ISTITUTO DI SCIENZA E TECNOLOGIE
DELL'INFORMAZIONE "A. FAEDO"

L'EVOLUZIONE DELLA DOMOTICA VERSO L'INTERNET DELLE COSE

Matteo Pampana, Vittorio Miori

Indice

Elenco delle figure

Elenco dei codici

Sommario

La presente relazione ha lo scopo di illustrare gli obiettivi e i risultati raggiunti durante la realizzazione di un progetto presso il Domotics Lab dell'Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo" (ISTI) del Consiglio Nazionale delle Ricerche (CNR) di Pisa.
E' stata analizzata e realizzata una soluzione, chiamata *gateway*, per aprire una finestra su Internet ad una rete domotica eterogenea, ovvero formata da dispositivi domotici con tecnologie diverse, ed è stata realizzata un'interfaccia web per l'interazione con essa. Si possono, ad esempio, spegnere le luci di casa nostra da qualsiasi parte del mondo tramite questa semplice applicazione. Ogni dispositivo viene inoltre associato a un indirizzo Internet, sfruttando il vastissimo spazio di indirizzamento di *IPv6*, tramite il quale può essere interrogato e sollecitato. Per rendere il tutto più *user-friendly* è stato realizzato un server DNS per tradurre in nomi simbolici di facile memorizzazione (es. lampadina, sveglia, lavatrice) i difficili indirizzi IPv6, formati da svariati gruppi di cifre esadecimali.

Capitolo 1

Introduzione

1.1 Domotica

Per Domotica intendiamo la scienza che studia tecniche e metodologie atte a migliorare la qualità della vita all'interno di abitazioni o ambienti antropizzati. Il termine deriva dal francese Domotique, neologismo introdotto in Francia unendo i termini Domus (casa) e Informatique (Informatica). Gli scopi che le applicazioni domotiche si prefiggono possono essere suddivisi nel seguente modo:

- incremento del livello di comfort: le applicazioni domotiche devono rendere più accogliente lo spazio abitativo, facilitandone la gestione soprattutto per gli utenti disabili (ad esempio standard per l'audio/video, per il controllo e il monitoraggio remoto di elettrodomestici, luci, tapparelle, porte, etc);

- raggiungimento di un adeguato livello di sicurezza: i sistemi domotici devono garantire l'incolumità dell'utente a fronte di situazioni di emergenza (ad esempio tecniche di antintrusione, rilevamento di incendi, fughe di gas, etc.);

- ricerca di tecniche per il risparmio energetico: l'impiego della domotica deve consentire una gestione più accurata ed efficiente del livello dei consumi energetici (ad esempio strumenti avanzati per ottene-

re informazioni che permettano una più equa distribuzione dei carichi energetici).

Il termine domotica è anche sinonimo di home automation e building automation ; il primo termine si riferisce all'automazione di una singola abitazione, il secondo all'automazione di complessi residenziali, alberghi, ospedali, centri commerciali, fabbriche e aerei.

1.2 Sistemi domotici e Problematiche

Una delle grosse limitazioni alla diffusione di sistemi domotici è l'impossibilità di cooperazione tra tecnologie appartenenti a diversi standard e/o produttori. Quindi, una volta che l'utente investirà su un tipo di sistema, sarà vincolato in futuro all'acquisto di sole tecnologie compatibili con esso. E ciò si traduce in una dipendenza commerciale dell'utente nei confronti di un certo produttore. Emerge quindi l'esigenza di riuscire a mettere in contatto queste tecnologie e di permetterne l'interoperabilità. Per ovviare a questo problema è stato sviluppato, presso l'ISTI-CNR di Pisa, DomoNet. L'obiettivo di questo software, come verrà descritto più in dettaglio nel seguito, è proprio quello di studiare un'architettura che permetta la collaborazione tra dispositivi eterogenei dal punto di vista tecnologico e di fornire la possibilità di implementare applicazioni su di essi. Questo diventa possibile creando un livello di astrazione che permetta di descrivere i dispositivi domotici sia dal punto di vista fisico che comportamentale in modo da avere una visione omogenea indipendentemente dalle tecnologie sottostanti.

Scenario

Per comprendere la necessità di un framework che garantisse l'interoperabilità tra i diversi sistemi domotici presenti sul mercato, siamo partiti da uno scenario generico. In un ipotetico ambiente in cui coesistono alcuni tra i principali middleware domotici, è possibile identificare ogni sistema come una particolare "sotto-rete", pur senza soffermarsi sui dettagli e sulle infrastrutture necessarie.

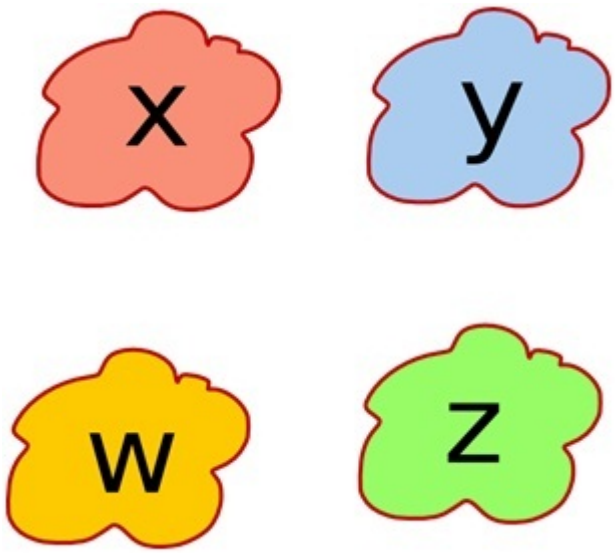

Figura 1.1: Rappresentazione dei middleware domotici

Nella figura precedente ogni "nuvola" rappresenta un middleware, ovvero una sotto-rete all'interno della quale sono presenti sia le infrastrutture hardware e software per il supporto, sia i dispositivi conformi. Una di queste nuvole potrebbe essere, nel caso interessante per questa relazione, la rete Internet. Tutti i dispositivi all'interno della stessa sotto-rete colloquiano e cooperano per costituire un sistema funzionante ed indipendente, ma chiuso. Di fatto è impossibile controllare un dispositivo che si trova in una nuvola tramite un altro presente in una nuvola diversa, poiché non esiste nessun tipo di collegamento tra le due sotto-reti. Per collegamento si intende un'infrastruttura logica che consenta ai dispositivi di qualsiasi sotto-rete di conoscere tutti i dispositivi presenti nelle altre sotto-reti. Infine, anche qualora fosse possibile avere una visione completa dell'intera rete, sarebbe comunque necessario un meccanismo comune per lo scambio di informazioni tra i dispositivi. Vi sono quindi due aspetti da risolvere: da un lato riuscire a costruire un'infrastruttura logica di collegamento tra i vari middleware, dall'altro mettere a punto un linguaggio di comunicazione universale veicolato da quest'infrastruttura. Una possibile soluzione a questi due problemi è quella di introdurre tra ogni coppia di sotto-reti un modulo hardware e/o software che sia provvisto di

un'interfaccia verso entrambe le nuvole. Ogni modulo, che nella terminologia generale è detto gateway, deve non solo fornire funzionalità di traduzione tra i due protocolli delle sotto-reti che collega, ma anche conciliare le differenze tra i paradigmi di comunicazione su cui si basano i due sistemi. Quest'ultima necessità risulta evidente quando i sistemi domotici che si desidera collegare presentano caratteristiche notevolmente differenti (ad es. middleware a configurazione manuale vs. plug and play). L'implementazione di un particolare gateway non può dunque prescindere da una conoscenza approfondita di entrambi i sistemi domotici da collegare. Questa soluzione, tuttavia, risulterebbe poco scalabile a causa del numero di gateway necessari al crescere dei sottosistemi da collegare: infatti, all'aumentare del numero delle sotto-reti, il numero di gateway da sviluppare cresce sensibilmente. A questa prima soluzione se ne è preferita una seconda, che interviene ad un livello superiore. La validità e l'applicabilità dell'idea che si trova dietro tale approccio è stata derivata direttamente dal mondo delle reti di calcolatori. In passato diversi costruttori hanno sviluppato diverse infrastrutture per la costruzione di reti tra piattaforme hardware e software dello stesso produttore (ad esempio AppleTalk dei sistemi Macintosh, NFS per i sistemi Unix, NetBeui per i PC Windows oppure alle reti geografiche IBM SNA, NOVELL/IPX e altre ancora). Questa filosofia creava notevoli problemi di scalabilità delle reti poiché risultava impossibile mettere in comunicazione macchine appartenenti ad infrastrutture diverse. Per risolvere questo problema le diverse aziende in gioco hanno cercato di imporre il proprio sistema come unico standard. Nessuna di esse è riuscita in questo obiettivo, e il motivo di tale fallimento è dovuto soprattutto allo sviluppo di un'infrastruttura, ispirata alla pila ISO/OSI, in grado di superare il vincolo dell'unicità della piattaforma. Si tratta del noto stack di protocolli TCP/IP. Sulla base delle stesse considerazioni si è pensato di introdurre un'ulteriore "nuvola" in grado di gestire i diversi sistemi domotici e che fosse in grado di consentire la comunicazione tra nodi appartenente a sotto-reti diverse.

Infine, focalizzando l'attenzione sulla nuvola della rete Internet, possiamo pensare alla necessità che diversi sistemi di questo tipo comunichino tra loro, per scambiarsi informazioni utili all'utente. Ad esempio, una stazione

meteo potrebbe comunicare alla nostra rete domotica l'arrivo imminente di un temporale e questa potrebbe prendere le relative contromisure: abbassare le tapparelle, suggerire l'uso dell'ombrello agli abitanti, attivare generatori secondari per evitare il rischio di danneggiamenti ai dispositivi elettronici, etc.
Purtroppo i dispositivi domotici non posseggono schede di rete compatibili con il protocollo TCP/IP ne' lo supportano via software. In questa relazione si è cercato di trovare una soluzione a tutti questi problemi, basandoci sul nuovo paradigma dell'Internet delle Cose (Internet Of Things), cercando di mettere su Internet una rete domotica eterogenea in grado di interfacciarsi con un utente remoto.

1.3 Internet Of Things

> *"The Internet of Things (IoT) is a vision. It is being built today. The stakeholders are known, the debate has yet to start."*

Definire con precisione cosa sia l'Internet delle Cose non è facile. E' più una filosofia, una *nuova visione della società*. Recentemente si sono avvicinati tantissimi esperti di diversi settori all'argomento e c'è ancora della confusione attorno a questo nuovo *paradigma*.

La definizione per noi più convincente è quella di una rete dove gli oggetti comunichino tra loro per aiutare l'uomo ad avere un'esperienza di vita migliore. Basti pensare a tutte quelle persone con mobilità limitata, anziani e disabili per capirne immediatamente i vantaggi. Un esempio che calza a pennello è quello scritto precedentemente: la stazione meteo che comunica alle reti domotiche della zona interessata da rovesci di abbassare le tapparelle, senza che nessuno muova un dito. In questa relazione è stato mosso il primo passo per portare le reti domotiche verso l'Internet delle Cose: diamo un indirizzo Internet proprio ad ogni dispositivo (tapparella, lavatrice, lavastoviglie, allarme, etc) e inoltre vi è la possibilità, tramite una semplice interfaccia web, di interagire con essi.

1.4 Una possibile soluzione

Torniamo al problema della non interoperabilità tra sistemi domotici di diverse tecnologie, come è stato risolto questo problema?
L'idea è quella di avere un gruppo di web service ognuno dei quali è capace di interagire con i dispositivi domotici da lui fisicamente raggiungibili, tipicamente in un'area geografica ben precisa e circoscritta intorno alla sua locazione fisica. Ogni area geografica, è rappresentata logicamente dal web service, il quale espone come servizi l'insieme delle funzionalità offerte dai dispositivi presenti. Ogni web service deve essere in grado di interpretare e agire di conseguenza allo stato del livello astratto usando dei moduli capaci di interfacciarsi con i middleware dei dispositivi. Tutto questo deve essere contenuto in una nuova infrastuttura che gestisca l'interazione tra i diversi sistemi domotici. La nuova infrastruttura sviluppata prende il nome di DomoNet, basa il suo funzionamento sui web services e permette di ottenere una visione completa dell'intera topologia della rete (collegando le varie "nuvole") secondo un modello SOA. Al fine di permettere l'interoperabilità tra i vari nodi, è necessario sviluppare degli opportuni gateway detti *tech manager* ed utilizzare una grammatica standard basata su XML, chiamata DomoML. Sarà necessario avere un gateway per ognuna delle nuvole presenti nel sistema e ognuno di essi dovrà avere un'interfaccia verso il sistema domotico al quale si connette e un'altra interfaccia rivolta verso DomoNet.

DomoNet pur risolvendo i problemi legati all'interoperabilità tra diversi sistemi domotici, non risolve il problema dell'Internet of Things. I sistemi domotici che stanno dietro DomoNet vanno a formare una rete domotica *locale* ma non hanno la possibilità di comunicare su Internet. Come è possibile aggiungere questa funzionalità a DomoNet?

L'idea è stata quella di porci ad un livello di astrazione più alto: DomoNet si occupa di interagire con i vari sistemi domotici al livello locale, questo progetto interfacciandosi con DomoNet realizzerà la compatibilità con lo stack Internet.

Ma come assegniamo un indirizzo Internet univoco ad ogni dispositivo? I dispositivi domotici non contengono hardware compatibile con lo stack

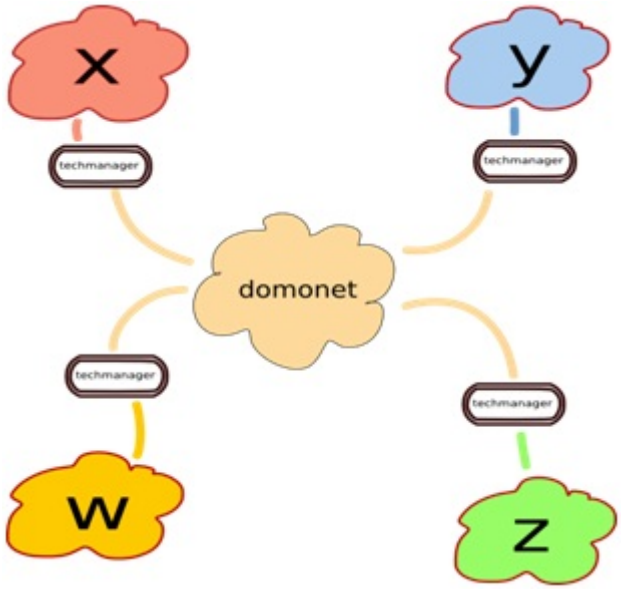

Figura 1.2: Framework DomoNet

TCP/IP. Le possibili soluzioni sono due:

- Realizzare per ogni dispositivo domotico una scheda di rete che realizzi la compatibilità con Internet (livello hardware)
- Realizzare un software (un *gateway*) capace di mappare ogni dispositivo tramite l'indirizzamento IPv6

Per questo progetto è stata scelta la seconda opzione. La prima, infatti, sarebbe poco scalabile: dovremmo montare questa scheda di rete banalissima (non certo una grande innovazione), su tutti i dispositivi domotici di una rete. E' una soluzione che spetta più ai produttori di dispositivi domotici che alla ricerca. Con la seconda opzione, in un colpo solo, tutti i dispositivi dietro DomoNet diventano compatibili con Internet, anche se in maniera indiretta. Inoltre, grazie alla vastissima disponibilità di indirizzi IPv6, possiamo assegnare tramite una "tabella delle corrispondenze", che vedremo in seguito come verrà implementata, un *indirizzo globale univoco* a ciascun dispositivo domotico. A questo punto resta da capire come funzioni IPv6 per sfruttare a pieno le sue capacità e analizzare il framework DomoNet, per capire in che modo interagire con esso.

Capitolo 2

IPv6

La prima esigenza che è sorta nella risoluzione di questo problema è stata quella di indirizzare, in maniera univoca e globale, ciascun dispositivo. L'indirizzamento di rete più diffuso su Internet è IPv4, ma la mole di dispositivi che dovranno essere indirizzati scoraggia immediatamente questa scelta. Ogni dispositivo dovrà avere un proprio indirizzo pubblico e IPv4 avendo uno spazio di indirizzamento molto limitato non può rispondere a questa esigenza. Fortunatamente alla fine degli anni '90 è stato pubblicato un nuovo standard chiamato IPv6.

> *"Se l'intero pianeta, terraferma e acqua, fosse coperto di computer, IPv6 permetterebbe di utilizzare* 7×10^{23} *indirizzi IP per metro quadro [...] questo numero è più grande del numero di Avogadro"*

Questa citazione parla da sola: gli indirizzi IPv6 sono composti di 128 bit e sono rappresentati come 8 gruppi di 4 cifre esadecimali (16 bit).

2.1 Rappresentazione indirizzi IPv6

Ci sono tre forme convenzionali per rappresentare gli indirizzi IPv6 come stringhe di testo:

1. La forma preferita è $x : x : x : x : x : x : x : x$ dove le x sono i valori esadecimale degli 8 gruppi da 16 bit dell'indirizzo. Esempi:

- FEDC:BA98:7654:3210:FEDC:BA98:7654:3210
- 1080:0:0:0:8:800:200C:417A

Non è necessario scrivere gli zero in testa a ciascun gruppo esadecimale.

2. Può capitare che alcuni indirizzi contengano sequenze molto lunghe di bit uguali a zero. Per rappresentare questi indirizzi in maniera più facile e compatta vi è una sintassi "speciale". L'uso del simbolo "::" indica gruppi multipli di 16 bit di zero. Questo simbolo può comparire solamente una volta all'interno dell'indirizzo.

   Ad esempio i seguenti indirizzi:

   - 1080:0:0:0:8:800:200C:417A (indirizzo *unicast*)
   - FF01:0:0:0:0:0:0:101 (indirizzo *multicast*)
   - 0:0:0:0:0:0:0:1 (indirizzo di *loopback*)
   - 0:0:0:0:0:0:0:0 (indirizzo non specificato)

   Possono essere scritti rispettivamente come:

   - 1080::8:800:200C:417A
   - FF01::101
   - ::1
   - ::

3. Una forma alternativa che può essere più conveniente quando si ha a che fare un ambiente misto di nodi IPv4 e nodi IPv6 è $x : x : x : x : x : x : d.d.d.d$ dove le x rappresentano gruppi esadecimali di 16 bit e le d rappresentano le cifre decimali della rappresentazione IPv4.

   Ad esempio:

   - 0:0:0:0:0:0:13.1.68.3
   - 0:0:0:0:0:FFFF:129.144.52.38

   o in forma compressa:

- ::13.1.68.3
- ::FFFF:129.144.52.38

2.2 Rappresentazione dei prefissi

La rappresentazione dei prefissi in IPv6 è simile alla rappresentazione dei prefissi CIDR in IPv4. Un prefisso IPv6 è rappresentato come:

indirizzoipv6/lunghezza

dove:

- indirizzoipv6: è un indirizzo IPv6 espresso nella notazione vista sopra.
- lunghezza: è un valore decimale che specifica quanti bit a sinistra e contigui dell'indirizzo fanno parte del prefisso.

Esempio: 12AB:0:0:CD30::/60

2.3 Tipi di indirizzo

Il tipo di un indirizzo IPv6 è indicato dai bit in testa all'indirizzo che vanno a formare un campo di lunghezza variabile chiamato Format Prefix (FP). Alcuni sono:

Tipo	Prefisso (binario)
Riservato	0000 0000
Indirizzi Unicast Globali	001
Indirizzi Unicast Link Local	1111 1110 10
Indirizzi Unicast Unique Local	1111 110
Multicast	1111 1111

Al tipo "Riservato" appartengono l'indirizzo non specificato (::), l'indirizzo di loopback (::1) e gli indirizzi IPv6 con indirizzo IPv4 integrato (::192.168.0.1). Gli indirizzi Multicast si distinguono dagli indirizzi Unicast dalla coppia esadecimale in testa (FF).

2.3.1 Indirizzi Unicast

Gli indirizzi unicast identificano una specifica interfaccia di un nodo all'interno di una rete. Gli indirizzi IPv6, come visto in precedenza, possono essere rappresentati come:

indirizzo/N

Questa è una notazione molto simile al CIDR di IPv4 che si traduce nel seguente schema:

Il subnet prefix conterrà il FP (Format Prefix) e alcuni campi aggiuntivi a seconda del tipo di indirizzo unicast. L'interface ID è usato per identificare una certa interfaccia su un link. E' univoco al livello di link, ma con buona probabilità può esserlo anche a più ampia portata. Per alcuni tipi di indirizzi Unicast e su interfacce dotate di indirizzo MAC, questo campo dell'indirizzo viene calcolato automaticamente in base al formato EUI-64.

EUI-64

Il formato EUI-64 definisce un metodo per creare un ID dell'interfaccia da 64 bit da un indirizzo MAC. Si effettua inserendo due gruppi di 8 bit, corrispondenti ai valori esadecimali di 0xFF e 0xFE, tra il companyid e il vendorsuppliedid del MAC:

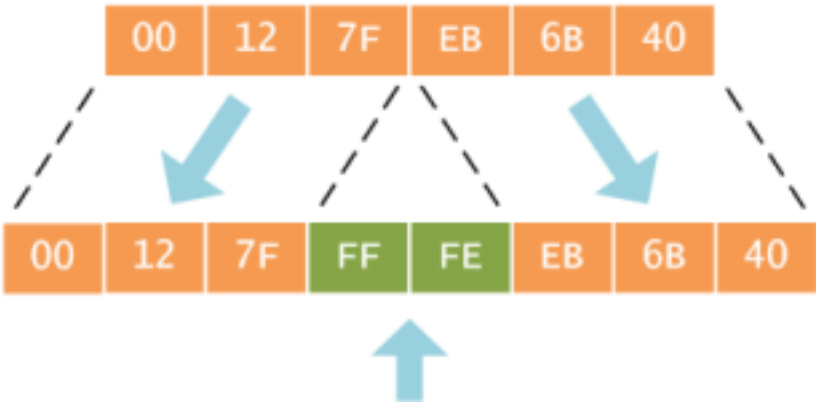

Figura 2.1: EUI-64: Generazione ID interfaccia con MAC Address (fase1)

Quello ottenuto non è ancora l'ID dell'interfaccia. Il secondo bit meno significativo della coppia esadecimale più a sinistra dell'ID ottenuto è un flag

speciale (U/L) che indica lo scope dell'ID dell'interfaccia (0 = unico, 1 = locale). Nell'ipotesi che questo sia un indirizzo IPv6 locale, avremo che:

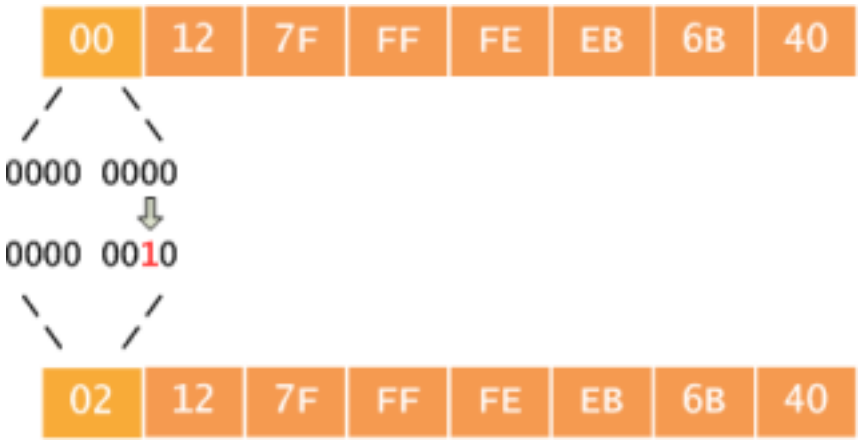

Figura 2.2: EUI-64: Generazione ID interfaccia con MAC Address (fase2)

Link Local

Un indirizzo Link-Local è un tipo di indirizzo IPv6 usato per lo specifico scopo della comunicazione sul link, ovvero per far comunicare due host all'interno di un segmento di una rete locale o punto-punto. I pacchetti con indirizzi Link-Local non devono essere inoltrati dai Router. Questo tipo di indirizzo è stato progettato per essere usato solamente sul link per scopi quali:

- Configurazione automatica degli indirizzi
- ND (Neighbor Discovery)

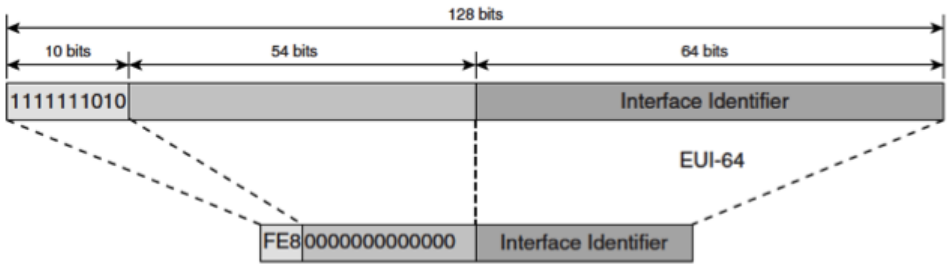

Figura 2.3: Schema indirizzo Link-Local

Il prefisso è costituito da 64 bit, di cui i primi 10 sono il FP (1111 1110 10) e gli altri 54 bit sono tutti a 0. Ottengo così il tipico prefisso Link-Local FE80::/64, che caratterizza tutti gli indirizzi di questo tipo. L'ID di interfaccia viene generato nel formato EUI-64.

Unique Local

Il concetto di indirizzo Unique-Local è simile al concetto di indirizzo Link-Local, solo che in questo caso invece di riferirci al Link ci riferiamo a un sito o a un insieme limitato di siti. Un sito è un insieme di segmenti di una rete locale. Il fatto che l'indirizzo assegnato sia unico in tutto il "sito" è garantito dall'enorme spazio di indirizzamento offerto da IPv6. La probabilità di collisione è prossima allo 0, inoltre alcuni protocolli di assegnamento automatico degli indirizzi effettuano un controllo sull'unicità dell'indirizzo da assegnare.

Come per Link-Local vale il fatto che non i router non debbano inoltrare fuori dal "sito" i pacchetti con indirizzi Unique-Local. Sono indirizzamenti locali (al sito).

1. Il prefisso lungo 7 bit è, come visto precedentemente, 1111 110.
2. L è un flag:
   - 1: prefisso assegnato localmente (per la maggior parte dei casi avremo come prefisso totale FD00::/8)
   - 0: per usi futuri
3. Global ID: un identificatore di 40 bit per generare un indirizzo globale univoco (generato in maniera pseudo casuale)
4. Subnet ID: 16 bit che caratterizzano la sottorete appartenente al "sito"

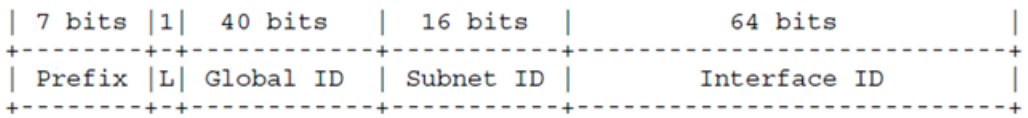

Figura 2.4: Schema indirizzo Unique-Local

Abbiamo, dunque, un "ID di sito" formato dagli 8 bit di prefisso più i 40 di ID, che garantiscono l'univocità dello stesso: abbiamo un prefisso /48, ovvero un numero di siti possibili più grande dello spazio di indirizzamento di IPv4

2.3.2 Indirizzi Multicast

Multicast vuol dire inviare un pacchetto a un certo gruppo di host.

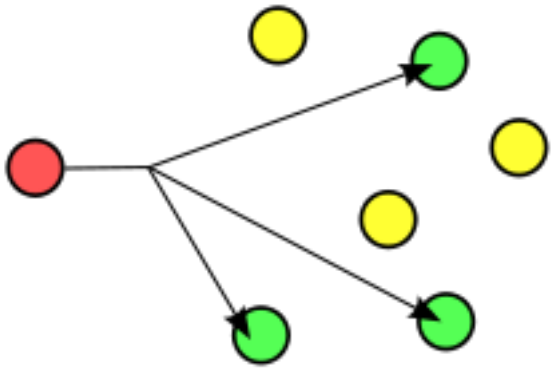

Figura 2.5: Multicast

In IPv6, non esiste più il concetto di *broadcast* - inviando un pacchetto a un gran numero di host non specificati (il famoso 255.255.255.255 di IPv4). E' stato rimpiazzato dal concetto di multicast. In IPv6 tutti i nodi devono supportare il multicast, cosa che non accadeva con IPv4.

Il range di IPv6 del tipo FF00::/8 è riservato per il multicast, come visto precedentemente. Il Multicast può avere diverse tipologie: può essere globale o locale.

Prefisso	Scope
ff02::	MC Link Local
ff05::	MC Unique Local
ff0e::	MC Globale

Lo schema di un indirizzo Multicast è il seguente:

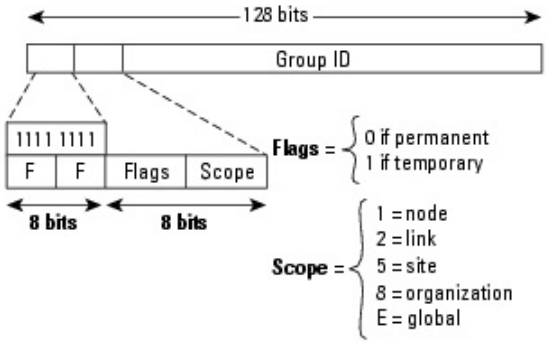

Figura 2.6: Schema indirizzo Multicast

2.4 Assegnazione degli indirizzi

Possiamo assegnare un indirizzo ad un'interfaccia principalmente in tre modi:

1. Automaticamente e stateless: Non richiede la configurazione manuale degli host, ma una configurazione minima router e nessun server aggiuntivo. Questo approccio viene chiamato *SLAAC* che vuol dire StateLess Address Auto Configuration. In pratica al prefisso (che può essere di uno dei tipi visti precedentemente) viene aggiunto automaticamente il Group ID o l'Interface ID tramite il processo EUI-64. Il protocollo di riferimento è l'NDP, che vedremo più avanti.

2. Automaticamente e statefull: questo approccio è quello più familiare agli utenti IPv4 e riguarda l'uso del DHCPv6

3. Manualmente

2.5 NDP - Neighbor Discovery Protocol

Il protocollo NDP (Neighbor Discovery Protocol) per IPv6 viene utilizzato dai nodi, host e router, per stabilire gli indirizzi a livello collegamento per i neighbor che si trovano sui collegamenti associati e per gestire tabelle di

instradamento per ogni singolo collegamento durante le connessioni attive. Inoltre, gli host utilizzano NDP per individuare i router vicini che desiderano inoltrare pacchetti per loro conto e individuare indirizzi collegamento-livello modificati. NDP utilizza ICMP (Internet Control Message Protocol) versione 6 con i relativi tipi di messaggi univoci. In genere, il protocollo NDP di IPv6 corrisponde a una combinazione dei protocolli IPv4 ARP (Address Resolution Protocol), ICMP RDISC (Router Discovery) e ICMP Redirect (ICMPv4), ma con numerosi miglioramenti rispetto ai protocolli IPv4. Questo meccanismo consente a un host di generare indirizzi propri utilizzando una combinazione delle informazioni disponibili in locale e delle informazioni rese pubbliche dai router. Questi ultimi rendono pubblici prefissi che identificano le sottoreti associate a un collegamento, mentre gli host generano un token di interfaccia che identifica in maniera univoca un'interfaccia su una sottorete. Un indirizzo è formato dalla combinazione di entrambi questi valori. In assenza di router, un host può generare solo indirizzi Link Local. Tuttavia, questi indirizzi consentono la comunicazione tra i nodi associati allo stesso collegamento.

Scopi di NDP

NDP è utilizzato per diversi scopi, tra i quali:

- per la configurazione automatica degli indirizzi IPv6
- per determinare i prefissi di rete e altre informazioni di configurazione
- per segnalare indirizzi duplicati (Duplicate IP Address Detection **DAD**)
- per determinare gli indirizzi di Livello 2 dei nodi sullo stesso Link
- per trovare router vicini che possano instradare i pacchetti
- per tenere traccia di quali vicini siano raggiungibili e quali no (Neighbor Unreachability Detection)
- per rilevare la modifica degli indirizzi IPv6

Tipi di messaggi NDP

In questo protocollo router e host si scambiano dei messaggi che possono essere di vario tipo:

- Router Solicitation (RS): Gli Host chiedono attraverso i messaggi RS di localizzare i Router sul link.
- Router Advertisement (RA): I Router segnalano la loro presenza periodicamente(in genere ogni 600 secondi), o in risposta a un messaggio RS
- Neighbor Solicitation (NS): Usato dai nodi per determinare l'indirizzo di un vicino, o per verificare che quel vicino sia sempre raggiungibile.
- Neighbor Advertisement (NA): Per rispondere ai messaggi NS
- Redirect: I Router possono informare gli Host di un hop migliore per la destinazione scelta.

Tutti questi messaggi sono inviati in Multicast attraverso il protocollo ICMPv6.

Un host e un router

Immaginiamo che l'Host A necessiti di un indirizzo globale per uscire su Internet e non abbia ancora un indirizzo IP. Nella rete dell'host vi è un router di frontiera.

1. A crea un messaggio RS, con indirizzo sorgente nullo (::) e indirizzo destinatario Multicast FF02::2 che significa "tutti i router presenti sul link".
2. Il router risponde con un messaggio RA, con indirizzo sorgente, l'indirizzo Link-Local dell'interfaccia e come indirizzo destinatario, l'indirizzo dell'host che ha inviato la RS. Nel campo opzioni mette il Prefisso Globale.

3. A riceve il messaggio, ne estrae il prefisso globale e crea il proprio indirizzo globale, grazie al formato EUI-64 (SLAAC)

Due host sullo stesso link

Immaginiamo che un Host A voglia comunicare con l'Host B, che si trova sullo stesso link:

1. A determina un indirizzo MAC multicast valido per B per generare l'indirizzo IPv6 Multicast giusto. Ad esempio se il MAC Multicast è 33-33-FF-02-6E-A5, l'indirizzo corrispondente sarà FF02::1:FF02:6EA5.
2. A invia un messaggio Multicast NS sul link
3. B risponde in Unicast ad A con un messaggio NA

DAD

Immaginiamo di assegnare un set di indirizzi globali ad una certa interfaccia in maniera casuale. Come sono sicuro che tali indirizzi siano unici in una rete globale? Per questo problema ci viene in aiuto il sistema DAD di NDP. Un nodo come si accorge che l'indirizzo che vuole assegnare a quell'interfaccia sia già presente sulla rete (duplicato) ?

1. Invia un messaggio NS con l'indirizzo in questione
2. Se qualcuno risponde, l'indirizzo è duplicato e viene scartato

Capitolo 3

domoNet

Il framework domoNet nasce come evoluzione del progetto di ricerca HATS (Home Automation Technologies and Standards) presso l'Istituto di Scienza e Tecnologie dell'Informazione ISTI del CNR di Pisa, il cui scopo originario era la creazione di ambienti domestici intelligenti che consentissero l'integrazione e l'interoperabilità tra i servizi offerti. Il modello su cui si basa domoNet è la cosiddetta HAN (Home Area Network), ovvero segue i principi tipici di un'architettura orientata ai servizi (SOA - Service Oriented Architecture). Alla luce del grado di interoperabilità raggiunto con lo sviluppo dei web service nei contesti applicativi SOA, è stato scelto di porre questi ultimi e le tecnologie standard ad essi correlate, come fondamenta del framework. Le principali funzionalità offerte da domoNet sono la cooperazione tra dispositivi domotici eterogenei, e la possibilità di controllare tali dispositivi da remoto, ovvero indipendentemente dalla loro locazione. Questo è stato reso possibile creando un livello di astrazione che permette di descrivere i dispositivi domotici sia dal punto di vista fisico che comportamentale in modo da avere una visione omogenea indipendentemente dalle tecnologie utilizzate. La visione d'insieme del framework è illustrata nella figura 3.1.

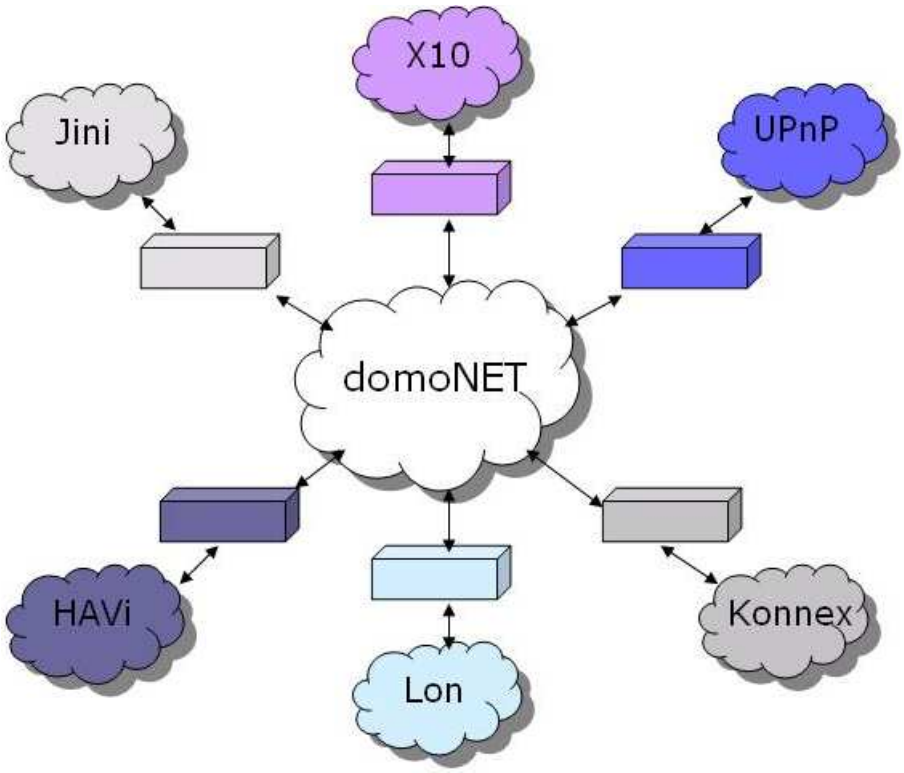

Figura 3.1: Il framework domoNet

3.1 L'architettura di domoNet

L'architettura domoNet è composta da un insieme di server (web service) e da alcuni client per il controllo remoto dei dispositivi domotici. Il controllo remoto permette di interagire con tutti i dispositivi raggiungibili dai web service appartenenti alla rete domoNet. In figura 3.2 viene dato un primo sguardo all'architettura domoNet. L'architettura mostrata è composta da una parte client (il domoNet- client + domoNetclientUI) che, connessa alla parte server (web service + tech manager) è in grado di effettuare richieste tramite TCP/IP affinchè vengano eseguite operazioni sui dispositivi. La parte server, eseguita la richiesta, è in grado di rispondere al client con un dato o con un codice interno di successo o fallimento, in base alla riuscita o meno dell'operazione.

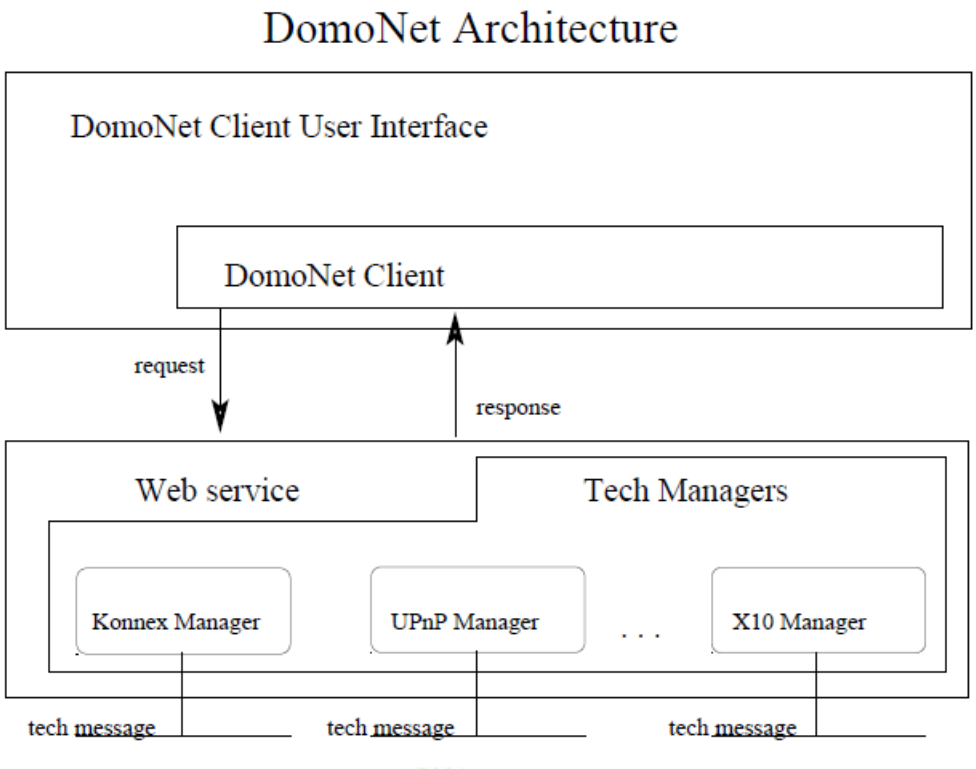

Figura 3.2: L'architettura domoNet

Il livello astratto è implementato attraverso un linguaggio basato su XML chiamato domoML. Con questo linguaggio è possibile descrivere tutti i dispositivi domotici (domoDevice) e i relativi messaggi (domoMessage). DomoML è un linguaggio intermedio da e verso il quale tradurre descrizioni e azioni. Ogni web service è connesso fisicamente ai dispositivi domotici ed è in grado di controllarli invocando moduli specializzati, chiamati tech manager, in base alla tecnologia con la quale è necessario interagire. Per implementare la cooperazione, ogni web service è in grado di catturare un messaggio proveniente dal modulo di una tecnologia, convertirlo in domoMessage per poi indirizzarlo al modulo gestore della tecnologia di destinazione che provvederà a riconvertirlo in modo che possa essere eseguito. Per implementare il controllo remoto, il client è dispone di una lista dei dispositivi domotici connessi ad uno o più web service. Ogni dispositivo è rappresentato usando il formalismo del domoDevice. Al momento in cui deve essere eseguita

un'azione remota, il client genera il corrispondente domoMessage e lo invia verso il giusto web service. Raggiunto il web service, il domoMessage viene inviato al giusto modulo ed eseguito. L'esito o l'eventuale risposta derivata dall'esecuzione viene ritrasmessa al mittente della richiesta col formalismo domoMessage.

3.2 Terminologia

Si riassume nel seguito la terminologia utilizzata:

- dispositivo: apparecchiatura fisica appartenente ad una determinata tecnologia;
- domoML: formalismo XML usato per descrivere e implementare le funzionalità dei dispositivi nell'applicazione;
- domoDevice : un dispositivo domoNet descritto tramite il linguaggio domoML;
- domoMessage : descrizione di un messaggio tramite il linguaggio domoML;
- domoAddress : una coppia composta dall'URL del web service e dall'id che identifica in maniera univoca il domoDevice ;
- real address : l'indirizzo reale di un device nella tecnologia di appartenenza;
- tech manager : modulo del web service che amministra una tecnologia. Ogni web service pùo avere più tech manager di tecnologie differenti;
- tech message : messaggio che proviene o è generato da un tech manager o da un client esterno al web service.

3.3 Lato server: il Web Service

Il lato server dell'applicazione risolve la cooperazione tra i dispositivi che appartengono a diverse tecnologie e esegue i comandi provenienti da altri server o da un client.

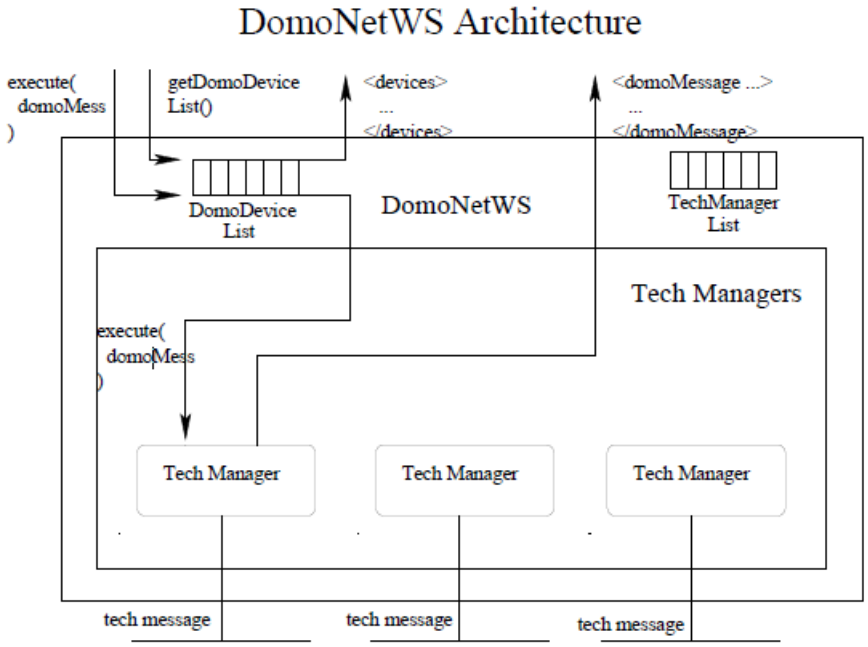

Figura 3.3: L'architettura WebService di domoNet

3.3.1 Eseguire un domoMessage

La richiesta di eseguire un domoMessage pùo venire da un'applicazione client (per il controllo a distanza dei dispositivi), dallo stesso o da un altro web service (per la cooperazione). Quando arriva un domoMessage, per prima cosa viene individuato il domoDevice del dispositivo interessato grazie al DomoDeviceId calcolato usando gli attributi receiverURL e receiverId del messaggio. Dal domoDevice viene individuato il tipo di tecnologia di appartenenza del dispositivo e, con la techManagerList, si identifica il tech mananger di competenza al quale inoltrare il messaggio. Il tech manager traduce il domoMessage nel suo corrispettivo tech message e provvede alla sua esecuzione. Eseguito

il messaggio, il tech manager provvede a costruire un nuovo domoMessage di successo (type=SUCCESS) o fallimento (type=FAILURE) con un'eventuale risposta da parte del dispositivo.

3.3.2 Cooperazione tra i diversi tech Manager

Il tech manager riceve un tech message dalla propria rete ed individua il domoDevice che rappresenta il mittente ed il servizio coinvolto. I dati trovati vengono analizzati dal web service che provvede a verificare l'esistenza di linkedService1 nella descrizione del servizio invocato. Se trovati, genera i nuovi domoMessage e li invia ai web service interessati altrimenti non compie alcuna azione. Ogni web service infatti può anche importare e gestire i dispositivi appartenenti ad altri web service sfruttando la rete Internet e comunicando direttamente con l'interessato. All'interno della domoDeviceList2, i domoDevice importati sono riconosciuti attraverso l'attributo url. In questo modo è possibile avere l'interoperabilità tra dispositivi domotici appartenenti a web service diversi.

3.4 Lato client: il domoNetClient

Il DomoNetclient consiste in un'interfaccia applicativa in grado di visualizzare e modificare lo stato dei dispositivi domotici presenti. Le funzionalità offerte sono le seguenti:

- connessione ad uno o più web service : mediante l' invocazione del metodo List getDeviceList() viene effettuata una chiamata remota indirizzata verso il web service richiesto, creando una copia locale dei domoDevice nella DomoDeviceList indicizzabile tramite il domoDeviceId.

- esecuzione di un domoMessage: mediante l'invocazione del metodo DomoMessage execute(DomoMessage domoMessage) viene effettuata una chiamata al web service coinvolto, il quale eseguirà il domoMessage e fornirà l'esito del comando.

3.5 Il DomoML

DomoML è un linguaggio creato per rappresentare in maniera compatta i dispositivi domotici, i servizi offerti e le relative interazioni, astraendo dalla tecnologia di appartenenza. Viene principalmente utilizzato per descrivere le seguenti entità:

- domoDevice : descrizione astratta dei dispositivi;
- domoMessage : descrizione astratta delle interazioni da e verso i dispositivi.

3.5.1 domoDevice

Un domoDevice pùo essere rappresentato attraverso un albero che ha al massimo 4 livelli di profondità:

1. device: apre il tag che descrive il domoDevice dando generiche informazioni come:
   - description: una descrizione in lingua naturale del domoDevice ;
   - id: valore usato per prendere e generare il domoDeviceId. Identifica il domoDevice all'interno del web service ;
   - manufacturer: il produttore del dispositivo;
   - positionDescription: una descrizione in lingua naturale della locazione del dispositivo;
   - serialNumber: il numero seriale (può essere composto anche da lettere) del dispositivo;
   - tech: la tecnologia originale del domoDevice rappresentato;
   - type: la tipologia del dispositivo;
   - URL: identifica il web service coinvolto nella gestione del dispositivo.

   Tutti i campi sono opzionali, eccetto per id, URL e tech , che sono essenziali per identificare e utilizzare un dispositivo.

2. service: descrive un singolo servizio offerto dal domoDevice. Questa informazione è usata per creare un domoMessage e per convertirlo nel tech message. Per ogni domoDevice ci possono essere più servizi. I campi di questo tag sono:

   - description: una descrizione in linguaggio naturale del servizio;
   - name: un identificatore che può essere utile quando il domoMessage viene generato e tradotto in tech message.
   - output: il domoML.domoDevice.DomoDevice.DataType si aspetta un valore di ritorno quando viene eseguito il domoMessage. Questo campo va omesso se non è previsto alcun valore di ritorno;
   - outputDescription: una descrizione in linguaggio naturale per l'attributo output. Se non è presente l'attributo output, questo attributo non deve essere utilizzato;
   - prettyName: una etichetta in linguaggio naturale del servizio;

3. due possibili tag per questo livello:

   - input: indica che il servizio ha bisogno di un valore di input per poter essere eseguito. Questo tag è opzionale e ogni servizio pùo avere più input. I campi per questo tag sono:
     - name: un identificativo dell'input;
     - description: una descrizione dell'input;
     - type: il domoML.domoDevice.DomoDevice.DataType delll'input atteso;
   - linkedService: è il servizio da associare quando il servizio principale viene eseguito. Può essere invocato un servizio dello stesso o di un qualsiasi altro domoDevice. L'associazione di servizi non ha vincoli semantici, ovvero è possibile combinare un servizio con qualsiasi altro a patto di riuscire a dare dei valori ragionevoli ai linkedInput (discusso successivamente). Questo tag è il cuore della cooperazione tra dispositivi reali eterogenei. I suoi attributi sono:

  - URL: l'identificativo del web service che gestisce il domoDevice da collegare al servizio;
  - id: l'identificativo del domoDevice da collegare al servizio all'interno del web service ;
  - service: il nome del servizio del domoDevice individuato dai precedenti attributi da collegare;

4. in base al tag precedente, a questo livello è possibile avere:

   - allowed: se il tag precedente è input. Rappresenta un possibile valore che può assumere l'input. Questo parametro è opzionale e un input può avere più tag allowed; l'unico attributo per questo tag è:
     - value: il valore in formato stringa ammesso.
   - linkedInput: se il tag precedente è linkedService. Rappresenta l'associazione di un input del presente servizio con un input del servizio da richiamare. Ogni input del servizio da richiamare deve avere un'associazione. In questo modo, il valore dell'input del presente servizio viene passato come input al servizio da richiamare. Gli attributi per questo tag sono:
     - from: il nome dell'input dal quale prendere il valore;
     - to: il nome dell'input che deve usare il valore richiamato;

Codice 3.1: Esempio di domoDevice: una lampadina

```
<device description="lampada_a_risparimo_energetico" id="0" manufacturer="pholips"
    positionDescription="comodino_accanto_al_letto" serialNumber="xxxxxxxxx"
    tech="KNX" type="lampada" url="http://www.questowebservice.it/servizio">
      <service description="Get_the_status" output="BOOLEAN"
          outputDescription="The_value" name="GET_STATUS" prettyName="Get_
          status" />
      <service description="Set_the_status" name="SET_STATUS" prettyName="Set_
          status">
          <input description="The_value" name="status" type="BOOLEAN">
                <allowed value="TRUE" />
                <allowed value="FALSE" />
          </input>
          <linkedService id="3" service="setPower"
              url="http://www.altrowebservice.it/servizio">
                <linkedInput from="status" to="power" />
          </linkedService>
      </service>
</device>
```

Dal codice 3.1 si nota che il domoDevice descritto è una lampada posta nella camera da letto. Essa è stata prodotta dalla Pholips, usa la tecnologia Konnex e fa parte della categoria lampada. E' comandata dal web service

http://www.questowebservice.it/servizio

I servizi offerti dalla lampada sono due: prendere il suo stato corrente in formato booleano e settare il suo stato attraverso un input, sempre booleano, che può assumere i valori TRUE o FALSE. Quando viene invocato quest'ultimo servizio, ne viene chiamato anche un altro di nome setPower appartenente al domoDevice gestito dal web service

http://www.altrowebservice.it/servizio

con id 3. Il valore dell'input assegnato al primo servizio (status) viene assegnato anche al secondo (power). Se viene invocato quindi il servizio setStatus del domoDevice con url=http://www.questowebservice.it/servizio e con id=0 passando come valore di input TRUE, viene anche invocato il servizio setPower del domoDevice con url=http://www.altrowebservice.it/servizio, con id=3 e con valore di input TRUE.

3.5.2 domoMessage

Il domoMessage ha lo scopo di creare un meccanismo semplice, compatto ed efficace per rappresentare le interazioni tra i domoDevice astraendo dalle tecnologie.Anche il domoMessage può essere rappresentato attraverso un albero largo e poco profondo (si usa una grammatica altamente attributata e con pochi figli). L'albero ha al massimo 2 livelli:

1. message: apre il tag che descrive il domoMessage e contiene i seguenti attributi:
    - message: il corpo del messaggio, tipicamente il nome del servizio da invocare;
    - messageType: il tipo di messaggio. Può essere:
        - COMMAND: se trattasi di servizio da eseguire;
        - SUCCESS: se richiesto l'esito successivo al COMMAND. In questo caso il campo message pu'o contenere il valore di risposta dell'esecuzione del servizio;
        - FAILURE: se richiesto l'esito successivo al COMMAND. In questo caso il campo message può contenere il codice di errore del fallimento;
    - receiverId: l'identificatore del domoDevice ricevente del messaggio;
    - receiverURL: l'url del web service che contiene il domoDevice richiesto;
    - senderId: l'identificatore del domoDevice mittente del messaggio;
    - senderURL: l'url del web service che contiene il domoDevice che invia il messaggio;
2. input: valori di input da associare al message. Questo tag è opzionale e ogni tag message può avere più tag input. Gli attributi per questo tag sono:
    - name: il nome dell'input;

- type: il DomoDevice.DataType dell'input;
- value: il valore in formato stringa dell'input;

Codice 3.2: Esempio di domoMessage: setStatus

```
<message message="SET_STATUS" messageType="COMMAND" receiverId="1"
    receiverURL="http://www.altrowebservice.it/servizio" senderId="0"
    senderURL="http://www.questowebservice.it/servizio">
      <input name="status" type="BOOLEAN" value="TRUE" />
</message>
```

In questo esempio viene richisto di eseguire il servizio SET_STATUS sul domoDevice con web service:

http://www.questowebservice.it/servizio

e id 1 con input di tipo booleano dal valore TRUE.

Eseguito il servizio, è possibile ricevere il seguente messaggio di conferma:

Codice 3.3: Esempio di domoMessage: conferma

```
<message message="TRUE" messageType="SUCCESS"
    receiverURL="http://www.questowebservice.it/servizio" receiverId="0" senderId="1"
    senderURL="http://www.altrowebservice.it/servizio" />
```

Questo messaggio dà la conferma al mittente che il precedente messaggio è stato eseguito con successo e che la lampada è ora accesa.

Capitolo 4

Il Progetto

Il progetto prevede la progettazione e lo sviluppo di un modulo software per l'implementazione di un gateway che, adottando gli emergenti paradigmi dell'Ambient Intelligence, permette ai dispositivi domotici di divenire compatibili con gli obiettivi dell'Internet Of Things. L'applicativo prevede l'introduzione del supporto del protocollo IPv6 ad un esistente progetto, sviluppato dal laboratorio di domotica dell'ISTI – CNR di Pisa, chiamato DomoNet, per la realizzazione dell'interoperabilità tra tutti i dispositivi domotici anche conformi a sistemi o produttori diversi. Viene utilizzato il linguaggio di programmazione Java in ambiente di sviluppo Linux.

Il lavoro trova le soluzioni alle seguenti problematiche:

- implementare il meccanismo di supporto del protocollo IPv6 ai dispositivi domotici della rete DomoNet;
- associare i dispositivi della rete DomoNet con indirizzi del protocollo IPv6, attraverso la realizzazione di un meccanismo di mapping;
- interfacciare l'applicazione web con DomoNet, utilizzando tecnologie di comunicazione standard e aperte come il paradigma dei web services e XML;
- realizzare una interfaccia web secondo il paradigma MVC (Model View Controller).

L'interazione con il dispositivo avviene attraverso un'applicazione web che, prendendo in input l'indirizzo IPv6 di un dispositivo, ne visualizza lo stato e le relative funzionalità invocabili.

4.1 Implementare IPv6

La scelta di assegnare indirizzi IPv6 è nata dall'esigenza di portare le reti domotiche verso l'Internet delle Cose. Ovvero un giorno vorremmo che due dispositivi da parti diverse del mondo possano dialogare direttamente tra loro. Per realizzare ciò serve che i due dispositivi abbiano un indirizzo IP pubblico (globale) preciso. Data la vasta disponibilità di indirizzi IPv6 la scelta è stata ovvia.

Il problema però rimane: come posso assegnare un indirizzo IPv6 ad un dispositivo che non ha interfacce di rete compatibili con esso? Le strade possibili sono due:

- Realizzare una NIC (Network Interface Card) per i dispositivi domotici, compatibile con IPv6
- Trovare una maniera alternativa

La NIC per quanto sia una soluzione didatticamente valida, ha alcuni problemi: poco scalabile e sicuramente non di nostra competenza. Infatti se le case produttrici volessero dotare questi dispositivi di tale interfaccia lo avrebbero già fatto e inserire per ogni dispositivo domotico una scheda di rete, risulterebbe noioso già per piccole reti domotiche.

Dobbiamo pensare ad una soluzione migliore. L'idea è stata quella di realizzare una corrispondenza, a livello software, tra il dispositivo (che a questo livello è un dispositivo di DomoNet) con un indirizzo IPv6.

Il ragionamento è simile a quello del NAT ma concettualmente diverso:

- abbiamo un server dove gira domoNet;
- ad un'interfaccia del server assegniamo N indirizzi IPv6 globali;

- ad ogni dispositivo che si registra in domoNet viene assegnato uno di questi indirizzi;
- ogni richiesta fatta a tale indirizzo, verrà inoltrata al dispositivo corretto;

4.2 La "rete" di test del progetto

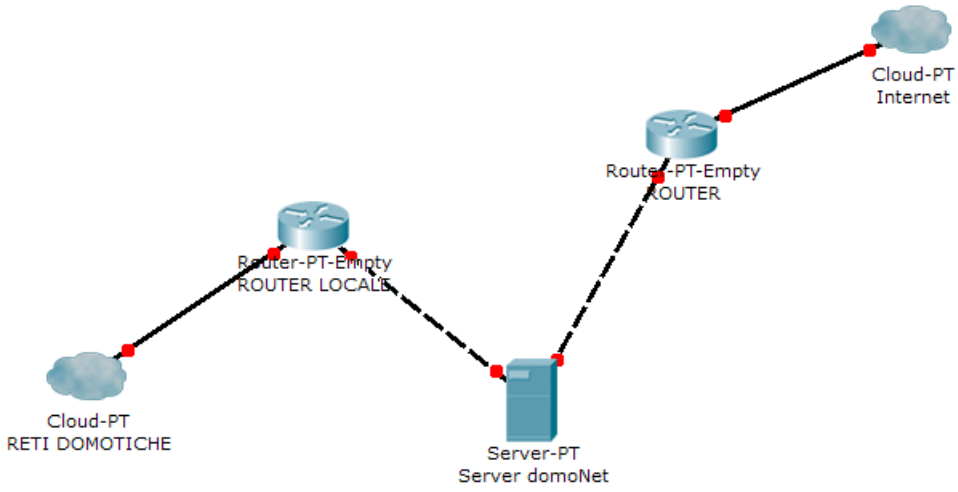

Figura 4.1: Schema della rete del laboratorio di domotica

La rete è molto semplice, abbiamo un router locale che gestisce la rete domotica che è collegato al server domoNet, che provvederà a omogeneizzare le diverse tipologie di sistemi domotici presenti. Abbiamo un router di frontiera, che collega il server ad Internet. Il server ha due interfacce di rete:

- una configurata con indirizzi locali
- una configurata con indirizzi pubblici

Il server domoNet è un server che supporta il dual stack, ovvero sceglie a seconda delle occorrenze se usare il protocollo IPv4 o il protocollo IPv6. Infatti la rete locale comunica solo in IPv4: il *ROUTER LOCALE* non ha la compatibilità con IPv6. La rete "pubblica", quella che consente la raggiungibilità su Internet, supporta tutti e due gli indirizzamenti.

4.3 Meccanismo di Mapping

Il meccanismo di mapping avviene all'interno del server domoNet. Al server arriva una richiesta da internet per un determinato indirizzo IPv6. Tale indirizzo viene confrontato all'interno di una *tabella delle corrispondenze*, dove avremo:

- Indirizzo IPv6
- Indirizzo dispositivo domoNet

Il server riconoscerà il dispositivo domoNet corrispondente e comunicherà con esso attraverso la rete locale.

4.4 Ant, Axis, Tomcat

Per la realizzazione di questo progetto gli strumenti usati sono:

- Apache Ant
- Apache Axis
- Apache Tomcat

Ant è uno strumento simile al *makefile* usato per il linguaggio C, molto utile in fase di compilazione del progetto. Infatti permette, tramite un file di configurazione XML, di impostare varie fasi del processo di compilazione e attraverso dei *task* che possono essere:

- javac
- java

, dove è possibile specificare, ad esempio, il classpath, ottenere i file .class in maniera semplice ed efficace.

Tomcat è il web server. La scelta di questa tipologia di web server al posto del più conosciuto e tradizionale Apache è dovuta al fatto che supporta

nativamente le applicazioni Java. Dato che domoNet nasce con questo linguaggio di programmazione, ci è sembrato, quasi naturale, utilizzare questo tipo di server.

Axis è uno strumento che gira sotto Tomcat utile nella realizzazione e nella gestione dei webservices. domoNet già usava questo tool per la parte server, noi lo sfruttiamo per una veloce realizzazione della parte client.

4.5 Realizzazione del Mapping

Vediamo come è stato realizzato in pratica quanto detto sopra.

4.5.1 Assegnazione indirizzi

Per gestire le interfacce di rete di un sistema Unix, nel nostro caso Debian, il comando più al passo con i tempi è senz'altro il comando *"ip"*. Infatti, il più famoso *ifconfig* sembra non essersi ancora adattato al 100% ad IPv6 e non lo supporta benissimo. Il comando *ip* è abbastanza comodo da utilizzare. Ad esempio se volessimo visualizzare tutti gli indirizzi IPv6 assegnati ad una certa interfaccia (in questo caso *eth0*) ci basterebbe digitare:

Codice 4.1: Comando ip: show ipv6 addresses

```
ip -6 addr show dev eth0
```

Per assegnare un indirizzo IPv6 ad un'interfaccia dobbiamo usare la notazione con indicazione del numero di bit di prefisso, ad esempio:

Codice 4.2: Comando ip: add ipv6 addresses

```
ip -6 addr add 2a00:1620:c0:50::3/64 dev eth0
```

Come vediamo possiamo assegnare qualsiasi tipo di indirizzo all'interfaccia (globale, locale), sarà poi il sistema operativo a gestire l'effettiva assegnazione dell'indirizzo. Questo vuol dire che se, accidentalmente, assegnassimo un indirizzo "duplicato", il sistema operativo richiamando il protocollo NDP farà il Duplicate Address Detection e in caso negativo ci restituirà un errore. Possiamo, quindi, assegnare un set di indirizzi IPv6 completamente casuali

all'interfaccia senza preoccuparci troppo che tali indirizzi non vadano bene o siano duplicati, abbiamo la garanzia che quelli che saranno effettivamente assegnati all'interfaccia siano indirizzi "buoni".

Possiamo iniziare il nostro script bash, con un ciclo da 1 a un valore soglia scelto dall'utente da riga di comando per assegnare un set di indirizzi IPv6 ad un'interfaccia del nostro server domoNet. Resta un altro problema, qual'è il prefisso globale che ci manda il router? Anche qua il sistema operativo ci facilita notevolmente il lavoro. Infatti, appena la scheda di rete viene collegata ad un router sempre grazie al protocollo NDP vengono scambiate le informazioni essenziali per la comunicazione sulla rete, tra le quali il prefisso globale degli indirizzi con cui uscire su Internet.

Dove possiamo trovarlo? Il comando ip permette di visualizzare la tabella di "routing" dell'host, lì sicuramente troveremo una riga dove è scritto il prefisso globale per comunicare su Internet. Il comando per visualizzare la tabella di routing è:

Codice 4.3: Comando ip: route

```
ip -6 route
```

Il comando *egrep*, una versione di grep estesa, e le espressioni regolari ci vengono in aiuto per isolare il prefisso dall'output di questo comando in modo tale da poterlo utilizzare per la generazione automatica degli indirizzi da assegnare all'interfaccia scelta dall'utente da riga di comando.

4.5.2 Associazione dei dispositivi - primo approccio

A questo punto abbiamo un'interfaccia del server domoNet che comunica su Internet alla quale sono stati assegnati N indirizzi IPv6 globali. Per un dispositivo dall'altra parte di Internet questi indirizzi non devono essere altro che gli indirizzi di un dispositivo domotico, all'interno della rete domoNet. Sorge dunque l'esigenza di associare ognuno di questi indirizzi ad un relativo dispositivo domotico.

Possiamo realizzare un client in Java, che si interfacci, tramite i web services, a domoNet per capire quali dispositivi siano effettivamente connessi e realizzi un'associazione 1 a 1.

4.5.3 Bind9: il DNS

Ora che abbiamo associato ciascun indirizzo IPv6 ad un dispositivo domotico ben preciso, il lavoro potrebbe concludersi qua, ma se vogliamo fare in modo che un utente possa modificare da qualsiasi parte del mondo lo stato del proprio dispositivo è impensabile lasciare l'accesso ad esso solo tramite la digitazione dell'indirizzo IPv6. Una persona non potrà mai ricordare svariati gruppi di cifre esadecimali. La soluzione migliore è quella di configurare un semplice server DNS in modo tale da assegnare un nome simbolico al dispositivo.

Il nome scelto sarà ricavato, dopo un opportuno filtraggio per i caratteri "speciali" (spazi, chiocciole, ampersand, etc), dall'attributo *descrizione* del domoDevice.

Il server DNS scelto per implementare questa funzionalità è *bind9*.

Panoramica su bind9

Bind9 è un DNS server che gira in ambiente Unix-like, nel nostro caso Debian. Ci sono diversi metodi per configurare BIND9. Alcune delle configurazioni più comuni consistono in un server dei nomi cache, un server primario e un server secondario.

- Quando configurato come un server dei nomi cache, BIND9 troverà la risposta alle interrogazioni sui nomi e la archivierà.
- Come server primario, BIND9 legge i dati per una zona da un file ed è autoritativo per quella zona.
- Nella configurazione come server secondario, BIN9 ottiene i dati della zona da un altro server dei nomi per quella zona.

I file di configurazione di DNS sono archiviati nella directory "/etc/bind", il file di configurazione principale è "/etc/bind/named.conf".

La riga *include* specifica il nome del file contenente le opzioni DNS, la riga directory nel file "/etc/bind/named.conf.options" indica a DNS dove cercare i file. Tutti i file usati da BIND sono presenti in questa directory.

Il file "/etc/bind/db.root" descrive i server dei nomi radice nel mondo. Questi server cambiano col tempo, quindi il file "/etc/bind/db.root" deve essere aggiornato ogni tanto, procedura che viene svolta, solitamente, con gli aggiornamenti al pacchetto bind9. La sezione *zone* definisce un server principale ed è archiviata in un file indicato dall'opzione *file*.

È possibile configurare un server dei nomi come server cache, server primario o secondario. Un server può essere lo Start of Authority (SOA) per una zona, fornendo al tempo stesso servizi di server secondario per un'altra e servizi di cache per gli host nella rete locale.

Configurare bind9: server dei nomi cache

La configurazione predefinita comporta l'utilizzo come server di cache. È necessario solamente aggiungere gli indirizzi IP dei server DNS del proprio ISP (abbiamo utilizzato il DNS di Google). De-commentare e modificare quanto segue nel file "/etc/bind/named.conf.options":

Codice 4.4: BIND9: Caching

```
        forwarders {
                8.8.8.8;
           };
```

Ogni volta che non troverà una corrispondenza all'interno dei propri file di zona, "forwarderà" la richiesta ai server specificati all'interno del blocco *forwarders*

Configurare bind9: server primario

In questa sezione, BIND9 viene configurato come server primario per il dominio *localhost.prova*.

File zona forward Per aggiungere una zona DNS a BIND9, trasformando BIND9 in un server primario, la prima cosa da fare è modificare il file "/etc/bind/named.conf.local":

Codice 4.5: BIND9: configurazione zona primario forward

```
zone "localhost.prova" {
```

```
        type master;
        file "/etc/bind/db.example.com";
};
```

Il file di zona sarà:

Codice 4.6: BIND9: file zona primario forward

```
$TTL 3h
$ORIGIN localhost.prova.

@ IN SOA ns1 hostmaster (
                2008012400 ; Serial
                10H ; refresh
                3H ; retry
                1W ; expire
                1D ) ; minimum
        IN NS ns1

;database risolutivo

ns1  IN AAAA 2a00:1620:c0:50:20b:6aff:fe82:6ecb
$INCLUDE /etc/bind/db.dispositivi ;includo i file dns generati automaticamente
```

L'ultima riga è quella interessante: include un file di zona, generato automaticamente dal programma Java di associazione degli indirizzi. E' quello che rende "dinamico" il server DNS in base ai dispositivi connessi.

File zona reverse Una volta configurata la zona e la risoluzione dei nomi con un indirizzo IP, è necessaria anche una zona Reverse. Una zona Reverse consente a DNS di trasformare un indirizzo in un nome.

Per fare ciò occorre modificare il file "/etc/bind/named.conf.local" aggiungendo quanto segue:

Codice 4.7: BIND9: configurazione zona primario reverse

```
zone "0.5.0.0.0.c.0.0.0.2.6.1.0.0.a.2.ip6.arpa"
{
        type master;
        file "/etc/bind/pz/0.5.0.0.0.c.0.0.0.2.6.1.0.0.a.2.ip6.arpa";
};
```

Tale file conterrà:

Codice 4.8: BIND9: file zona primario reverse

```
$ORIGIN 0.5.0.0.0.c.0.0.0.2.6.1.0.0.a.2.ip6.arpa.
$TTL 604800
@ IN SOA ns1.localhost.prova. root.localhost.prova. (
                                1 ; Serial
                           604800 ; Refresh
                            86400 ; Retry
                          2419200 ; Expire
                           604800 ) ; Negative Cache TTL

        NS ns1.localhost.prova.
b.c.e.6.2.8.e.f.f.f.a.6.b.0.2.0 PTR ns1.localhost.prova.
$INCLUDE /etc/bind/reverse.dispositivi ;includo i file dns generati automaticamente
```

Come precedentemente scritto per la configurazione forward, il file incluso nell'ultima riga serve per dare dinamicità al DNS.

4.5.4 Associazione dei dispositivi - secondo approccio

Vediamo come aggiungere al ragionamento fatto per il primo approccio, la mappatura completa di DNS.

Abbiamo un'interfaccia del server domoNet che comunica su Internet alla quale sono stati assegnati N indirizzi IPv6 globali. Possiamo realizzare un client in Java, che si interfacci, tramite i web services, a domoNet per capire quali dispositivi siano effettivamente connessi.

Il programma, in primis, leggerà tutti gli indirizzi effettivamente assegnati all'interfaccia scelta. Successivamente, facendo delle richieste tramite i webservices, riceverà da domoNet tanti domoDevice quanti sono i dispositivi connessi alla rete domotica. Per ogni dispositivo connesso dovrà fare l'associazione con un indirizzo dell'interfaccia. Infine andrà a scrivere il risultato delle associazioni in due file di configurazione del DNS, uno forward e uno reverse. Se non arrivano richieste di domoDevice per un lasso di tempo di 1 minuto, viene riavviato il server DNS che renderà effettive tutte le modifiche contenute nei file di configurazione scritti precedentemente.

DomoBeans

I DomoBeans sono delle classi che realizzano quanto viene letto dal domoML della rete domotica. Si rifanno al concetto dei JavaBean che saranno spiegati

più avanti. Avremo quindi le seguenti classi:

- Devices: che conterrà tutti i Device
- Device: che conterrà vari Service e gli attributi del dispositivo
- Service: che avrà o meno degli Input e gli attributi del servizio
- Input: che avrà o meno dei valori possibili (Allowed)
- Allowed: il valore possibile

Queste classi vengono realizzate in maniera intelligente attraverso un "meta-parser" XML, chiamato Digester. Digester prende in input un file di configurazione XML e genera i vari Bean automaticamente parsando il file DomoML.

Codice 4.9: Esempio di file di configurazione (Digester-rules) per Digester: parsing del domoML

```
<?xml version="1.0"?>
<!DOCTYPE digester-rules PUBLIC
  "-//Apache Commons //DTD digester-rules XML V1.0//EN"
  "http://commons.apache.org/digester/dtds/digester-rules-3.0.dtd">
<digester-rules>
    <pattern value="devices">
        <object-create-rule classname="com.domoBeans.Devices" />
        <set-properties-rule/>
        <pattern value="device">
            <object-create-rule classname="com.domoBeans.Device" />
            <set-properties-rule>
                <alias attr-name="id" prop-name="id"/>
                <alias attr-name="manufacturer"/>
                <alias attr-name="position"/>
                <alias attr-name="positionDescription"/>
                <alias attr-name="serialNumber"/>
                <alias attr-name="tech"/>
                <alias attr-name="type"/>
                <alias attr-name="url" prop-name="url"/>
                <alias attr-name="description" prop-name="nome"/>
            </set-properties-rule>
            <pattern value="service">
                <object-create-rule classname="com.domoBeans.Service" />
                <set-properties-rule>
                    <alias attr-name="description"/>
                    <alias attr-name="name" prop-name="serviceName"/>
```

```
                    <alias attr-name="output" prop-name="output"/>
                    <alias attr-name="outputDescription"/>
                    <alias attr-name="outputName" prop-name="outputName"/>
                    <alias attr-name="prettyName" prop-name="nome"/>
                </set-properties-rule>
                <pattern value="input">
                    <object-create-rule classname="com.domoBeans.Input" />
                    <set-properties-rule>
                        <alias attr-name="description"/>
                        <alias attr-name="name" prop-name="nome"/>
                        <alias attr-name="type" prop-name="tipo"/>
                    </set-properties-rule>
                    <pattern value="allowed">
                        <object-create-rule classname="com.domoBeans.Allowed" />
                        <set-properties-rule>
                            <alias attr-name="value" prop-name="valore"/>
                        </set-properties-rule>
                        <set-next-rule methodname="addValore" />
                    </pattern>
                    <set-next-rule methodname="addInput" />
                </pattern>
                <pattern value="linkedService">
                    <pattern value="linkedInput">
                    </pattern>
                </pattern>
                <set-next-rule methodname="addServizio" />
            </pattern>
            <set-next-rule methodname="addDevice" />
        </pattern>
    </pattern>
</digester-rules>
```

Ad ogni "pattern", che corrisponde ad un tag XML, viene associato un oggetto tramite il tag *object-create-rule*, successivamente vengono mappati gli attributi XML con gli attributi dell'oggetto tramite i tag *alias*.
Ovviamente, essendo dei Bean, tutti gli attributi di queste classi vengono acceduti tramite metodi Getter e Setter.

ipAssigner

Una volta realizzati i domoBeans, che sono indispensabili per questo progetto, si è realizzata una classe che gestisce tutto il processo di mapping: dalla lettura degli indirizzi dell'interfaccia scelta, alla scrittura dei file di configu-

razione e all'aggiornamento degli stessi. Un oggetto ipAssigner per essere istanziato deve conoscere:

- Interfaccia pubblica del server, dalla quale leggere gli indirizzi IPv6 globali
- Una lista di dispositivi (Device) da mappare
- I nomi dei file di configurazione (forward e reverse) del DNS
- Il nome della zona DNS da configurare

La classe ipAssigner offre due metodi per agire sui file di configurazione DNS:

- boolean writeConfigFiles() - crea i due file di configurazione DNS in base a una tabella di mapping costruita all'interno dell'oggetto ipAssigner. Restituisce *true* in caso di corretta scrittura dei file, *false* altrimenti.
- boolean insertDevice(String) - aggiorna i file di configurazione DNS in base al nome, passato come parametro, del dispositivo aggiunto. Restituisce *true* in caso di corretta scrittura dei file e se il dispositivo non è già presente, *false* altrimenti.

TimerHandler

La necessità di riavviare il server DNS, solamente dopo una lunga attesa tra l'arrivo di un device e un altro, è nata dalla ricerca di un *trade-off* tra l'esigenza di avere il sistema aggiornato (che richiederebbe idealmente un riavvio ogni device che si collega alla rete domotica) e l'esigenza di non sovraccaricare troppo il server con continui riavvii. L'attesa tra l'arrivo di un device e un altro, prima che si effettui il riavvio del DNS è configurabile da riga di comando. La classe che gestisce il timer, in modo tale che scaduto il quanto di tempo impostato si riavvii il server DNS è la classe TimerHandler.

Questa è stata sviluppata in modo tale che l'utilizzatore della classe possa personalizzare l'azione da compiere una volta scaduto il quanto di tempo. Per istanziare un oggetto di questo tipo bisogna avere:

- Un oggetto Runnable che specifichi le azioni da compiere allo scadere del quanto di tempo

- Il quanto di tempo

La classe offre due metodi:

- void doStart() - avvia il timer

- void doStop() - arresta il timer

ManageWS

La classe ManageWS si occupa della gestione della comunicazione con i WebServices offerti da domoNet. Un oggetto di questa classe per essere istanziato deve conoscere:

- l'URL del server che offre i WebServices

Questa classe provvederà a collegarsi al server di domoNet e a fornire un'interfaccia per comunicare con i suoi servizi in maniera facile e intuitiva, nascondendo la complessità della connessione ai WebServices. Offre i seguenti metodi:

- void connectToWebServices() - si collega ai servizi domoNet e legge i dispositivi connessi alla rete

- Devices getDisp() - restituisce un DomoBean Devices indicante la topologia completa della rete domotica

- void registerTCPServer(int) - prende come parametro la porta dove è in ascolto un server TCP, e lo registra per ricevere gli aggiornamenti

- String execute(DomoMessage) - prende come parametro un domoMessage da eseguire (di tipo COMMAND). Restituisce *FAILURE* se qualcosa è andato male, altrimenti l'output dell'esecuzione del servzio.

- String executeService(Device, Service, Vector<Input>, Vector<String>) - prende come parametro un dispositivo, un suo servizio, una lista di input e i vari valori scelti per tali input. Crea un domoMessage ed esegue la execute().

Programma principale

Il programma principale utilizza le classi sopraelencate per realizzare il meccanismo di mapping descritto nei paragrafi precedenti. Questo programma compie le seguenti azioni:

- Legge da riga di comando i parametri di configurazione (indirizzo del servizio, porta, nomi dei file, etc...)
- Si collega ai WebServices di domoNet tramite la classe ManageWS
- Legge tutti i dispositivi connessi e realizzerà il mapping tramite la classe ipAssigner
- Crea un Thread che manderà in esecuzione un server TCP, per mappare eventuali dispositivi che si collegheranno alla rete
- Registra il server creato tramite la funzione registerTCPServer per ricevere gli aggiornamenti da domoNet
- Attende che il server abbia completato l'esecuzione

Codice 4.10: Mapping: Programma principale

```
import domoutils.TCPServer;
import domoutils.ipAssigner;
import com.domoBeans.*;
import common.*;
import java.util.Vector;
import java.lang.Integer;

public class esegui {

    /*
        posso impostare gli argomenti del main per configurare:
            - args[0] url, del servizio
```

```
            - args[1] porta, dove deve girare il server tcp
                - args[2] iface, interfaccia scelta
                - args[3] zoneName, nome della zona dns
                - args[4] filename_f, nomefile conf.dns forward
                - args[5] filename_r, nomefile conf.dns reverse
                - args[6] delay, tempo che intercorre tra due segnalazioni di device
                                    prima del riavvio DNS (in millisecondi)
    */
    public static void main(String[] args)
    {
        try {
            //controllo che ci siano tutti i parametri
            //di configurazione
            if (args.length != 7)
                throw new Exception();

            String serviceURL = args[0];
            int porta = Integer.parseInt(args[1]);
            String iface = args[2];
            String zoneName = args[3];
            String filename_f = args[4];
            String filename_r = args[5];
            int delay = Integer.parseInt(args[6]);

            //ManageWS e' la classe che si occupa del dialogo con i webservices
            //il costruttore prende l'url del server
            ManageWS manager = new ManageWS(serviceURL);

            //Scarichiamo da domoNet tutti i domoDevice dei dispositivi connessi
            manager.connectToWebServices();
            //Automaticamente vengono riempite delle classi Bean in base
            //al contenuto del domoML, in base al file di configurazione
            //domodevicerules.xml
            Devices disps = manager.getDisp();
            Vector disp_array = disps.getDevices();
            //Si scorrono tutti i dispositivi connessi alla rete domoNet
            for (int i = 0; i < disp_array.size(); i++)
            {
                Device temp = (Device) disp_array.elementAt(i);
                //vengono stampati a video, come conferma della corretta lettura
                System.out.println("Dispositivo: " + temp.getNome() );
            }

            //ipAssigner e' la classe che si occupa del mapping dei dispositivi
            //con indirizzi IPv6 assegnati ad una certa interfaccia
            ipAssigner mapper = new ipAssigner(iface, disp_array, filename_f, filename_r,
                zoneName);
            //vengono scritti i file di configurazione DNS, forward e reverse
            mapper.writeConfigFiles();
```

```
                //Creo un server TCP che si metta in ascolto su una certa porta
                //dovra' ricevere gli aggiornamenti (in questo caso nuovi device)
                // dalla rete domoNet
                Runnable myserver = new TCPServer(porta, mapper, delay);
                Thread server = new Thread(myserver);
                server.start();
                //questa funzione serve a registrare il nostro server
                //all'interno di domoNet per ricevere gli aggiornamenti
                manager.registerTCPServer(porta);

                //attendo che il Thread creato finisca
                server.join();
            } catch (Exception e)
            {
                System.out.println(e.toString());
            }
        }
}
```

Il server TCP

Il server TCP, semplicemente, si mette in ascolto su una porta. Stabilisce la connessione con domoNet e riceve delle String di aggiornamento (o domoMessage o domoDevice). All'arrivo di un domoDevice lo aggiunge al mapper (un oggetto ipAssignment).

Codice 4.11: Mapping: TCPServer

```
package domoutils;

import timing.TimerHandler;
import java.net.*;
import java.io.*;
import common.*;
import java.lang.*;
import domoML.domoDevice.*;

/**************************************************************

 class TCPServer

      classe che implementa l'interfaccia Runnable
      dara' vita a un Thread, che fara' da server TCP
      in ascolto su una certa porta per ricevere
      gli aggiornamenti dalla rete domotica
```

```
**************************************************************/
public class TCPServer implements Runnable {

    private Socket socket;
    private ServerSocket serverSocket;
    private BufferedReader is;
    private DataOutputStream os;
    private int porta;
    private ipAssigner mapper;
    private int delay;

    //Classe che mi serve per definire il comportamento
    //che dovra' eseguire il Task che andra' in esecuzione
    //ogni volta che scade il timer (in questo caso riavvio DNS)
    class MyTask implements Runnable {

        @Override
        public void run()
        {
            try {
                Process pr = Runtime.getRuntime().exec("sudo /etc/init.d/bind9 restart");
                pr.waitFor();
                BufferedReader buf = new BufferedReader(new
                    InputStreamReader(pr.getInputStream()));
                String line = "";
                while ((line=buf.readLine())!=null) {
                    System.out.println(line);
                }
            } catch (Exception e)
            {
                System.out.println(e.toString());
            }
        }

    }

    public TCPServer (int porta, ipAssigner mapper, int delay)
    {
        this.porta = porta;
        this.mapper = mapper;
        this.delay = delay;
    }

    public void run() {
        try {
            serverSocket = new ServerSocket(porta);
            //Informazioni sul Server in ascolto
            InetAddress indirizzo = serverSocket.getInetAddress();
            String server = indirizzo.getHostAddress();
            int port = serverSocket.getLocalPort();
```

```
Debug.writeln("Initializing update socket on " + server + ":" + port);
TimerHandler myTime = new TimerHandler(new MyTask(), delay);
myTime.doStart();
//Ciclo infinito per ascolto dei Client
while (true) {
    socket = serverSocket.accept();
    //Informazioni sul Client che ha effettuato la chiamata
    InetAddress address = socket.getInetAddress();
    String client = address.getHostName();
    int porta = socket.getPort();
    Debug.writeln("Opened update socket with: " + client + ":" + porta);

    //Stream di byte utilizzato per la comunicazione via socket
    is = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    //os = new DataOutputStream(socket.getOutputStream());
    try {
        while (true) {
            String inputLine = is.readLine();
            if(inputLine != null) {
                    //devo verificare che sia un domodevice
                    //leggere il campo description
                    //se e' un device riavviare il timer
                    //se e' un device fai insertDevice(dev)
                try {
                    DomoDevice temp = new DomoDevice(inputLine);
                    if (temp.getDescription().length() != 0)
                    {
                        //e' un device, faccio ripartire il timer
                        myTime.doStop();
                        myTime.doStart();
                        //lo vado ad inserire nelle configurazioni
                        boolean ok =
                            mapper.insertDevice(temp.getDescription());
                        if (!ok)
                            Debug.writeln("Error while adding device: " +
                                temp.getDescription() );
                    }
                } catch (Exception e)
                {
                    //non e' un device (non ha getDescription e lancia l'eccezione)
                }
                Debug.writeln(inputLine);
            }
        }
    } catch (Exception e) {
        terminate();
    }
}
} catch(IOException e) {
    Debug.writeln("Closed update socket");
```

```
            }
        }

        public void terminate() {
            try {
                is.close();
                os.close();
                socket.close();
                serverSocket.close();
            } catch(IOException e) {
                e.printStackTrace();
            }
        }
}
```

Script bash

Per completare il meccanismo di mapping è stato realizzato un semplice script bash, che compie le seguenti azioni:

- Legge il prefisso globale assegnato, tramite NDP, dal router all'interfaccia del server (con il comando route)
- Genera un set di N indirizzi, con il prefisso appena trovato, e li assegna all'interfaccia del server
- Avvia il programma java che si occuperà dell'associazione dei dispositivi agli indirizzi assegnati all'interfaccia

4.6 L'interfaccia web

Vogliamo realizzare una semplice interfaccia web, che risponda all'indirizzo del dispositivo con le sue funzioni e permetta di interagire con esse.

4.6.1 MVC

L'architettura MVC è un insieme di regole per strutturare un sito-web dinamico. Queste regole complicano la struttura del sito, ma promettono di semplificare le operazioni successive di mantenimento e aggiornamento.

L'architettura MVC si basa su tre componenti principali:

- Model (M): rappresenta lo stato della nostra applicazione web (database, file, informazioni di sessione, etc.)
- View (V): sono una serie di viste interfacce utente del modello
- Controller (C): una serie di applicazioni che racchiudono la logica applicativa del nostro sito, attraverso delle modifiche al modello.

La parte Model rappresenta lo stato della nostra applicazione web.

Viene realizzato tramite una serie di Javabeans che incapsulano delle risorse (Database, files, risorse di rete, informazioni di sessione, informazioni delle richieste http, etc.). Il punto di forza di incapsulare tutte le risorse dentro JavaBeans è quello di astrarre l'accesso a diverse risorse tramite metodi set e get di alto livello. La parte View è una serie di viste del modello.

Questa parte è tipicamente implementata tramite JSP che accedono ai Javabeans del Modello.

La parte Controller è una serie di applicazioni che modificano il modello. Questa parte è tipicamente realizzata tramite Servlet o JSP.

4.6.2 JavaBean

Le JavaBean (letteralmente, chicchi di Java) sono classi scritte in linguaggio di programmazione Java secondo una particolare convenzione. Sono utilizzate per incapsulare più oggetti in un oggetto singolo (il bean), cosicché tali oggetti possano essere passati come un singolo oggetto bean invece che come multipli oggetti individuali. La specifica della Sun Microsystems le definisce come componenti software riutilizzabili che possono essere manipolate visivamente in un tool per il build.

Convenzioni

Al fine di funzionare come una classe JavaBean, una classe di un oggetto deve obbedire a certe convenzioni in merito ai nomi, alla costruzione e al comportamento dei metodi. Queste convenzioni rendono possibile avere tool che possono usare, riusare, sostituire e connettere JavaBean. Le convenzioni richieste sono:

- La classe deve avere un costruttore senza argomenti;
- Le sue proprietà devono essere accessibili usando get, set, is (usato per i booleani al posto di get) e altri metodi (così detti metodi accessori) seguendo una convenzione standard per i nomi;
- La classe dovrebbe essere serializzabile (capace di salvare e ripristinare il suo stato in modo persistente);
- Non dovrebbe contenere alcun metodo richiesto per la gestione degli eventi;

Codice 4.12: Esempio di JavaBean

```
// PersonaBean.java

public class PersonaBean implements java.io.Serializable {

    private String nome;
    private boolean sposata;

    // Costruttore senza argomenti
    public PersonaBean() { }

    public String getNome() {
        return this.nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }

    // Diversa sintassi per gli attributo boolean ( 'is' al posto di 'get' )
    public boolean isSposata() {
        return this.sposata;
    }
    public void setSposata(boolean sposata) {
        this.sposata = sposata;
    }
}
```

4.6.3 Servlet e JSP

Una Servlet è un programma scritto in Java e residente su un server, in grado di gestire le richieste generate da uno o più client, attraverso uno

scambio di messaggi tra il server ed i client stessi che hanno effettuato la richiesta. Tipicamente sono collocate all'interno di Application Server o Web Application Server come, ad esempio, Tomcat.

Visto che, come detto, le Servlet sono scritte in Java esse possono avvalersi interamente delle Java API che, come è noto, consentono di implementare con relativa semplicità anche svariate funzionalità complesse e importanti come, ad esempio, l'accesso ad un database. Ad esse si aggiungono, poi, le più specifiche servlet API, che mettono a disposizione un'interfaccia standard utilizzata per gestire la comunicazione tra un client Web ed una Servlet. Vediamo di illustrare, adesso, come lavora una HttpServlet avvalendoci dello schema seguente:

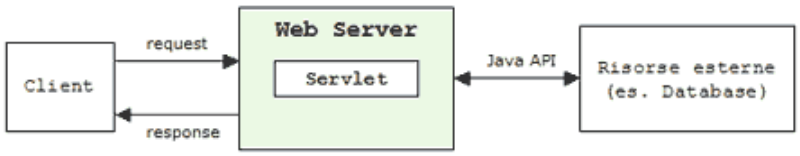

Figura 4.2: Schema di una servlet

È possibile riassumere il flusso rappresentato nella figura in questo modo:

- Un client invia una richiesta (request) per una servlet ad un web application server.
- Qualora si tratti della prima richiesta, il server istanzia e carica la servlet in questione avviando un thread che gestisca la comunicazione con la servlet stessa. Nel caso, invece, in cui la Servlet sia già stata caricata in precedenza (il che, normalmente, presuppone che un altro client abbia effettuato una richiesta antecedente quella attuale) allora verrà, più semplicemente, creato un ulteriore thread che sarà associato al nuovo client, senza la necessità di ricaricare ancora la Servlet.
- Il server invia alla servlet la richiesta pervenutagli dal client
- La servlet costruisce ed imposta la risposta (response) e la inoltra al server

- Il server invia la risposta al client.

HttpServletRequest e HttpServletResponse

Abbiamo visto, esaminando il flusso di esecuzione di una servlet, che il flusso stesso è incentrato su due componenti fondamentali: la richiesta (request, inviata dal client verso il server) e la risposta (response, inviata dal server verso il client). In Java, questi due componenti sono identificati, rispettivamente, dalle seguenti interfacce:

- *javax.servlet.http.HttpServletRequest*
- *javax.servlet.http.HttpServletResponse*

Un oggetto di tipo HttpServletRequest consente ad una Servlet di ricavare svariate informazioni sul sistema e sull'ambiente relativo al client. L'oggetto di tipo HttpServletResponse, invece, costituisce la risposta da inviare al client e che, come si è detto, può essere dipendente da svariati data source.

Metodi principali

Creare una Servlet vuol dire, in termini pratici, definire una classe che derivi dalla classe HttpServlet. I metodi più comuni per molti dei quali si è soliti eseguire l'overriding nella classe derivata sono i seguenti:

- void doGet(HttpServletRequest req, HttpServletResponse resp) Gestisce le richieste HTTP di tipo GET. Viene invocato da service()
- void doPost(HttpServletRequest req, HttpServletResponse resp) Gestisce le richieste HTTP di tipo POST. Viene invocato da service()
- void service(HttpServletRequest req, HttpServletResponse resp) Viene invocato al termine del metodo
- void doPut(HttpServletRequest req, HttpServletResponse resp) Viene invocato attraverso il metodo service() per consentire di gestire una richiesta HTTP di tipo PUT. Tipicamente, una richiesta del genere

consente ad un client di inserire un file su un server, similmente ad un trasferimento di tipo FTP.

- void doDelete(HttpServletRequest req, HttpServletResponse resp) Viene invocato attraverso il metodo service() per consentire ad una Servlet di gestire una richiesta di tipo HTTP DELETE. Questo genere di richiesta viene utilizzata per rimuovere un file dal server.

- void init() Viene invocato soltanto una volta dal Servlet Engine al termine del caricamento della servlet ed appena prima che la servlet stessa inizi ad esaudire le richieste che le pervengono.

- void destroy() È invocato direttamente dal Servlet Engine per scaricare una servlet dalla memoria.

- String getServletInfo() È utilizzato per ricavare una stringa contenente informazioni di utilità sulla Servlet (ad es.: nome della Servlet, autore, copyright). La versione di default restituisce una stringa vuota.

JSP

JavaServer Pages, di solito indicato con l'acronimo JSP (letto anche talvolta come Java Scripting Preprocessor) è una tecnologia di programmazione Web in Java per lo sviluppo di applicazioni Web che forniscono contenuti dinamici in formato HTML o XML. Si basa su un insieme di speciali tag con cui possono essere invocate funzioni predefinite o codice Java. Nel contesto della piattaforma Java, la tecnologia JSP è correlata con quella delle servlet: all'atto della prima invocazione, le pagine JSP vengono infatti tradotte automaticamente da un compilatore JSP in servlet. Una pagina JSP può quindi essere vista come una rappresentazione ad alto livello di un servlet. Per via di questa dipendenza concettuale, anche l'uso della tecnologia JSP richiede la presenza, sul Web server, di un servlet container, oltre che di un server specifico JSP detto motore JSP (che include il compilatore JSP); in genere, servlet container e motore JSP sono integrati in un unico prodotto (per esempio, Tomcat svolge entrambe le funzioni).

4.6.4 Realizzazione

L'entità da rappresentare e da manipolare in questa applicazione è il dispositivo domotico. A partire da questa riflessione, dovremmo avere un JavaBean che rappresenti il dispositivo domotico interrogato. Per quanto riguarda le servlet dovremo avere una servlet che vada in esecuzione appena viene richiesta la pagina del dispositivo che, leggendo l'indirizzo a cui viene fatta la richiesta setti il JavaBean del dispositivo in maniera adeguata con i dati ricevuti da domoNet, attraverso i webservices. La pagina JSP prenderà un JavaBean e stamperà a video le sue caratteristiche: nel nostro caso il nome del dispositivo e la lista delle funzioni invocabili per visualizzare e modificare lo stato. Una volta selezionata una funzione, entrerà in gioco un'altra servlet che provvederà alla manipolazione dello stato del dispositivo interagendo con domoNet tramite i webservices. La creazione dell'interfaccia grafica viene demandata completamente alle pagine JSP, secondo il modello MVC.

L'importanza di questa suddivisione sta nel fatto che, cambiando il modo di visualizzare l'output della pagina JSP, si può portare il sistema a rispondere alle più svariate necessità grafiche e di linguaggio. In questo caso la *View* è stata realizzata in HTML e con una grafica molto scarna ma nulla vieta di scegliere un altro linguaggio e nulla vieta di cambiare l'aspetto grafico di queste pagine.

In parole povere: cambia la pagina JSP, cambia il modo di visualizzare le informazioni ma restano valide e immutate le servlet che stanno dietro ad esse.

Pagina principale del dispositivo

La pagina principale del dispositivo è raggiungibile tramite l'indirizzo *"dispositivo.domonet.prova:8080"*. Viene realizzata grazie alla stretta collaborazione tra servlet e jsp, per mezzo dei JavaBean:

- La servlet principale si occupa di riempire i domoBean, collegandosi ai WebServices di domoNet grazie ad un oggetto della classe ManageWS.

- Passa il domoBean ad una pagina JSP che stampa a video il nome e le funzioni invocabili da quel dispositivo.

Si realizza così il meccanismo MVC spiegato precedentemente. La Servlet inoltre è completamente configurabile tramite il file di deployment delle servlet (web.xml) per quanto riguarda i suoi parametri di funzionamento:

- Indirizzo del server Axis di domoNet
- Nome della zona DNS

Figura 4.3: Pagina principale

Codice 4.13: DomoWEBClient: Deployment Servlet e parametrizzazione

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
    2.3//EN"
   "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>

    <display-name>
        DomoWebClient
    </display-name>
```

```
    <description>
        Un semplice client web per testare i dispositivi domotici mappati con indirizzi IPv6
    </description>

    <servlet>
        <servlet-name>testServlet</servlet-name>
        <description>servlet principale</description>
        <servlet-class>myservlet.domoWebClient</servlet-class>
        <init-param>
            <param-name>indServ</param-name>
            <param-value>http://localhost:8080/axis/services/DomoNetWS</param-value>
        </init-param>
        <init-param>
            <param-name>nomeZona</param-name>
            <param-value>domonet.prova</param-value>
        </init-param>
    </servlet>

    <servlet>
        <servlet-name>execute</servlet-name>
        <description>servlet che esegue i servizi del dispositivo</description>
        <servlet-class>myservlet.Execute</servlet-class>
    </servlet>

    <servlet-mapping>
            <servlet-name>testServlet</servlet-name>
            <url-pattern>/testServlet</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
            <servlet-name>execute</servlet-name>
            <url-pattern>/execute</url-pattern>
    </servlet-mapping>

    <session-config>
        <session-timeout>60</session-timeout>
    </session-config>

</web-app>
```

Codice 4.14: DomoWEBClient: Servlet principale

```
package myservlet;
import com.domoBeans.*;
import domoutils.ipAssigner;
import common.*;

import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.net.Inet6Address;
import java.net.InetAddress;

//servlet principale
public class domoWebClient extends HttpServlet
{
    String indServ;
    String nomeZ;

    public void init()
        throws ServletException
    {
        //legge i parametri di configurazione dal file web.xml
        indServ = getServletConfig().getInitParameter("indServ");
        nomeZ = getServletConfig().getInitParameter("nomeZona");
    }

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        try{
            //si collega ai webservices di domoNet
            ManageWS manager = new ManageWS(indServ);
            manager.connectToWebServices();
            //imposta il manager come variabile di sessione
            req.getSession().setAttribute("managerws", manager);

            //ricavo il nome del dispositivo dall'URL tramite una chiamata DNS
            Inet6Address addr = (Inet6Address) InetAddress.getByName(req.getLocalAddr());
            String myhostName = addr.getHostName();
            //isolo il nome "filtrato" del dispositivo
            String mydeviceName = myhostName.substring(0,myhostName.indexOf("."+nomeZ));

            //vado a cercare il dispositivo corrispondente tra quelli connessi
            //a quello digitato nell'URL
            Devices disp = manager.getDisp();
            Device x = new Device();
            Vector disps = disp.getDevices();
            for (int i = 0; i < disps.size(); i++)
```

```
                {
                    Device temp = (Device) disps.elementAt(i);
                    String filteredName = ipAssigner.filterName(temp.getNome());
                    if (filteredName.equals(mydeviceName))
                    {
                        x = temp;
                        break;
                    }
                }

                //imposto il bean Device con scope di sessione
                req.getSession().setAttribute("dispositivo", x);
                //richiamo la pagina jsp per visualizzare i dati
                forward(req, res, "/mostra.jsp");
            }catch(Exception e)
            {
                Debug.writeln(e.toString());
            }
        }

        private void forward(HttpServletRequest request, HttpServletResponse response, String page)
            throws ServletException, IOException
        {
            ServletContext sc = getServletContext();
            RequestDispatcher rd = sc.getRequestDispatcher(page);
            rd.forward(request,response);
        }

}
```

Codice 4.15: DomoWEBClient: Pagina JSP mostra.jsp

```
<%@ page import="java.util.*" %>
<%@ page import="java.io.*" %>
<%@ page import="com.domoBeans.*" %>
<jsp:useBean id="dispositivo" class="com.domoBeans.Device" scope="session" />
<html>
<head>
<title>Dispositivo </title>
</head>
<body>
<div>
<h1><jsp:getProperty name="dispositivo" property="nome" /></h1>
<%
    Vector servizi = dispositivo.getServizi();
    for(int i = 0; i < servizi.size(); i++)
    {
        Service temp = (Service) servizi.elementAt(i);
        out.println("<div><a href='execute?sid="+i+"'>"+ temp.getNome() +"</a></div>");
    }
%>
</div>
</body>
</html>
```

Pagina di esecuzione del servizio

La pagina di esecuzione del servizio viene creata in base alle caratteristiche del servizio scelto.

- Se il servizio ha un output da mostrare, mostra il risultato
- Se il servizio richiede l'inserimento di alcuni dati, crea una form in base agli input necessari al suo funzionamento

Anche per questa pagina vi è una divisione netta dei compiti tra ServLet e JSP. La Servlet va ad occuparsi dell'interazione con i servizi di domoNet poi, impostando alcuni attributi e alcuni Bean, permette alla JSP di generare la pagina HTML in maniera adeguata a seconda del servizio scelto.

Figura 4.4: Pagina di esecuzione del servizio: solo output

Figura 4.5: Pagina di esecuzione del servizio: inserimento input

Codice 4.16: DomoWEBClient: Servlet execute

```
package myservlet;
import com.domoBeans.*;
import common.*;
import domoML.domoMessage.*;

import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.lang.Integer;

//servlet che si occupa della gestione dei servizi del dispositivo scelto
public class Execute extends HttpServlet
{
    //Risposta della servlet a richieste di tipo GET
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        try{
            //ricavo il Device Bean dalla sessione
            Device mydevice = (Device) req.getSession().getAttribute("dispositivo");

            //se non e' stato settato, c'e' un problema
            if (mydevice == null)
            {
                throw new Exception();
            }

            //prendo il parametro GET sid, indicante il servizio selezionato
            int id = Integer.parseInt(req.getParameter("sid"));
            //ricavo il servizio selezionato del dispositivo scelto
            Service selected_service = (Service) mydevice.getServizi().elementAt(id);

            //imposto due attributi:
            //uno setta il JavaBean relativo al servizio
            //l'altro imposta di default, la visualizzazione del link di ritorno al dispositivo a 0
            req.setAttribute("servizio", selected_service);
            req.setAttribute("link", 0);

            //se il servizio scelto ha subito un output, lo visualizzo
            if (selected_service.hasOutput())
            {
                //esegue subito il message e restituisce il risultato
                //inutile aspettare, non serve l'intervento dell'utente
                ManageWS manager = (ManageWS) req.getSession().getAttribute("managerws");
                mydevice.setUrl(manager.getURL());
                String risultato = manager.executeService(mydevice, selected_service, null, null);

                //ovviamente non lo visualizzo dalla servlet (modello MVC)
                //passo alla pagina JSP un attributo con il risultato
```

```
                req.setAttribute("hasOutput", risultato);
                req.setAttribute("link", 1);

            }
            //se il servizio richiede l'acquisizione di dati devo creare un'interfaccia per l'utente
            if (selected_service.hasInput())
            {
                //non posso farlo nella servlet (modello MVC)
                //demando questa funzionalita' alla JSP con l'attributo
                //hasInput settato a 1
                req.setAttribute("hasInput", 1);
            }

            //carico la pagina JSP che visualizzera' l'interfaccia grafica corretta
            //rispetto al tipo di servizio scelto
            forward(req, res, "/mostra_servizio.jsp");
        }catch(Exception e)
        {
            Debug.writeln(e.toString());
        }
    }

    //Risposta della servlet a richieste di tipo POST
    //(in questo caso la sottomissione della form di input del servizio)
    public void doPost(HttpServletRequest req,HttpServletResponse res)
        throws ServletException,IOException
    {
        //se i dati sono stati inseriti dall'utente eseguo il codice
        if ( req.getParameter("doit") != null)
        {
            try {
                //attraverso il Device bean di sessione mi ricavo tutti gli elementi necessari
                //all'invocazione del metodo executeService() per costruire un domoMessage
                // ed eseguirlo
                int sid = Integer.parseInt( req.getParameter("sid") );
                Device mydevice = (Device) req.getSession().getAttribute("dispositivo");
                Service selected_service = (Service) mydevice.getServizi().elementAt(sid);
                ManageWS manager = (ManageWS) req.getSession().getAttribute("managerws");
                mydevice.setUrl(manager.getURL());
                Vector input_set = selected_service.getInput();
                Vector<String> resultset = new Vector<String>();
                for (int i = 0; i < input_set.size(); i++)
                {
                    Input temp = (Input) input_set.elementAt(i);
                    resultset.addElement( req.getParameter(temp.getNome()) );
                }

                String risultato = manager.executeService(mydevice, selected_service, input_set,
                    resultset);
```

```
                    //attraverso l'attributo hasOutput e l'attributo postExec
                    //informo la pagina JSP che visualizza i risultati
                    //che ha un risultato da visualizzare proveniente da una chiamata
                    //al metodo executeService (quindi deve visualizzarne l'esito)
                    req.setAttribute("link", 1);
                    req.setAttribute("servizio", selected_service);
                    req.setAttribute("hasOutput", risultato);
                    req.setAttribute("postExec", 1);

                    //carico la pagina JSP
                    forward(req, res, "/mostra_servizio.jsp");
                } catch (Exception e)
                {
                    Debug.writeln(e.toString());
                }
            }
        }


        private void forward(HttpServletRequest request, HttpServletResponse response, String page)
            throws ServletException, IOException
        {
            ServletContext sc = getServletContext();
            RequestDispatcher rd = sc.getRequestDispatcher(page);
            rd.forward(request,response);
        }

}
```

Codice 4.17: DomoWEBClient: Pagina JSP mostra_servizio.jsp

```
<%@ page import="java.util.*" %>
<%@ page import="java.io.*" %>
<%@ page import="java.lang.*" %>
<%@ page import="com.domoBeans.*" %>
<jsp:useBean id="servizio" class="com.domoBeans.Service" scope="request" />
<html>
<head>
<title>Servizio</title>
</head>
<body>
<div>
<h1><jsp:getProperty name="servizio" property="nome" /></h1>
<%
    String grafica = "";
     int id = Integer.parseInt(request.getParameter("sid"));
    if(request.getAttribute("hasOutput") != null)
    {
        String risultato = (String) request.getAttribute("hasOutput");
        grafica += "<div><span><b>Risultato: </b></span><span>";
        if (risultato.equals("FAILURE"))
        {
            grafica += "ERRORE</span></div>";
        }
        else
        {
            if (request.getAttribute("postExec") != null)
                risultato = "Eseguito con successo..." + risultato;
            grafica += risultato + "</span></div>";
        }
    }
    else if(request.getAttribute("hasInput") != null)
    {
        grafica += "<form method=\"POST\">\n<input type=\"hidden\" value=\""+id+"\"
            name=\"sid\">\n";
        if (servizio.hasInput())
        {
            Vector input_set = servizio.getInput();
            for (int i = 0; i < input_set.size(); i++)
            {
                Input temp = (Input) input_set.elementAt(i);
                grafica += "<div>\n<label>"+temp.getNome()+"</label>\n";
                if (temp.hasAllowed())
                {
                    grafica += "<select name=\""+temp.getNome()+"\">\n";
                    Vector opzioni = temp.getValori();
                    for (int j = 0; j < opzioni.size(); j++)
                    {
                        Allowed opt = (Allowed) opzioni.elementAt(j);
```

```
                                grafica += "<option
                                    value=\""+opt.getValore()+"\">"+opt.getValore()+"</option>\n";
                            }
                            grafica += "</select>\n</div>\n";
                        }
                        else
                        {
                            grafica += "<div>\n<input type=\"text\"
                                name=\""+temp.getNome()+"\">\n<div>\n";
                        }
                    }
                    grafica +="<div>\n<input type=\"submit\" value=\"Esegui\"
                        name=\"doit\">\n<div>\n";
                }

                grafica += "</form>\n";
            }
            out.println(grafica);

            int x = (Integer) request.getAttribute("link");
            if (x != 0)
            {
                out.println("<a href='/'>Torna al dispositivo </a>");
            }
        %>
        </div>
        </body>
        </html>
```

Capitolo 5

Conclusioni

Questo lavoro vuole essere un primo piccolo passo per estendere le funzionalità delle reti domotiche verso l'universo dell'Internet delle Cose. I dispositivi ora, seppur virtualmente, hanno un loro indirizzo IPv6 al quale possono rispondere a varie richieste. Per quanto riguarda questo progetto le richieste fatte ai dispositivi vengono inviate da una sorgente umana: un utente si collega all'applicazione web e seleziona la funzione del dispositivo scelto. In futuro saranno i dispositivi stessi a comunicare tra loro, tramite gli indirizzi IPv6 e i webservices, per realizzare gli scopi della domotica visti nel Capitolo 1.

Bibliografia

- Per la parte sulla domotica
  - Leone S., Russo D., Miori V., *La casa apprende automaticamente le abitudini dell'utente*
  - Bianchi Bandinelli R., Miori V., *Appunti di domotica*
  - Miori V., Russo D., Concordia C. *Meeting People's Needs in a Fully Interoperable Domotic Environment.* In: Sensors, vol. 12 (6) pp. 6802 - 6824. Select papers from UCAmI 2011 - the 5th International Symposium on Ubiquitous Computing and Ambient Intelligence (UCAmI'11). MDPI Publishing, Basel, Switzerland, 2012.
  - Miori V., Russo D., Aliberti M. *Domotic technologies incompatibility becomes user transparent.* In: Communications of the Acm, vol. 53 (1) pp. 153 - 157. ACM, 2010.
- Per la parte su IPv6
  - Hinden R., Deering S., Nokia, *RFC 2373: IP Version 6 Addressing Architecture*, 1998
  - Hinden R., Haberman B., Nokia, *RFC 4193: Unique Local IPv6 Unicast Addresses*, 2005
  - Narten T., IBM, Nordmark E., Sun Microsystem, Simpson W., Daydreamer, Soliman H., Elevate Technologies, *RFC 4861: Neighbor Discovery for IP version 6*, 2007

  - http://packetlife.net/blog/2008/aug/4/eui-64-ipv6/, *EUI-64 in IPv6*, 2008 (immagini)
  - Tanenbaum A.S, *Reti di calcolatori*, 2003

- Per la parte su domoNet
  - Russo D., Bianchi Bandinelli R., Miori V., *La domotica e Internet. Una soluzione per l'interoperabilità*, 2006
  - Miori V., Russo D. *DomoNet architecture.* [Sofware] Release 0.2 , 21 November 2009.
  - Miori V., Tarrini L., Manca M., Tolomei G. *An open standard solution for domotic interoperability.* In: IEEE Transactions on Consumer Electronics, vol. 52 (1) pp. 97 - 103. IEEE, 2006.

- Per la parte sulle Servlet e JSP
  - Goodwill J., *Developing Java Servlets*, 2001
  - Brittain J., Darwin I.F., *Tomcat: The Definitive Guide*, 2008