincronizzazione (all'inizio ed alla fine).

Il Payload può cambiare a seconda del tipo di messaggio. Sono stati sviluppati numerose tipologie di messaggi a seconda del tipo di applicazione utilizzata. Alcune, riconosciute come standard, sono state riportate nella tabella, mentre molte altre vengono sviluppate di continuo.

Un speciale tipo di messaggio è quello identicato come MULTIHOP MESSAGE: il messaggio multihop tipico delle reti sensoriali mesh in cui i sensori instradano i pacchetti dei vicini verso il nodo gateway. Viene quindi creato un nuovo tipo di incapsulamento del messaggio (*Figura 8*).

Figura 8. La struttura del pacchetto multihop

Il pacchetto multihop contiene tutte le informazioni riguardanti il percorso eseguito dalla sorgente alla destinazione e viene configurata dal programmatore ad hoc per il tipo di applicazione. La *Tabella 2* illustra la struttura del pacchetto multihop così come è stata usata in laboratorio.

Tabella 2		
#Byte	***Campo***	***Descrizione***
0-1	Source Address	ID del nodo che ha eseguito il forwarding dei dati
2-3	Origin Address	ID del nodo che ha originato il pacchetto
4-5	Sequential Number	Numero di sequenza di chi ha eseguito il forwarding
6-7	Origin Sequential Number	Numero di sequenza di chi ha generato il pacchetto
8	Time to live	Tempo di vita del pacchetto
9	id	non utilizzato
10	Rack ID	Il Rack a cui appartiene il nodo che ha generato il pacchetto
11-n	Data	I dati dell'applicazione che utilizza il protocollo multihop

Per quanto riguarda il valore del campo **Data**, questo è variabile a seconda del tipo di messaggio. L'informazione principale resta, comunque, la lettura da parte dei sensori del dato monitorato e convertito in digitale. Considerando sempre l'esempio del caso di studio, il campo **Data** è assume i valori presenti in *Tabella 3.*

Tabella 3		
#Byte	***Campo***	***Descrizione***
0-3	Sequence Number	Numero di sequenza del nodo che ha originato il pacchetto
4-5	Internal Voltage	Voltaggio delle batterie del nodo (milivolt)
6-7	Internal Temperature	Temperatura del sensore (valore da convertire in °C)
8-9	Parent	ID del parent node del nodo che ha generato il pacchetto
10	neighborsize	Lunghezza della lista dei vicini
11	retransmissions	RSSI
12-n	neighbors	lista contenente gli Id dei vicini

n+1-k	quality	lista qualità dei collegamenti con i vicini

2.4 Il progetto *VIGILIA*

Il progetto VIGILIA (Very Implantable Gear Inspecting Local Input Array) è nato all'interno dei laboratori di ArsLogica[4], ditta dedita alla ricerca e allo sviluppo di soluzioni innovative in campo informatico ed elettronico. Grazie ai continui sviluppi nel campo delle comunicazioni senza fili, dell'abbassamento del prezzo di componenti elettrici e dei sensor node, è ormai possibile creare soluzioni economiche che utilizzano le WSN come soluzione. Una recente stima[5] mostra che il mercato delle reti sensoriali raggiungerà nel 2010 un valore di 7 miliardi di dollari e che nel 2012 vi saranno 14 milioni di piccole WSN casalinghe nel mondo. Questa stima ci fa intuire la grandezza che questo tipo di tecnologia raggiungerà in breve tempo. Lo scopo del progetto è quello di creare un'architettura stratificata su più livelli che utilizzi le WSN come soluzione a problemi di monitoring. I livelli presenti sono i seguenti:

1) **WSN e gateway**: lo strato più basso dell' architettura è composto dalla rete sensoriale e dal gateway adibito alla raccolta dei pacchetti
2) **WSN server**: server che gestisce l'arrivo delle informazioni dal gateway, riconosce gli eventi occorsi all'interno della WSN e comunica con lo strato 3.
3) **WSN application**: lo strato più alto è quello delle applicazioni che utilizzano gli eventi rilevati dallo strato 2. Ogni applicazione avrà un proprio template a seconda del caso.

Quest'architettura stratificata ha l'obbiettivo di rendere facile la vita ai programmatori sviluppando soluzioni ad hoc, limitandosi esclusivamente al livello specifico, indipendente dagli altri. Da questo progetto, sono nati successivamente altri sottoprogetti, tra cui *Status*, nato appositamente per utilizzare le WSN per il controllo delle server farm (vedere cap. 4).

4 www.arslogica.it
5 fonte www.onworld.com

3 Architettura generale del sistema realizzato

In questo capitolo verranno mostrati gli elementi che compongono il framework e illustrate le scelte architetturali che gli stanno alla base. Innanzitutto verrà presentata la struttura del sistema suddivisa in tre livelli: per ognun livello saranno mostrate le scelte architetturali fatte e quali sono i vantaggi del loro utilizzo. Successivamente verranno presentati i dettagli dei vari livelli: lo strato più basso, quello della WSN e l'utilizzo di file XML per il trasporto delle informazioni; il server che identifica gli eventi ed i moduli che li gesticono; i moduli che utilizzano gli eventi del server ed infine la presentazione dei dati all'utente finale.

3.1 Architettura del sistema

Lo schema proposto ricalca quello introdotto nel paragrafo 1.4 e può essere schematizzato dalla *Figura 9*.

Identifichiamo la **Wireless Sensor Network** come la fonte dei dati da utilizzare. E' caratterizzata da un numero di nodi variabile grande a piacere, programmati tramite il s.o. **TinyOs**. La tipologia della rete generalmente è di tipo mesh e, quindi, il tipo di pacchetto generato dal singolo nodo utilizzerà la struttura multihop vista nel paragrafo 1.3.5. I sensori in questione eseguono campionamenti regolari e mandano i pacchetti al nodo **gateway**, i quali, però, non possono essere utilizzati in maniera diretta poiché, a seconda dell'applicazione utilizzata sui sensori, il significato dei singoli byte può variare. Il nodo gateway è collegato ad un sistema embedded: la **FoxBoard**[6]. Compito di tale sistema è quello di trasformare il flusso grezzo di byte in un documento con un significato ben preciso che può essere letto dal server. Tramite l'utilizzo di documenti **XML** si vuole poter generare dei documenti strutturati, in modo da essere in grado di

6 La documentazione è al sito www.acmesystem.it da cui è anche possibile ordinarla

capire il significato di ogni byte del pacchetto [9]. Ci dovrà, poi, essere un'applicativo adibito alla gestione dei documenti tramite lettura, validazione e utilizzo dei dati: il **SerialForwarderServer**. La struttura che si vuole ottenere deve avere la capacità di potersi espandere in futuro permettendo a chi la usa di inserire nuove funzioni. E' necessario, quindi, l'utilizzo di un'applicazione server struttura in moduli separati. I dati contenuti nel file XML dovranno essere inseriti in un database che conterrà tutto lo storico della raccolta di dati della WSN. Un utente esterno dovrà poter visualizzare in tempo reale lo stato della rete sensoriale senza passare per il database, in modo da alleggerire il carico su questo che dovrà gestire già le numerose scritture da parte del server.

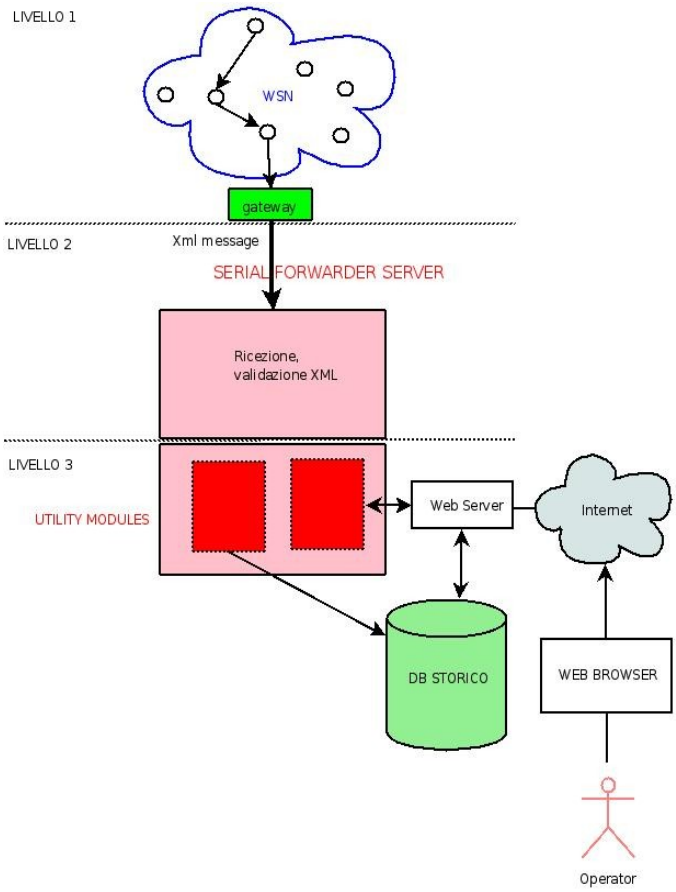

Figura 9. Il framework suddiviso in tre livelli

Il framework realizzato è diviso su 3 livelli (*Figura 9*):

livello 1: ***Rete di sensori (WSN)*** La parte hardware dell'architettura, i nodi con i sensori specifici vengono applicati sulla risorsa da monitorare. I pacchetti all' interno della rete verranno instradati fino a raggiungere il gateway che si occuperà della traduzione da flusso grezzo di byte a XML.

livello 2: ***SerialForwarderServer (SFS)*** è l'applicazione adibita all'analisi e all'utilizzo dei dati in arrivo dalla rete di sensori sotto forma di file XML. Deve raccogliere le informazioni da questo formato di file, pensare alla convalidazione e passare il documento ai moduli sottostanti. Questa parte del server è standard nell'architettura.

livello 3: la parte variabile del sistema. Ogni elemento di questo livello deve essere configurato ad hoc per il tipo di monitoraggio eseguito. In particolare i ***moduli del server*** i quali prelevano i documenti validati, estrapolano i dati e li utilizzano per i loro scopi. In questo livello viene inserito anche il ***Database*** con la funzione di storico per tutte le misurazioni e per gli allarmi catalogati. Ogni volta che i dati giungono ai moduli, uno di questi li inserisce nelle relative tabelle. Infine sarà presente un apposito ***Web Server*** adibito alla presentazione dei dati all'utente .

3.2 WSN: aquisizione dei dati e generazione XML

Ogni nodo, applicato alla risorsa da monitorare, esegue un campionamento a cadenza regolare, intervallo temporale che può essere impostato dal programmatore a seconda dei casi. Viene creato, quindi un pacchetto che segue la struttura del capitolo 1.3.5. Questo, instradato all'interno della rete, raggiunge il nodo Gateway, ed al sistema ad esso collegato. Su questo sistema è presente un software specializzato nelle seguenti operazioni:

- controllo dell'integrità del pacchetto
- accodamento del pacchetto in una coda FIFO
- costruzione e instradamento del file XML costruito in base al tipo di messaggio ricevuto.

Il controllo di integrità del pacchetto viene eseguita calcolando il CRC (Cyclic Redundancy Check), una pratica utilizzata nel caso di Wireless Sensor Network presenza di rumori di fondo. In caso fosse diverso, verrebbe mandato un messaggio al nodo mittente chiedendo la ritrasmissione dei dati. Se, invece, il riscontro fosse positivo, l'applicazione accoderà il pacchetto nella coda contenente i pacchetti da gestire.

L'ultima operazione è quella di costruzione del file XML e dell'instradamento verso l'**SFS**. L'applicativo costruisce, quindi, il file (la politica di creazione viene spigata successivamente nel capitolo 3.1) e passa poi all'invio al SerialForwarderServer. Terminato questo compito, passa poi all'analisi del pacchetto successivo

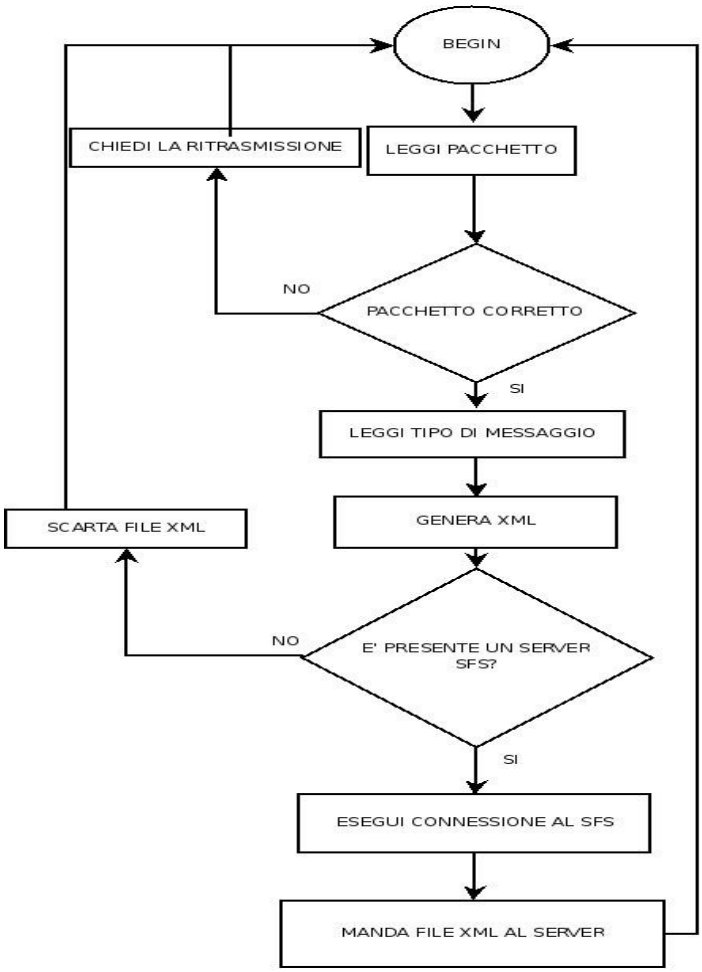

Figura 10. Operazioni eseguite dal serialforwarder all'arrivo di un pacchetto

La *Figura 10* rappresenta il flusso del programma dall'acquisizione dei dati (BEGIN) alla generazione dell'XML (ultimo box in basso). Inizialmente il programma attenderà l'arrivo del pacchetto da parte dell WSN. Una volta giunto, verrà controllata la correttezza dei byte (CRC) e, in caso fosse errato, verrà chiesta la ritrasmissione dal nodo sorgente. In caso, invece, il pacchetto fosse effettivamente corretto, verrà generato il file XML. Il programma verificherà, poi, la presenza di un server adibito alla raccolta dei messaggi: in caso vi sia, verrà instaurata una connessione e instradato il file XML; in caso contrario il file verrà scartato. Non è possibile memorizzare i file, poiché l'applicativo è in esecuzione su di un calcolatore con limitate risorse e, quindi, se la connessione non è presente la memoria va liberata e va scartato.

3.3 SerialForwarderServer e moduli

Una volta inviato il file XML contenente le informazioni rilevanti del campionamento, questo viene ricevuto dall'applicativo **SerialForwarderServer (SFS)**, il programma che avrà la funzione principale di gestire i dati tramite l'utilizzo di diversi moduli, ognuno con la propria funzione. L'utilizzo, quindi , della modularità permette al server di essere, un domani esteso con nuove funzioni che sfruttano i dati legati alla WSN. Rimane comunque una struttura verticale in cui i moduli potranno essere aggiunti.

Tale struttura è composta da:

- **Listener**: adibito alla connessione con il serialforwarder e al salvataggio del file XML
- **Validator**: con il compito di validare il file XML e, in caso fosse valido, aggiungerlo alle code dei moduli sottostanti.
- **Moduli**: thread separati ognuno con la propria coda FIFO di documenti XML convalidati. Eseguono le operazioni con i dati come l'inserimento in un database

o l'invio ad un web server.

Di seguito, la *Figura 11* mostra la struttura dell'SFS con l'integrazione dei moduli adibiti all'inserimento dei dati nel database e quello che gestisce il trasferimento dei dati al Web Server per la visualizzazione tramite web browser.

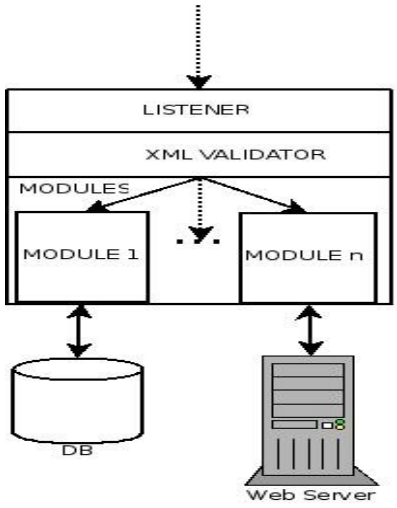

Figura 11. SFS e passaggio dei messaggi ai moduli

Il livello superiore della classe è presente il **Listener**, la parte del server che gestisce la connessione al serialforwarder. Essenzialmente esegue due operazioni: instaura una connessione con il programma serialforwarder e memorizza il file XML in arrivo dal client. Se le operazioni sopracitate vanno a buon fine, il controllo passa al modulo di validazione **Validator** che esegue le seguenti operazioni: ricezione del documento XML, validazione e accodamento nelle code per ogni modulo sottostante.

Le code sono delle liste di elementi formati da:

- il documento XML
- un flag che indica se il documento è valido o meno

A seconda del numero di moduli presenti nel server, verranno create le relative code, inizialmente vuote, associate ad ognuno di essi. Ogni qual volta un documento viene validato o no, questo viene messo in ognuna delle code dei moduli. In maniera autonoma, quindi, i documenti verranno gestiti a seconda dell'uso necessario. Prelevato un documento valido dalla coda (o scartato se non valido), il modulo leggerà le

informazioni necessarie e le utilizzerà. Nella *Figura 11* vengono presentati due esempi di moduli: *module 1* rappresenta la parte del server che utilizza le informazioni del documento per inserirle nel Database storico. *Module n*, invece, preleva i dati per inoltrarli al web server (entrambi i moduli sono stati implementati e illustrati nel dettaglio nel paragrafo 3.3).

3.4 Database per la memorizzazione dei dati

Il Database dei dati è la struttura centrale dell'architettura a cui si potranno interfacciare i moduli del server per l'inserimento dei dati e il web server per la richiesta di informazioni da mostrare all'operatore. Vanno quindi definite le entità che lo compongono e le relazioni che le uniscono. Innanzitutto possono esserci un numero arbitrario di reti monitorate, ognuna con un tipo diverso di monitoraggio ed identificate da un ID univoco. Ogni rete potrà essere attivata o disattivata dall'operatore. Potrà, inoltre, avere una sessione di raccolta dati attiva e un numero imprecisato di sesioni precedenti. La singola rete sarà, poi, composta da un insieme di nodi adibiti al campionamento, anch'essi etichettati da un'univoco ID. Ogni singola misurazione dei nodi, invece, dovrà essere catalogata in modo da fotografare lo stato della rete la quale potrà variare durante la sessione. Dovranno, quindi, essere memorizzati il nodo, l'eventuale rack a cui appartiene, data e tempo della misurazione ed infine il valore misurato. Nel caso una misurazione dovesse superare un valore massimo, l'utente potrà voler memorizzare il motivo dell'errore.

Le entità presenti all'interno della struttura sono le seguenti:

Reti: rappresenta la rete su cui viene eseguito il monitoring. Ogni rete ha un codice univoco che la identifica.

Sessioni: una sessione rappresenta l'inizio di un campionamento che l'utente decide di eseguire. Ogni sessione potrà iniziare o finire a piacere dell'utente. Conterrà inoltra il valore di soglia dopo cui verranno generati gli allarmi. Dovrà essere presente al massimo un'unica sessione attiva associata ad una rete.

nodi: è la lista completa dei nodi che eseguono il campionamento. Ogni nodo viene

identificato univocamento all'interno di una rete da un proprio ID ed ogni nodo invia all'interno del pacchetto anche l'informazione della durata della carica.

Measure: l'entità centrale del sistema in cui risiedono tutti i rilevamenti fatti dai nodi. Ogni misurazione è identificata dall nodo che ha eseguito il campionamento,dalla rete, la data e l'ora ed, infine, la misurazione del parametro (nel mio caso di studio un'altro parametro identificava la misurazione: il rack).

Allarmi: rappresenta i tipi di allarmi generati.

Allarmi generati: entità che include tutti gli allarmi gestiti dall'utente a cui è stata anche associata una tipologia contenuta in **Allarmi**.

Di seguito (*Figura 12*) viene presentato il modello entity-relationship utilizzato:

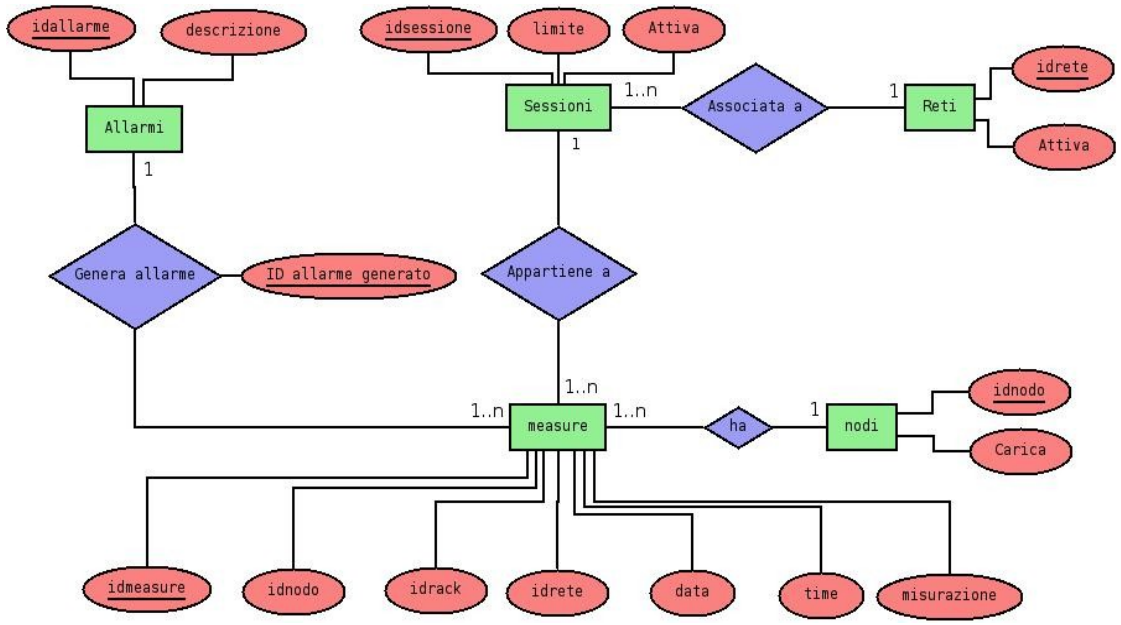

Figura 12. ER del database utilizzato per il framework

Dal disegno si notano tutte le entità (box verdi) descritte in precedenza, i loro attributi (ovali rossi) e le relazioni (rombi blu) che le collegano. L'entità fondamentale è l'entità **measure**, che conterrà ogni singolo campionamento eseguito dai nodi della rete. Ogni elemento dell'entità è identificato univocamente da un numero crescente (**idmeasure**), e contiene tutte le informazioni della misurazione come un'istantanea dello stato della

rete: il nodo che ha eseguito il campionamento (**idnodo**), in che rack è stato eseguito (**idrack**), in che rete è stato fatto il campionamento (**idrete**), quando (**data** e **time**) ed infine il dato vero e proprio (**misurazione**). L'entità appena descritta ha una relazione **ha** 1 a n con l'entità **nodi**, adibita alla memorizzazione dei nodi presenti nel sistema. Ad ogni nodo sarà associato un **idnodo** univoco e periodicamente verrà aggiornato il valore della batteria **carica.** Un'altra entità fondamentale è **reti**: poiché si potranno memorizzare contemporaneamente le informazioni di più reti, quest'entità memorizza l'**idrete**. In ogni momento è possibile attivare o disabilitare la rete e, quindi, vi è l'esigenza dell'attributo **abilitata** che fornirà l'informazione richiesta. L'entità rete è collegata tramite una relazione **Associata a** di 1 a n con l'entità **sessioni**, la quale contiene le informazioni della singola sessione associata alla rete: un **idsessione** univoco, il **limite** del valore monitorato oltre alla quale verranno generati gli allarmi e **attiva**, l'attributo che identifica la sessione attualmente in corso associata a quella rete. Un'altra entità è presente nel sistema è **allarmi** fondamentale per la memorizzazione in caso di superazione del limite della sessione. Questa entità è collegata tramite **Gen allarme** all'entità **measure**. con una relazione di 1 a n. Gen allarme ha, inoltre l'attributo **idallarmegenerato**, utile per identificare in modo univoco l'allarme con la misurazione che l'ha generato.

3.5 Web Server e presentazione dei dati

L'ultimo elemento che compone l'architettura è il web server Apache, il quale, tramite specifiche pagine JSP , o altre linguaggi per il web, rende disponibili i dati agli utenti che le richiedono.

Questa parte della progettazione dev'essere personalizzata per la tipologia di monitoraggio eseguito, poiché la presentazione dei dati potrà variare a seconda del caso. Possono, comunque, essere definite delle scelte comuni indipendentemente dal monitoraggio eseguite:

- scelta della rete di cui si vogliono ottenere le informazioni
- scelta della sessione associata alla rete
- visualizzazione degli allarmi della sessione
- statistiche della sessione
- visualizzazione in tempo reale dello stato della rete

Per svolgere i precedenti punti è stato scelta la tecnologia JSP (**JavaServer Pages**) [11], tecnologia appositamente creata per applicazioni web che, a differenza delle statiche pagine html, fornisce un contenuto dinamico della pagina richiesta, indipendentemente dalla piattaforma utilizzata.

Tramite l'utilizzo del **JDBC** (*Java DataBase Connectivity*) [10] all'interno delle pagine, viene visualizzato il contenuto delle tabelle precedentemente descritte e, grazie ad un'apposita presentazione dei dati, si permette all'utente di interagire con esso. è un connettore per database che consente l'accesso alle basi di dati da qualsiasi programma scritto con il linguaggio di programmazione Java, indipendentemente dal tipo di DBMS utilizzato. È costituita da una API, raggruppata nel package java.sql, che serve ai client per connettersi a un database e fornisce metodi per interrogare e modificare i dati.

Di seguito la *Figura 13* che mostra la differenza tra Html e il modello utilizzato per accedere ai dati

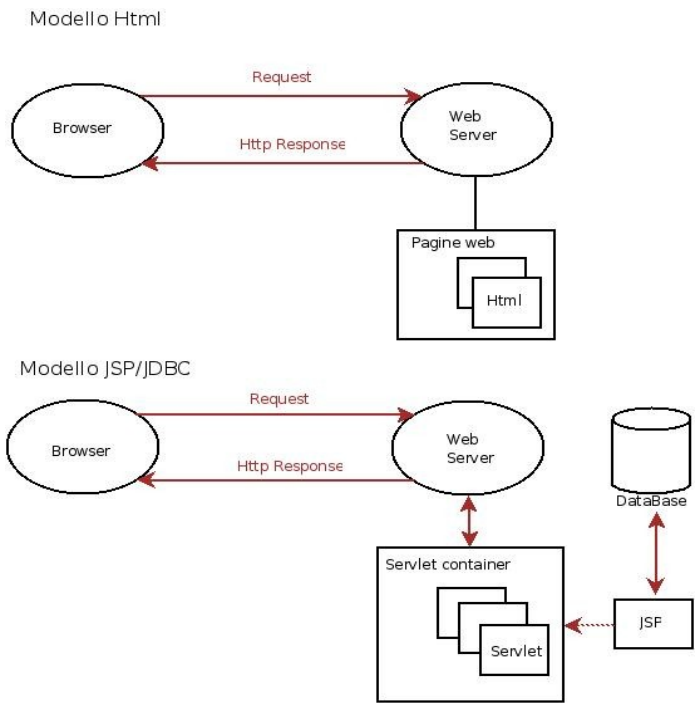

Figura 13. Differenza tra utilizzo di pagine Html e JSP

Nel modello **HTML** (mostrata sopra in figura), il browser, tramite l'inserimento dell'indirizzo web, esegue una richiesta al server di una pagina html identificata univocamente dall'indirizzo. Il server risponde inviando la pagina html statica, cioè tale e quale a com'è memorizzata all'interno della zona di memoria richiesta.

Il modello **JSP** (Javaserver Pages) è una tecnologia Java per lo sviluppo di applicazioni web che forniscono all'utente contenuti dinamici in formato Html (o XML). La tecnologia JSP è correlata a quello delle servlet, poiché all'atto della prima invocazione la pagina JSP viene automaticamente convertita in servlet. Una pagina JSP può, quindi, essere vista come una rappresentazione ad alto livello di una servlet. Deve, quindi, essere presente sul web server un servlet container ed un motore JSP in grado di convertire la pagina da JSP a servlet e restituire i dati richiesti dall'utente. Nella presente architettura la scelta è ricaduta su Tomcat. Il fatto di utilizzare Java per la creazione delle pagine dinamiche offre un notevole vantaggio: quello di poter utilizzare le API JDBC per eseguire query sul Database e restituirlo come output all'interno della pagina Html.

Per quanto riguarda la visualizzazione in tempo reale della rete, la scelta effettuata è radicalmente diversa. A differenza delle pagine JSP, viene utilizzata un sistema che

integra un'applet sulla pagina dell'utente ed una servlet a livello server (*Figura 14*). Quest'ultima gestisce il dialogo tra l'applet ed il server SFS tramite una connessione e richiesta di dati al server [12]. Il server risponderà con i dati del pacchetto ricevuto, i quali verranno inoltrati all'applet che penserà alla visualizzazione a video.

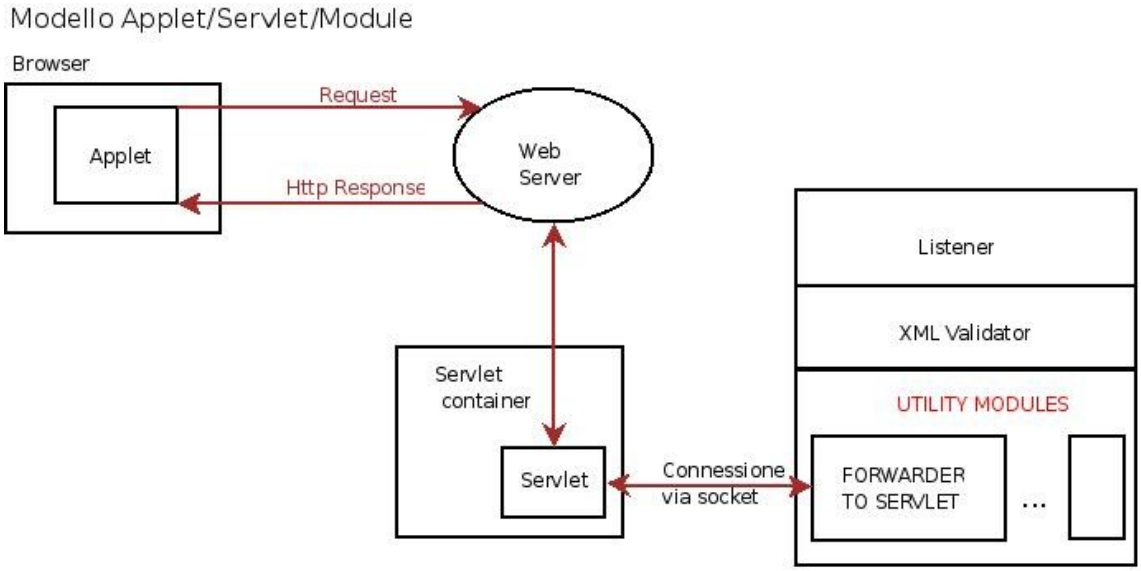

Figura 14. Interazione tra l'applet della pagina web e il SFS passando per la servlet

Come si vede nella figura lo schema è simile nella quello del modello JSP contenuto in *Figura 13*, ma presenta anche delle notevoli differenze. Innanzitutto la presenza dell'applet Java sul client. Le applet Java sono delle particolari applicazioni eseguite all'interno di programmi ospiti (nel caso presentato il browser) e vengono programmate similmente a programmi standard. Tramite un'applet si può dare, quindi, ad una pagina web ancora più dinamicità che il solo uso del JSP. Nel nostro caso, l'obbiettivo è quello di instaurare un dialogo in tempo reale con il server. Oltre all'applet viene utilizzata una speciale servlet che fungerà da ponte di comunicazione tra il server e l'applet: periodicamente l'applet eseguirà una richiesta alla servlet. Quest'ultima ad ogni richiesta provvederà ad eseguire una connessione al server e più esattamente al modulo adibito al forwarding dei dati all'applet. In caso venga eseguita con successo la connessione, il server invierà i dati alla servlet i quali li inoltrerà direttamente all'applet che li utilizzerà. Questo modello viene utilizzato per avere una visione istantanea della struttura della rete altrimenti impossibile tramite l'utilizzo di pagine JSP. L' applet, appunto, essendo una vera e propria applicazione Java, permette l'uso di una maggior numero di

istruzioni. In questo caso, il vantaggio dell'utilizzo di un'applet è anche quello di allegerire il carico del DataBase, utilizzato ampiamente dal SerialForwarderServer.

4 Dettagli implementativi

In questo capitolo vengono illustrate le scelte implementative fatte: in primo luogo viene mostrato il metodo di conversione del flusso grezzo di byte in un documento XML. Successivamente viene descritto il lavoro eseguito dal server all'arrivo di un file XML con i dati del campionamento. Infine vengono mostrati due moduli implementati del framework, in particolare quello adibito alla comunicazione tra il server e l'applet .

4.1 Trasformazione del pacchetto: dai byte a XML

Nel paragrafo 1.3.5 è stato descritto il pacchetto trasmesso dalla WSN e tutte le informazioni che contiene: a seconda dell'applicazione utilizzata dai sensori che compongono la rete, il flusso di dati cambia. Il nodo gateway riceve tutti i pacchetti e li trasmette immediatamente alla FoxBoard, la quale ha il compito di convertire questi in documennti strutturati che descrivono formalmente il contenuto. E' chiaro quindi che i file XML sono la scelta più consona per questo motivo.

L'applicazione **serialforwarder** descritta nel paragrafo 2.2 ha il compito di leggere i dati in arrivo dalla WSN, identificare il tipo di pacchetto e creare quindi il file XML sulla base delle proprie conoscenze. Per eseguire efficientemente il proprio lavoro il programma utilizza files contenuti internamente alla FoxBoard, la quale li consulterà all'arrivo di ogni pacchetto prima di creare il file XML.

I file utilizzati sono le seguenti:

- file **messages**: un file di tipo testuale che contiene le associazioni dei codici esadecimali dei tipi di pacchetto alle stringhe che lo definiscono
- file **skel.xml**: è lo scheletro del file XML. E' composto dalle informazioni in comune di tutti messaggi TinyOs
- files **tipo.xml**: il tipo di messaggio specifico in arrivo dalla WSN.

I due files XML presenti nella board rappresentano il modello da seguire. Una volta che il messaggio arriva, vengono unificati i file **skel.xml, tipo.xml** e generato

pacchetto.xml in cui i campi presenti vengono riempiti con i byte del pacchetto.

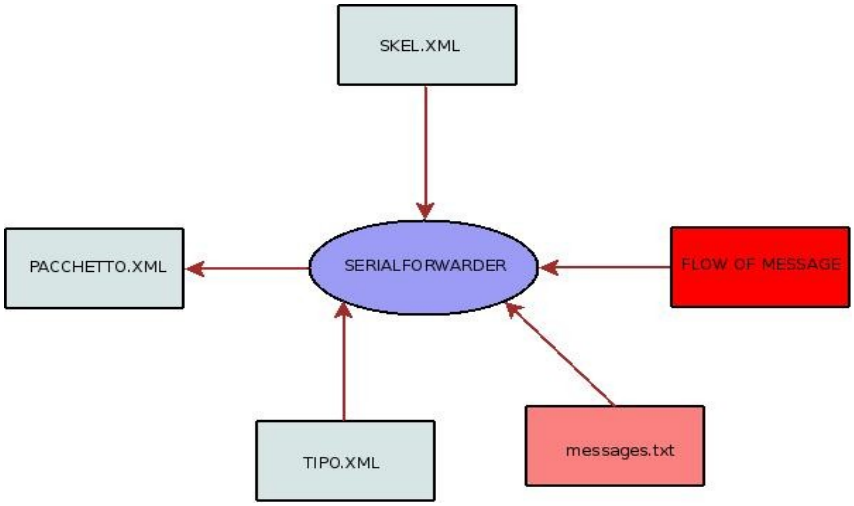

Figura 15. File utilizzati dal serialforwarder per produrre l'XML

Compito del serialforwarder è proprio quello di creare l'output analizzando i dati in ingresso e incrociandoli con quelli presenti nella Fox (*Figura 15*). Il box rosso (**flow message**) indica l'arrivo del pacchetto dalla rete sensoriale; il box rosa (**messages.txt**) rappresenta il file interrogaro dal serialforwarder per riconoscere il tipo di messaggio in arrivo; i box grigi (**tipo.xml**, **skel.xml**, **pacchetto.xml**) indicano i file XML processati dal programma: i due aventi le frecce in entrata vengono letti e utilizzati per produrre pacchetto.xml.

Innanzitutto viene letto il tipo di messaggio arrivato e, confrontando il valore dell'ottavo byte, viene confrontato con quello contenuto nel file *messages.txt* che associa il valore del byte a sinificato del messaggio. Un esempio del file utilizzato è il seguente:

```
xuart=0x00
multihopDebug=0x03
surgeMessage=0x11
xsensor=0x32
xmultihop=0x33
multihopMessage=0xFA
other=0xAA
```

Una volta conosciuto il tipo di pacchetto da analizzare, il programma passa alla costruzione dell'output incapsulando i file dei messaggi, in un unico file che rappresenta il modello del pacchetto.

Se, per esempio, il pacchetto pervenuto fosse di tipo multihop e al suo interno vi fosse racchiuso un messaggio di tipo surgemessage (applicazione standard di Tiny Os) verrà costruito l'output pacchetto.xml partendo dai seguenti file:

- Il file *skel.xml (Figura 16)*, involucro esterno dei dati, contiene le informazioni standard dei messaggi TinyOs come lunghezza, identificatore del tipo di messaggio,etc...

skel.xml:

```
<?xml version="1.0"?>
<skel xmlns="http://www.arslogica.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.arslogica.com skel.xsd">
  <head>
    <campo name="Sync_Byte" nbytes="1"></campo>
    <campo name="Packet_Type" nbytes="1"></campo>
  </head>
  <payloadData>
    <tinyOsPacket>
      <campo name="Length" nbytes="1"></campo>
      <campo name="Frame_Control_Field" nbytes="2"></campo>
      <campo name="Dat_Sequence_Number" nbytes="1"></campo>
      <campo name="Dest_PAN_Address " nbytes="2"></campo>
      <campo name="Dest_Address" nbytes="2"></campo>
      <campo name="Message_Type" nbytes="1"></campo>
      <campo name="Group_ID" nbytes="1"></campo>
      <data></data>
      <validity>
        <campo name="CRC" nbytes="2"></campo>
      </validity>
    </tinyOsPacket>
  </payloadData>
  <tail>
    <campo name="Sync_Byte" nbytes="1"></campo>
  </tail>
</skel>
```

Figura 16. Composizione del file XML skel.xml contenente tutti i dati generali

- il file *multihopmessage.xml (Figura 17)*, contenente le informazioni sui nodi che hanno instradato il messaggio, il nodo di origine, etc.

multihopmessage.xml:

```
<multihopMessage>
      <multihopHead>
      <campo name="Source_Address" nbytes="2"></campo>
      <campo name="Origin_Address" nbytes="2"></campo>
      <campo name="Sequence_Number" nbytes="2"></campo>
      <campo name="Origin_Sequence_Number" nbytes="2"></campo>
      <campo name="TTL" nbytes="1"></campo>
      <campo name="Id" nbytes="1"></campo>
      <campo name="Rack_id" nbytes="2"></campo>
      </multihopHead>
      <applicationData></applicationData>
</multihopMessage>
```

Figura 17. Il file XML multihop che contiene le informazioni sul percorso del pacchetto

- il file *surgemessage.xml (Figura 18)*, il quale contiene le informazioni della specifica applicazione. Nell'esempio proposto il file conterrà anche i dati del campionamento eseguito dal sensore (in questo caso *Temp*).

surgemessage.xml:

```
<?xml version="1.0"?>
            <surgeMessage>
                  <campo name="Sequence_Number" nbytes="4"></campo>
                  <campo name="Intvolt" nbytes="2"></campo>
                  <campo nme="Temp" nbytes="2"></campo>
                  <campo name="Parent_Addr" nbytes="2"></campo>
                  <campo name="NeighBorSize" nbytes="1"></campo>
                  <campo name="Retransmission" nbytes="1"></campo>
                  <campo name="NeighBors" nbytes="6"></campo>
                  <campo name="Quality" nbytes="6"></campo>
            </surgeMessage>
```

Figura 18. Il file XML surgemessage con i dati del campionamento

Per completare l'esempio prendiamo l'arrivo del seguente pacchetto:

```
7E4624210896FFFF7E00067D00000A00A534A2050322FFFFA501000059081A0B00000
30800000A00140022001B004C00A4B27E
```

Figura 19. Stringa analizzata dal programma serialforwarder

Seguendo il funzionamento presentato nel presente paragrafo, l'applicazione serialforwarder costruirà l'output della *Figura 20*

pacchetto.xml:

```
<?xml version="1.0"?>
<skel xmlns="http://www.arslogica.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.arslogica.com skel.xsd">
  <head>
    <campo name="Sync_Byte" nbytes="1">7E</campo>
    <campo name="Packet_Type" nbytes="1">46</campo>
  </head>
  <payloadData>
    <tinyOsPacket>
      <campo name="Length" nbytes="1">24</campo>
      <campo name="Frame_Control_Field" nbytes="2">2108</campo>
      <campo name="Dat_Sequence_Number" nbytes="1">96</campo>
      <campo name="Dest_PAN_Address " nbytes="2">FFFF</campo>
      <campo name="Dest_Address" nbytes="2">7E00</campo>
      <campo name="Message_Type" nbytes="1">06</campo>
      <campo name="Group_ID" nbytes="1">7D</campo>
      <data>
<multihopMessage>
      <multihopHead>
      <campo name="Source_Address" nbytes="2">0000</campo>
      <campo name="Origin_Address" nbytes="2">0A00</campo>
      <campo name="Sequence_Number" nbytes="2">A534</campo>
      <campo name="Origin_Sequence_Number" nbytes="2">A205</campo>
      <campo name="TTL" nbytes="1">03</campo>
      <campo name="Id" nbytes="1">22</campo>
      <campo name="Rack_id" nbytes="2">FFFF</campo>
      </multihopHead>
      <applicationData>
            <surgeMessage>
            <campo name="Sequence_Number" nbytes="4">A5010000</campo>
            <campo name="Intvolt" nbytes="2">5908</campo>
            <campo name="Temp" nbytes="2">1A0B</campo>
            <campo name="Parent_Addr" nbytes="2">0000</campo>
            <campo name="NeighBorSize" nbytes="1">03</campo>
            <campo name="Retransmission" nbytes="1">08</campo>
            <campo name="NeighBors" nbytes="6">00000A001400</campo>
            <campo name="Quality" nbytes="6">22001B004C00</campo>
            </surgeMessage>
      </applicationData>
</multihopMessage>
      </data>
      <validity>
        <campo name="CRC" nbytes="2">A4B2</campo>
      </validity>
    </tinyOsPacket>
  </payloadData>
  <tail>
    <campo name="Sync_Byte" nbytes="1">7E</campo>
  </tail>
</skel>
```

Figura 20. L'output finale del programma che verrà mandato al server

4.2 Server: ricezione, validazione e accodamento

Nel seguente paragrafo viene descritto l'approccio implementativo che sta alla base del server: **ricezione** del file XML, **validazione** e **passaggio** ai moduli di gestione dei dati. L'aspetto fondamentale dell'*SFS* è quello di una struttura in cui i due thread superiori eseguono la parte di gestione del documento in arrivo dalla rete sensoriale, demandando gli aspetti dell'utilizzo dei dati ai thread sottostanti in una struttura a più livelli, come visto nel capitolo 2.3. In questo modo, un'estensione del programma è facile ed immediata e richiede la sola implementazione del modulo, che tramite la propria coda dei pacchetti provenienti dalla WSN potrà utilizzare i dati come meglio crede.

La parte implementativa è stata codificata in linguaggio Java e per gestire la struttura modulare sono stati implementati Thread separati, ognuno con una specifica funzione.

Nel codice contenuto nella *Figura 21* è riportata l'inizializzazione del server, che utilizza i due moduli implementati nel caso di studio.

```
public SFServer (String[] args){
            System.gc();
            for (i=0;i<MyConstants.N_MODULE;i++){
                  events[i]=new Coda();
            }
            Thread                Listener=new                Thread(new
Listener(args[0],args[1],events));
            Thread          Forwarder=          new          Thread(new
ForwarderToServlet(events[0]));
            Thread DBInsert=new Thread(new DBInsert(events[1]));
            try{
            Listener.start();
            Forwarder.start();
            DBInsert.start();

          } catch (Exception e){

          }
```

Figura 21. SFS e lancio del Listener e dei moduli Forwarder, DBInsert

Ogni thread viene inizializzato con una coda che gestisce separatamente. In particolare nell'esempio riportato, viene inizializzato il *Listener*, il modulo che gestisce il collegamento con il *serialforwarder*, e a cui vengono associate tutte le code dei moduli

sottostanti. Questo thread è adibito alla connessione via socket alla board e alla lettura del file XML. La parte di validazione viene eseguita, poi, da un thread richiamato dal *Listener*: il thread *Validator* il quale gestisce la convalidazione e il passaggio del documento ai moduli che lo utilizzeranno (*Figura 22*). Per la parte di validazione dell'XML (e successiva estrapolazione dei dati) sono state utilizzate le librerie *javax.xml* le quali offrono al programmatore tutte le funzionalità di gestione e raccolta dati.

```
DocumentBuilder parser =

      DocumentBuilderFactory.newInstance().newDocumentBuilder();
            Document document = parser.parse(new File(nomefile));
            SchemaFactory factory =
                 SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA
_NS_URI);
                           Source  schemaFile  =  new  StreamSource(new
File("pacchetto.xsd"));
            Schema schema = factory.newSchema(schemaFile);
            Validator validator = schema.newValidator();
            validator.validate(new DOMSource(document));
            valido=1;
            System.out.print("document valid!\n");
            Event tmpevent=new Event(1,"document valid",document);
            for (i=0;i<MyConstants.N_MODULE;i++){
                  events[i].push(tmpevent);
                  events[i].printqueue(events[i]);
            }
```

Figura 22. Il parser e l'aggiunta alla coda degli eventi del documento valido

4.3 *DBInsert* e il *ForwardToServlet*

Nei seguenti due paragrafi vengono illustrate nel dettaglio le tecniche di programmazione utilizzate per la costruzione dei due moduli implementati: il DBInsert con la funzione di inserire i dati contenuti nel documento XML all'interno del Database, ed il ForwardToServlet con la funzione di inoltrare le informazioni del documento all'applet.

4.3.1 Inserimento valori nel database tramite JDBC

Una volta inserito nella coda il documento, il singolo thread gestirà il prelevamento delle informazioni rilevanti e la gestione di queste.
Il modulo che gestisce l'inserimento all'interno del database, *DBInsert*, viene collegato tramite JDBC al database che mantiene lo storico delle rilevazione (*Figura 23*).

```
Class.forName("org.gjt.mm.mysql.Driver");
            Connection conn = DriverManager.getConnection(
                        "jdbc:mysql://localhost/SensorNetwork",
                        "u_name",
                        "pass");
            for (;;){

             if ((now= dbevent.pop())!=null)
             {//gestisci il documento
```

Figura 23. Connessione tramite JDBC al Database e prelevamento del primo documento non ancora gestito

Dopo aver estratto dalla coda dei documenti *dbevent* il primo documento non ancora processato, verranno estratte tutte le informazioni rilevanti del caso ed eseguito un update del database. Il codice di *Figura 24* vuole mostrare come le informazioni sulla misurazione eseguita dal sensore vengano inserite all'interno del database.

```
                res        =        st.executeUpdate("INSERT        INTO
measure(idnodo,idrack,idrete,idsessione,data,time,temp)          VALUE
('"+OriginNode+"','"+ParentNode+"','"+Rete+"','"+sessione+"','2"+Data
+"','"+Tempo+"',"+Temp+")");
```

Figura 24. Un'esempio di query per inserire i dati del campionamento

4.3.2 La comunicazione tra *SerialForwarderServer* e applet

Un paragrafo a parte è occupato dalla comunicazione tra il SerialForwarderServer e l'applet che fa visualizzare la struttura della rete in tempo reale. Questa scelta è stata fatta per allegerire la mole di lavoro a cui il database puntualmente è sottoposto da parte del server che, quasi costantemente, inserisce le misurazioni all'interno del DB.

Come descritto nel capitolo 2.5, sono tre le parti coinvoltre nel dialogo:

- il modulo residente sul server *ForwardToServlet*
- la *servlet* che risiede sul server
- l'*applet* che viene caricata dall'utente

Simile a ciò che avviene nel modulo *DBInsert*, il modulo *ForwardToServlet* estrae innanzitutto i dati dal documento XML e crea una stringa che verrà successivamente interpretata dall'applet. Finita la creazione penserà alla creazione di una socket in ascolto, che manterrà per la durata di due secondi (*Figura 25*) . Nel caso in cui nessuna richiesta avvenisse, significherebbe che nessuna applet è attiva e che nessun utente è connesse per visualizzare la struttura. Nel caso in cui la richiesta di connessione avvenisse, verrebbe invece generata la stringa e scritta sulla socket in scrittura.

```
for(;;){
  if ((now=dbevent.pop())!=null)
    servSocket = new ServerSocket(Integer.parseInt(porta));
    servSocket.setSoTimeout(2000);
      try {
           cs = servSocket.accept();
          } catch (IOException e) {[...] }
      if (cs!=null){
                  BufferedOutputStream outputsock = new
                  BufferedOutputStream(cs.getOutputStream());
                    //creo la stringa da mandare alla servlet
                  [...]
                  outputsock.write(temp.getBytes(),
0,temp.length());
```

Figura 25. Invio del codice dal server SFS alla servlet

La **servlet** ha la sola funzione di eseguire il collegamento tra il modulo descritto e l'applet. Le servlet sono delle speciali classi di Java con speciali metodi generati in caso di richiesta (o risposta) da parte di un Web browser. In questo caso ogni volta che l'applet esegue una richiesta alla servlet, la servlet si occupa di eseguire una connessione al server via socket, prelevare la stringa con i dati ed inviarli all'applet che li richiede.

```
protected void doGet(HttpServletRequest request, HttpServletResponse
res) throws ServletException, IOException {
            String indirizzo=new String ("127.0.0.1");
            String porta=new String("8001");
            Integer c;
            PrintWriter out;
            out = res.getWriter();
              try{
                              Socket        socket        =        new
Socket(indirizzo,Integer.parseInt(porta));
              DataInputStream inputsock = new

      DataInputStream(socket.getInputStream());
                    while ((temp=inputsock.readline())!=null)
            {
              if (temp.length()>0)
              {
                     Calendar cal=Calendar.getInstance();
                    res.setContentType("text/plain");
                Runtime rt = Runtime.getRuntime();
                StringBuffer send=new StringBuffer(temp);
                out.println(send);
              }
              else
             System.out.println(temp);
```

Figura 26. Lettura dei dati della servlet

L'**applet** si occupa della visualizzazione della struttura della rete e, in caso di allarme, del monitoraggio degli avvisi. Ciclicamente richiede i dati alla servlet tramite le seguenti righe di codice.

```
public void run() {
      url = new URL(getCodeBase(), "/SensorViewer/ShowMemservlet");
      con = url.openConnection();
      con.setUseCaches(false);
      in = con.getInputStream();
      textStream = new DataInputStream(in);
      line1 = textStream.readline();
```

Figura 27. Lancio dell'applet nella pagina dedicata alla visualizzazione dei dati

5 Caso di studio: server farm e *Status*

Nel seguente capitolo verrà spiegato come la struttura precedentemente descritta, è stata utilizzata per lo specifico caso di studio: il monitoraggio delle temperature di una server farm.

5.1 Scenario

I server sono quanto di più prezioso si possa avere un'azienda: contengono dati vitali, vendono prodotti on-line o possono servire tutti gli impiegati della ditta ed è per questo che è importante monitorarli. Con queste prerogative è nato il progetto *Status*, sottoprogetto di VIGILIA (cap 1.4), sviluppato da ArsLogica.

Normalmente vengono colocati in un unico ambiente in modo da poter centralizzare la gestione, la manutenzione e la sicurezza. Non è raro avere server farm composte da centinaia (se non migliaia) di singoli server. Solitamente vengono posti in speciali ambienti climatizzati e raccolti in armadi chiamati Rack , mentre più armadi costituiscono un Gruppo . Vengono delineate quindi le seguenti figure:

- il server
- il rack
- il gruppo

La *Figura 29* successiva riassume il concetto appena spiegato.

Figura 29. Composizione di una server farm

L'importanza del monitoraggio delle temperature in strutture di questo è di enorme

importanza, poiché un eccessivo surriscaldamento di un server può causare notevoli danni all'azienda che lo utilizza e può riperquotersi all'interno dell'intero rack. La soluzione proposta prevede l'utilizzo di un singolo sensore all'interno di ogni server alimentato a batteria. Tale sensore rileva ad intervalli regolari la temperatura all'interno del server ad intervalli prestabiliti (in laboratorio era previsto un campionamento ogni 5 secondi). Ogni nodo viene etichettato con un ID univoco che lo identifica all'interno dell'intera rete. Il server viene poi inserito all'interno del rack. All'armadio stesso viene applicato un nodo (Cluster Head) sempre identificato univocamente da un ID (Rack ID).

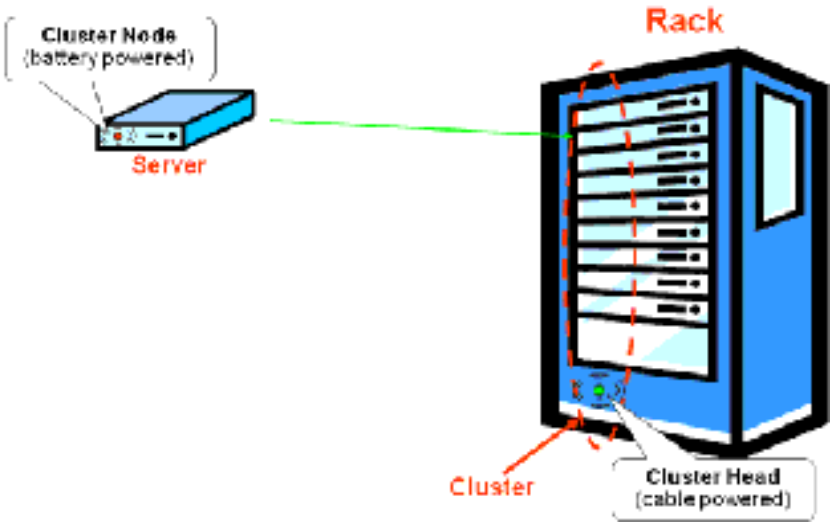

Figura 30. Applicazione dei nodi nei server e nel rack

I Rack così formati creano la vera e propria rete mesh che, tramite l'utilizzo di pacchetti multihop, raggiunge il gateway rappresentato da un nodo collegato alla Fox Board, la quale eseguirà la traduzione dal flusso di byte al formato XML, seguendo il procedimento descritto nel capitolo 3.1

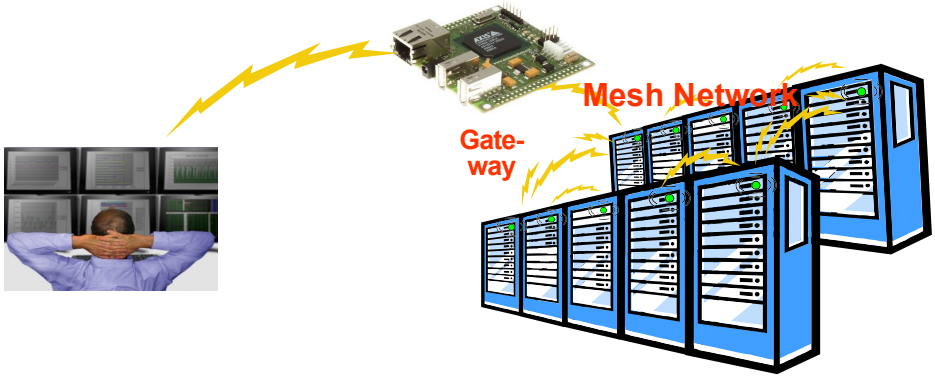

Figura 31. Collegamento tra server farm e l'operatore

5.2 Struttura del pacchetto e xml

Ogni nodo al'interno della rete manderà, quindi un pacchetto contenente tutte le informazioni rilevanti del caso ed in particolare:

- Origin Address: il nodo che ha originato il pacchetto
- Rack Id: Il rack a cui appartiene il nodo.
- Group id: il gruppo a cui appartiene il campionamento.
- Temp: la temperatura campionata dal sensore presente sul nodo.

Tutte queste informazioni (piiù altre), vengono raccolte e trasformate nel file XML trasferito al server in ascolto.

La simulazione (eseguita presso i laboratori ArsLogica) ha previsto l'utilizzo di sei sensori: due rappresentati i rack e tre rappresentanti i server all'interno del rack. Il sesto sensore è stato utilizzato come nodo gateway collegato alla FoxBoard. In locale è stato fatto girare il server Apache ed il Database MySQL e sono stati raccolti i dati in una sessione di dieci minuti. L'utente poteva visualizzare lo stato della rete dei sensori e, in caso di superamento del valore di soglia, essere avvertito dell'allarme. Come descritto nel cap 2.4 l'allarme può essere catalogato a seconda della tipologia, in modo da creare uno storico che potrà essere utilizzato per statistiche successive.

5.3 Visualizzazione dei dati della server farm

La visualizzazione dei dati rappresenta il fine ultimo del framework, poiché lo scopo è proprio quello di rendere le informazioni disponibili all'utente. Come visto nel paragrafo 2.5 sono state utilizzate due tecnologie per sviluppare questa parte: la prima è l'utilizzo di pagine JSP per la visualizzazione di contenuti dinamici sotto forma di pagine Html. La seconda è quella dell'integrazione di applet con una servlet connessa al server SFS.

Vediamo nel dettaglio le funzionalità possibile tramite l'interfaccia web.

Figura 32. La scelta delle reti dall'interfaccia

La schermata iniziale dell'interfaccia web è presentata in *Figura 32*: in alto un menu permette di cambiare pagina e di arrivare direttamente nella pagina richiesta. In ordine da sinistra a destra le opzioni possibili sono le sguenti:

- **selezione rete**: la pagina iniziale da cui l'utente seleziona la rete da osservare
- **visualizza struttura**: da questa pagina si accede all'applet che mostra in tempo reale la struttura della rete, la presenza di allarmi e traccia dei grafici in tempo reale relativi ai nodi
- **nodi**: serve per visualizzare i nodi presenti nella rete e lo stato della batteria.
- **log misurazioni**: tutte le misurazioni fatte all'interno del sistema. E' possibile filtrare o ordinare i risultati in base ad alcuni parametri.
- **gestione allarmi**:visualizza le misurazioni che hanno superato il limite impostato ma che ancora non sono stati memorizzati dall'operatore. Da qui sarà possibile associare agli allarmi la causa, in modo che in un secondo tempo sarà possibile rivedere le cause del surriscaldamento dei server.
- **allarmi salvati**: qui vengono memorizzati tutti gli allarmi etichettati con la relativa descrizione della causa. E' possibile filtrare, ordinare o modificarne la descrizione.

Innanzitutto l'utente dovrà scegliere la rete di cui vuole ottenere le informazioni. Come si vede in *Figura 32* queste informazioni vengono rappresentate tramite una lista di

reti: viene visualizzato l'id della rete, lo stato (se attiva o meno) ed inoltre un pulsante che permette di disattivare il monitoraggio della rete. Quest'ultima funzione è stata introdotta poiché nel momento in cui un tecnico volesse lavorare alla rete o ai nodi della WSN, si vuole poter interrompere il processo di memorizzazione dei dati che potrebbero risultare altrimenti corrotti.

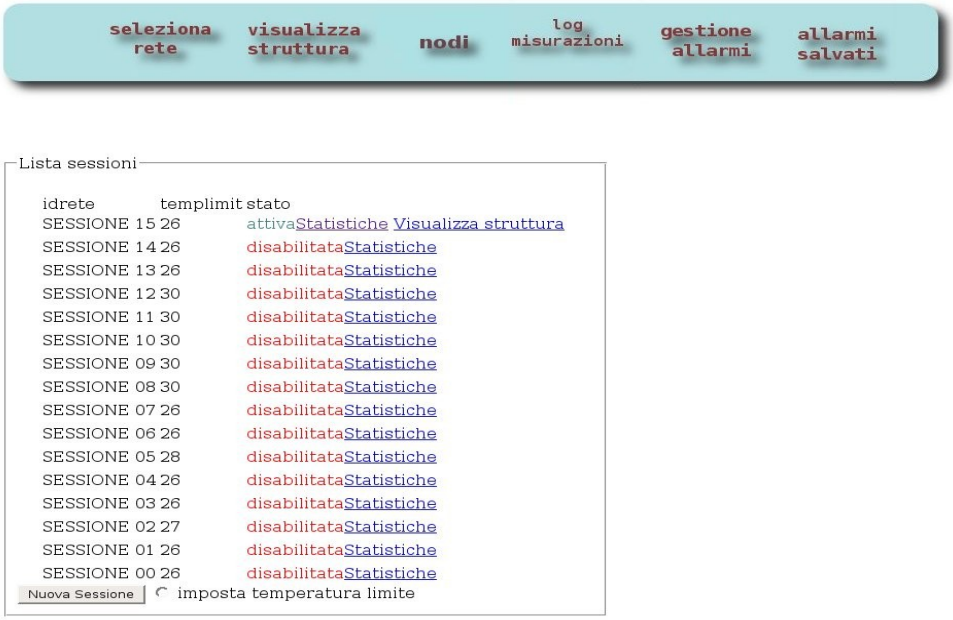

Figura 33. La scelta della sessione o la crezione di una nuova

Successivamente alla scelta della rete viene proposto all'utente una nuova pagina (*Figura 33*): all'interno del box verranno visualizzate tutte le sessione associate alla rete. Solamente una sarà attiva (segnalata in verde) e si potrà visualizzare la struttura, mentre di tutte le altre si potranno comunque avere informazioni tramite il collegamento alle statistiche . Nella parte inferiore al box è presente un pulsante con la funzione di creare una nuova sessione associata alla rete, interrompendo quella corrente. Della nuova sessione bisognerà impostare la temperatura limite oltre alla quale si vuole ricevere l'avviso di errore. Scegliendo di visualizzare la struttura della sessione attiva verrà mostrata la pagina illustrata in *Figura 34.* La pagina è suddivisa in due box principali: quello di sinistra è l'applet che visualizza la struttura; quello di destra permette di visualizzare le statistiche delle connessioni associate alla rete.

All'interno dell'applet le informazioni disponibili sono molteplici: nella parte centrale vengono visualizzati i vari server suddivisi per rack all'interno della rete. Il colore del pallino centrale identifica la presenza di allarmi non ancora visualizzati associati al nodo. Nella parte destra dell'applet, invece, un menù ci permette di selezionare uno specifico nodo e di visualizzare il grafico in tempo reale degli ultimi dieci pacchetti associati ad esso.

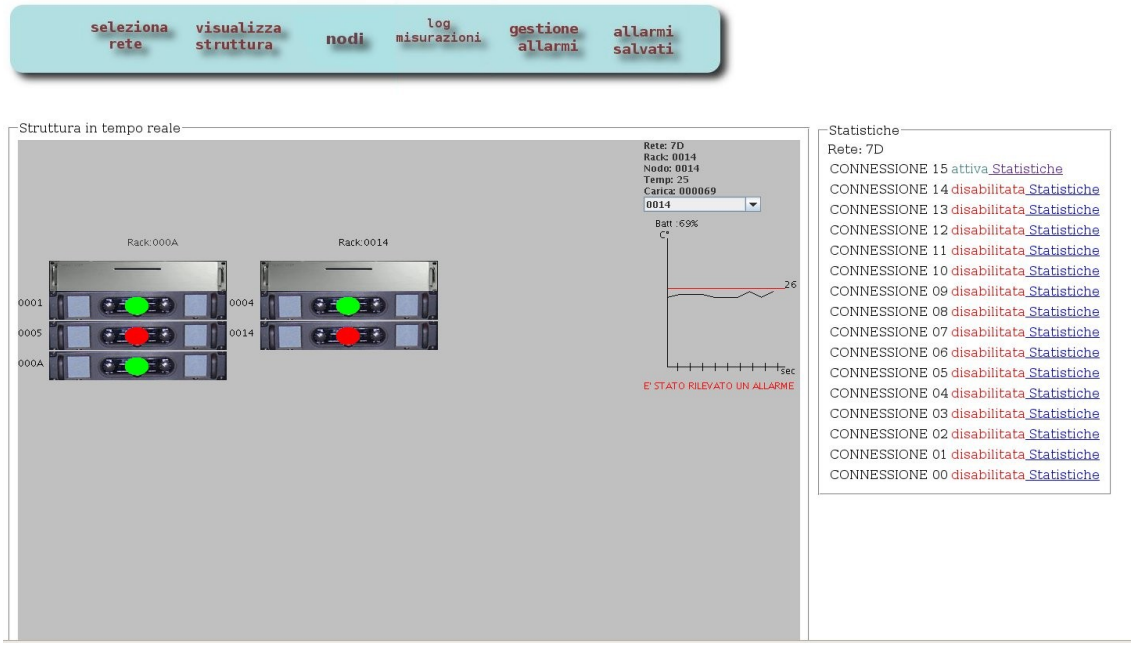

Figura 34. Visualizzazione dei server tramite l'applet

Per la visualizzazione della struttura, inizialmente è stata utilizzata la libreria Jung (Java Universal Network/Graph), nata appositamente per la gestione di grafi. A livello di visualizzazione, però non sempre un grafo è immediato per comprendere la struttura della rete, quindi si è passati all'utilizzo del package standard *java.awt* e alla visualizzazione della struttura tramite il caricamento di immagini che identificano i server e i rack che compongono la rete.

Il colore verde identifica uno stato di temperatura inferiore al limite impostato che, se superato, farà variare il colore in rosso.

Parte importante dell'interfaccia web è data dalle statistiche della sessione.

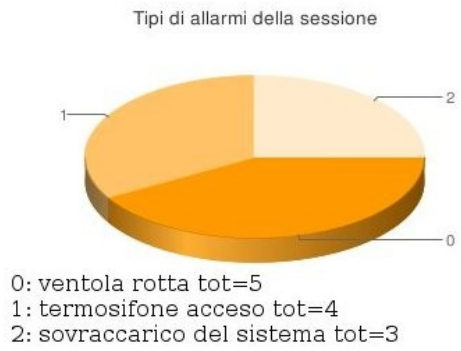

Figura 35. Esempio di grafico relativo agli allarmi

Per la parte delle statistiche la scelta è ricaduta sulle API di Google[7] che tramite l'invio di una stringa appositamente costruita, permettono il disegno di grafici precisi ed in modo veloce, senza la scrittura di ulteriori classi apposite o l'acquisto di tool specifici. I tipi di grafici sono di due formati: il primo, grafico a torta (*Figura 35*), viene utilizzato per rendere disponibile all'utente quanti e quali sono gli allarmi memorizzati nel database. In un modo rapido e veloce l'operatore può, quindi, capire quali sono i danni che interessano maggiormente un server e, in base ai risultati, prendere le dovute contromisure (es: aggiunta di una ventola per il raffredamento, distribuzione del carico di lavoro,...). Nell' esempio illustrato in figura all'interno della sessione sono stati rilevate tre tipologie di allarmi diverse, descritte nella parte in basso.

La seconda tipologia di grafico (*Figura 36*) utilizzato è quello a linee e rappresenta l'andamento della temperatura nella sessione di campionamento e può essere selezionato per nodo, rack o quello dell'intera rete. Il grafico contiene le informazioni di inizio e fine sessione e vengono indicate la temperatura massima, minima e la media dei campionamenti rilevati. Nell'esempio presentato in figura si vede una sessione iniziata il 15-02-2008 alle ore 10:27 e terminata lo stesso giorno alle ore 10:40. In questo periodo la temperatura massima è stata di 28°C, la minima di 21°C e la media di 23°C.

7 La documentazione della Google Chart API è disponibile al sito http://code.google.com/apis/chart/

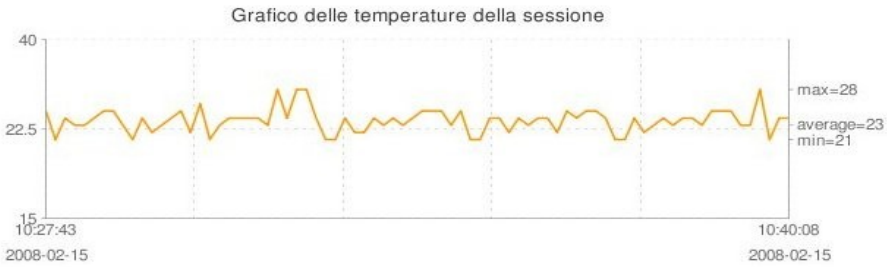

Figura 36. Esempio di grafico relativo alle temperature della sessione

Dal menù principale, come descritto in precedenza, è possibile selezionare la visione dei nodi che compongono la rete. La schermata visionata è quella di *Figura 37.*

Figura 37. Visualizzazione dei nodi e della carica della batteria

Come si vede dall'immagine nel box principale vengono visualizzate le indormazioni sull' id del nodo, il rack a cui appartiene, la carica della batteria in percentuale e informazioni su di essa. Nel caso presentato nella figura, le scritte appaiono gialle poiché tra il 70% ed il 40% la batteria è considerata in esaurimento e cambiata al più presto per evitare problemi di perdite di pacchetto.

La schermata successiva (*Figura 38)* mostra la gestione degli allarmi avvenuti non ancora schedati. Nella parte principale della pagina vengono visualizzati gli allarmi e tutte le informazioni relative ad essi.

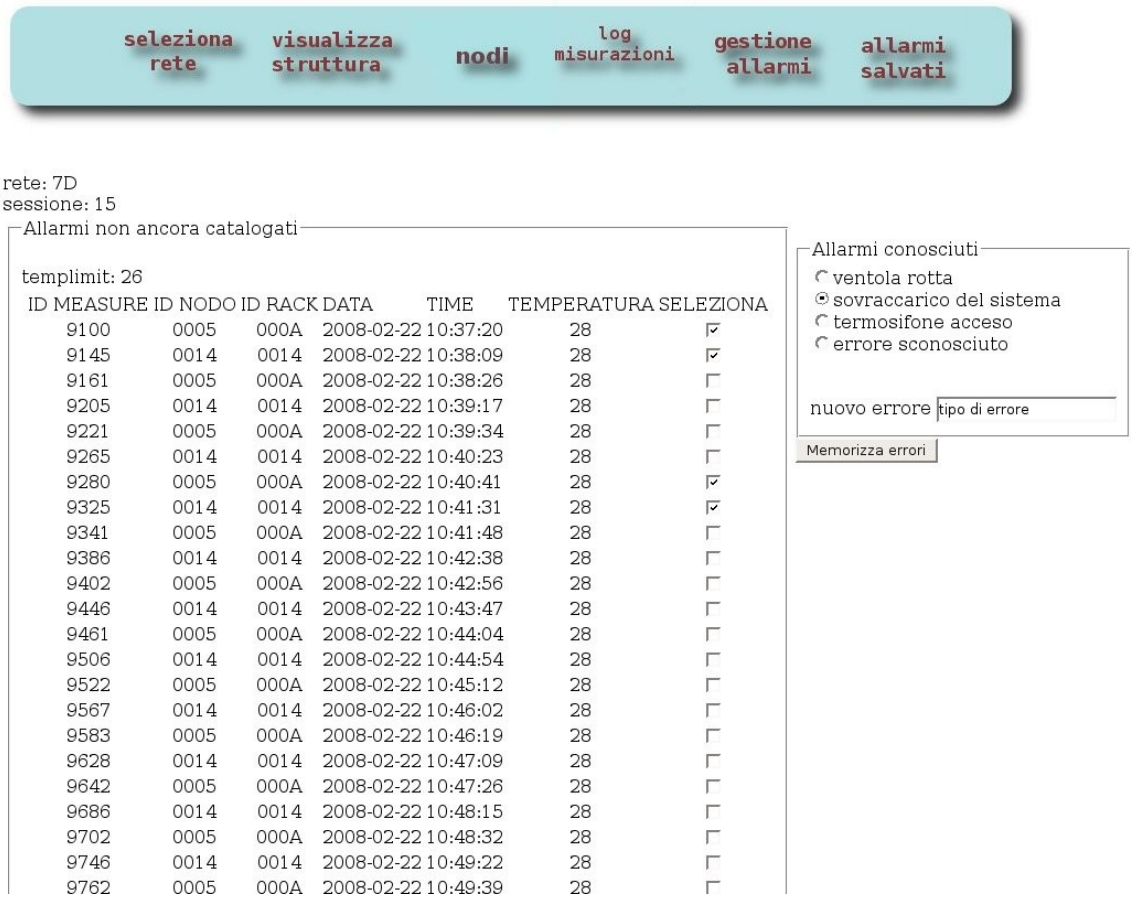

Figura 38. Gestione degli allarmi non ancora catalogati

Selezionando uno o più allarme e la tipologia dal box sulla destra, sarà possibile catalogarli in modo che l'operatore potrà visualizzare in un secondo tempo la causa degli allarmi. Infatti la schermata successiva (*Figura 39*) vuole mostrare proprio l'utilizzo della catalogazione come strumento di filtraggio per gli allarmi. Da questa schermata l'utente avrà a disposizione nel box principale tutte le informazioni relative all'allarme. Nella parte destra invece, si potrà scegliere il tipo di operazione da eseguire per la visualizzazione: ordinamento, filtraggio in base ad un parametro o modifica della descrizione di un allarme.

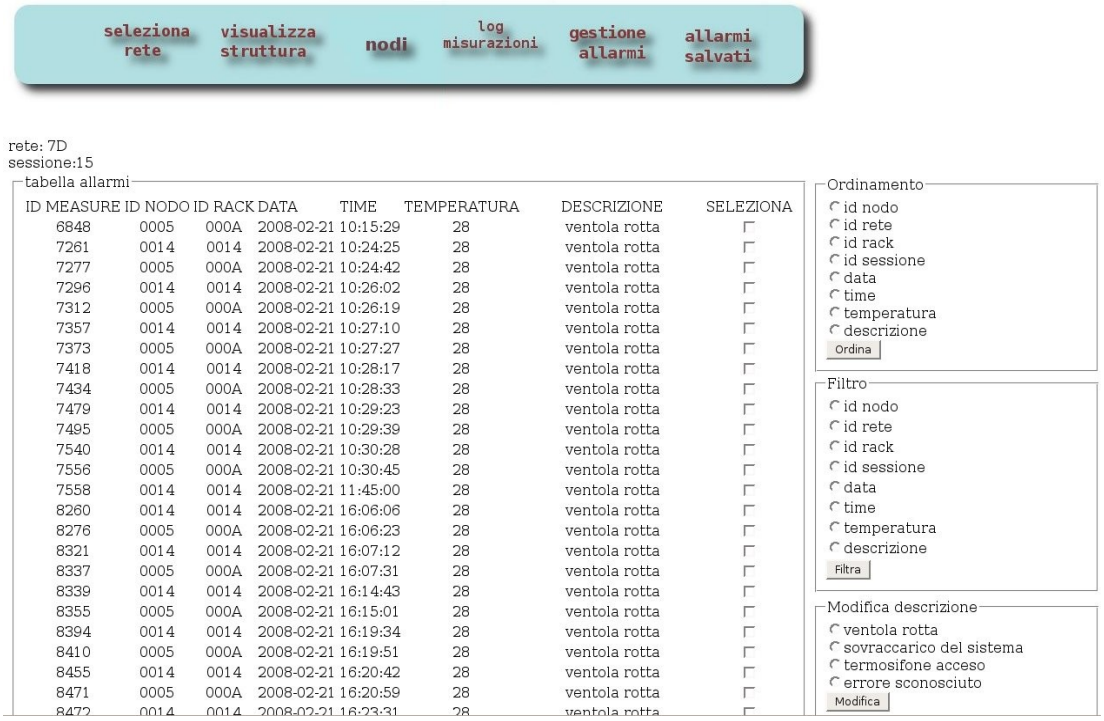

Figura 39. Gestione degli allarmi catalogati

6 Test

Il testing del software si è tenuto parallelamente allo sviluppo del framework. Innanzitutto è stato testato il programma di generazione dell' XML serialforwarder per controllare la correttezza dei dati generati. Per fare questo è stato utilizzato un unico nodo, la FoxBoard e un grezzo server con l'unico compito di validare i file arrivati dalla rete WSN. Il nodo era programmato con l'applicativo *Surge*, creato dagli stessi realizzatori di TinyOs, utilizzato principalmente per visualizzare la topologia della rete e il contenuto dei messaggi in arrivo.

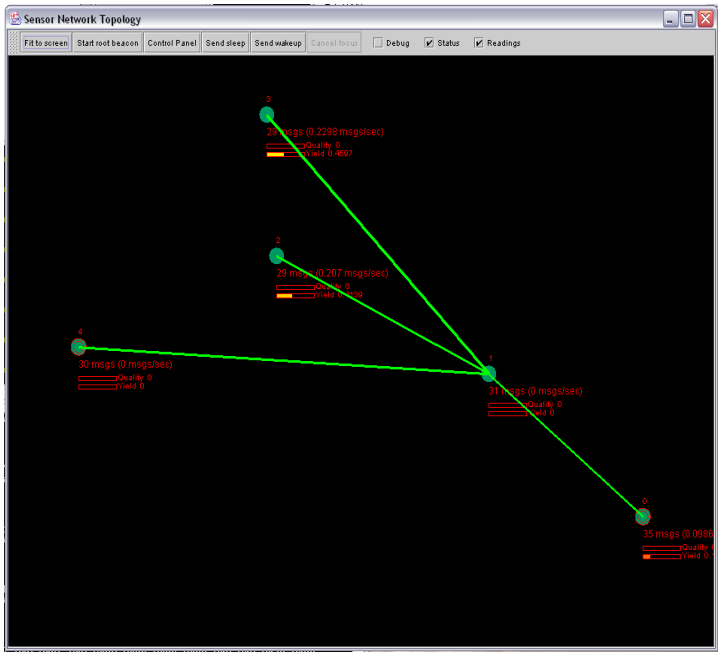

Figura 39. Una schermata di Surge

La *Figura 39* mostra la schermata principale di *Surge*, l'applicativo Java che è stato utilizzato in locale per visualizzare struttura dela rete e dati in arrivo. I nodi vengono rappresentati con dei cerchi verdi e le linee indicano i collegamenti tra i vari sensori. Verificato che la crazione dei file XML era stabile e non generava errori si è passato all'integrazione con il SFS, Database e interfaccia web.

Il test finale del framework sviluppato è stato effettuato presso i laboratori di ArsLogica ed ha previsto l'utilizzo del seguente hardware e software:

- cinque sensori Telos adibiti alla rilevazione della temperatura. I nodi,

precedentemente programmati con TinyOs , sono stati etichettati con ID **0001**, **0004**, **0005**, **000A** e **0014**.

- una **FoxBoard** a cui era applicato il nodo di gateway ed su cui era in esecuzione l'applicativo **serialforwarder**.
- un pc adibito alla funzione di server. Sul calcolatore era presente il **SerialForwarderServer**, MySQL server ed il Database per la memorizzazione dei dati, il server **Apache** con la **servlet** e le pagine **JSP** per la visualizzazione dei dati.
- il browser web **Firexfox** per la visualizzazione dei dati

I sensori che fino al momento del test utilizzavano *Surge*, sono stati riprogrammati con l'applicativo *ServerMonitor* che utilizza la stessa struttura del pacchetto vista nel paragrafo 3.1.

Due dei sensori (etichettati con ID **000A** e **0014**) rappresentavano i rack di computer e sono stati posti a distanza di dieci metri dal gateway della rete e l'uno dall'altro. Ad una distanza di poci centimetri dai rack sono stati posti gli altri tre sensori: due aggiunti al primo rack ed uno al secondo. Appena sistemati i nodi, sono stati lanciati gli applicativi sulle rispettive piattaforme. Successivamente è stata selezionata la rete (*Figura 32*) e creata una connessione (*Figura 33*) impostando come limite di sessione **25 °C**. Alla visualizzazione della struttura, subito l'applet ha iniziato ha raccogliere i pacchetti e a costruire la struttura logica dei server e dei relativi rack (*Figura 34*). Tutte le informazioni relative ai nodi (*Figura 37*) e alle misurazioni erano disponibili. Anche lo scambio dei nodi da un rack ad un altro è stato rilevato e non ha dato nessun tipo di problema.

Un problema ha invece colpito la rilevazione della temperatura, poiché il microcontrollore dei nodi non generava un valore accettabile. Tramite la documentazione relativa ai nodi è stato cercato se era presente un problema di conversione del dato da milivolt a gradi, ma non è stato trovato. Allora il test si è preferito svilupparlo raccogliendo sul server un valore accettabile di pacchetti proveniente dalla rete (circa cento) e modificando il valore della temperatura. Il sistema,

in questo modo, è potuto essere testato in tutte le sue funzionalità, compresa la generazione e la gestione degli allarmi o la visualizzazione delle statistiche.

Valutando i test eseguiti il sistema ha retto senza problemi la riprogrammazione dei nodi, il che indica una forte capacità di adattamento alle numerose applicazioni applicazioni utilizzate sui sensori programmati con TinyOs. La gestione delle informazioni tramite interfaccia web risulta chiara e permette all'utente di avere una visione completa della struttura. Peccato per quanto riguarda l'impossibilità di testare il software con dati precisi di temperatura anche se la ricezione dei pacchetti e la visualizzazione della struttura di server e rack è avvenuta con successo.

Conclusioni

La tesi aveva come scopo quello di progettare e sviluppare un framework funzionante per l'accesso e la visualizzazione dei dati tramite Internet. La modellazione dell'architetura ha avuto come prerogativa il fatto di creare un'interfaccia tra la rete sensoriale e la presentazione dei dati che potesse essere riutilizzata in futuro anche cambiando il tipo di programmazione dei nodi in modo da rilevare altri tipi di valori. Sviluppando, poi, una presentazione dei dati apposita per l'applicazione utilizzata, l'obiettivo principale è stato raggiunto. In fase di testing la programmazione dei nodi è cambiata spesso, ma il modello ha retto ai cambiamenti effettuati, richiedendo solamente la sostituzione dei modelli di file XML e nessun tipo di modifica per quanto riguarda server o presentazione dei dati. Allo stato attuale dello sviluppo dell'applicazione vi è la possibilità di monitoare una rete di sensori di temperatura, la possibilità di eseguire più sessioni, il controllo in tempo reale della struttura della rete, setting degli allarmi e visualizzazione delle statistiche.

Il vantaggio principale del framework proposto è quello dell'adattabilità della rete a numerosi tipi di osservazioni, assicurata dalla struttura dei primi due livelli del sistema. Creando un layout apposito è possibile, per esmpio, il monitoring di celle frigo, di caldaie e non solo. Tuttavia un problema da non trascurare è la creazione dei file XML: il programmatore dovrà essere a conoscenza del pacchetto in arrivo dalla rete sensoriale poiché, nonostante alcuni standard definiti da Tiny Os, la struttura del pacchetto è comunque variabile e definita in fase di programmazione. Un secondo svantaggio può essere il fatto della memorizzazione dei dati in modo centralizzato, il che porta, in caso di spegnimento del server, ad una mancata memorizzazione o visualizzazione dei dati. Il sistema presenta degli interessanti sviluppi per il futuro. Un possibile sviluppo immediato è quello della creazione di un modulo in grado di avvertire preventivamente l'operatore dell'avvicinamento di un allarme. Il sistema di intelligenza artificiale dovrà apprendere dagli allarmi avuti in precedenza e, osservando lo sviluppo del sistema, capire se ci sono o meno rischi di allarmi.

Il sistema potrebbe anche essere programmato per eseguire delle funzioni di risoluzione degli allarmi, a seconda del monitoraggio eseguito. Per esempio, nel caso di studio

presentato, il sistema potrebbe innescare l'avvio di una ventola per il raffreddamento a causa del superamento del valore di soglia impostato.

Bibliografia

[1] J.M.S. Pearce, 'A brief history of the clinical thermometer'. In *Oxford Journal* Volume 45, Numero 4, 2003.

[2] K. S. J. Pister, J. M. Kahn, and B. E. Boser. 'Smart dust: Wireless networks of millimeter-scale sensor nodes'. In *UCB Electronics Research Laboratory Research Summary*, 1999.

[3] G.J. Pottie, 'Wireless Sensor Network'. In *IEEE ITW* June 1998.

[4] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh. 'Monitoring volcanic eruptions with a wireless sensor network'. In *Proc. Second European Workshop on Wireless Sensor Networks (EWSN 05)* January 2005.

[5] Ning Xu, 'A Survey of Sensor Network Applications'. Computer Science Department University of Southern California. 2003

[6] lan F. Akyildiz, WellJan Su, Yogesh Sankarasubramaniam, Erdal Cayirci, 'A Survey on Sensor Networks'. In IEEE Communication Magazine, Agosto 2002.

[7] P. Levis, S. Madden, J. Polastre, R. zewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, e D. Culler. 'Tinyos: An operating system for wireless sensor networks'. In W. Weber, J. Rabaey, and E. Aarts, editors, Ambient Intelligence. Springer-verlag, 2004.

[8] Thorn, Jeff. 'Deciphering TinyOS Serial Packets'. Octave Technology. In *Octave Tech Brief #5-01* Aprile 2005

[9] T.Bray, J.Paoli, C.M.Sperberg-McQueen, 'Extensible Markup Language (XML) 1.0' . In *W3C Recommendation* 10 February 1998

[10] A. Yang, J. Linn,D. Quadrato, 'Developing Integrated Web and Database Applications Using JAVA Applets and JDBC Drivers'. In *ACM. Sigcse Bulletin*, vol.30, no.1, March 1998, pp.302-6.

[11] E. Pelegri-Llopart e L. Cable. 'JavaServer PagesTM Specification', version 1.1. SUN Microsystems, Nov. 1999.

[12] J. Hunter, W. Crawford 'Java Servlet Programming' *O'Reilly* 2001 pag.303