CONSIGLIO NAZIONALE DI RICERCA
Istituto "A. Faedo"
Laboratorio di domotica

Una piattaforma software universale per i sistemi domotici

Dott. Vittorio Miori
Dott. Dario Russo

Indice

Capitolo 1

Introduzione

L'obiettivo principale proposto da raggiungere è quello di studiare un'architettura che permetta di risolvere il problema della cooperazione tra dispositivi domotici eterogenei dal punto di vista tecnologico e di implementare un'applicazione basata su di essa. L'architettura usata deve inoltre permettere il controllo dei dispositivi domotici in remoto, indipendentemente dalla loro locazione.

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 di avere una visione omogenea indipendentemente dalle tecnologie sottostanti.

L'idea è quella di avere un gruppo di *web service* ognuno dei quali capace di interagire con i dispositivi domotici da lui fisicamente raggiungibili, tipicamente in un'area geografica ben precisa circoscritta intorno sua locazione fisica. Ogni area geografica, rappresentata logicamente dal *web service*, racchiude come servizi l'insieme delle funzionalità offerte dai dispositivi presenti. Ogni *web service* deve essere in grado di interpretare e comportarsi conseguentemente allo stato del livello astratto usando dei moduli capaci di interfacciarsi con i *middleware* dei dispositivi.

1.1 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 "sottorete", pur senza soffermarsi sui dettagli e sulle infrastrutture necessarie. La Figura 1.1 mostra lo scenario appena descritto:

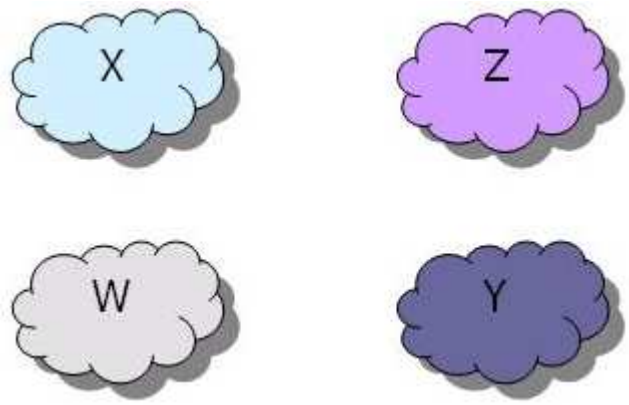

Figura 1.1: Rappresentazione dei middleware domotici.

Ogni "nuvola" rappresenta un *middleware*, ovvero una sottorete all'interno della quale sono presenti sia le infrastrutture *hardware* e *software* per il supporto, sia i dispositivi conformi.

Tutti i dispositivi all'interno della stessa sottorete 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 sottoreti. Per collegamento si intende un'infrastruttura logica che consenta ai dispositivi di qualsiasi sottorete di conoscere tutti i dispositivi presenti nelle altre sottoreti.

Infine, anche qualora fosse possibile avere una visione comple-

ta 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 meccanismo di comunicazione universale veicolato da quest'infrastruttura.

Una possibile soluzione a questi due problemi è quella di introdurre tra ogni coppia di sottoreti 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 sottoreti 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 sottoreti, il numero di gateway da sviluppare cresce sensibilmente.

A questa prima soluzione ne abbiamo preferita una seconda, che interviene ad un livello superiore.

Per riconoscere la validità di questo approccio e la sua applicabilità, siamo partiti da un'analogia che deriva direttamente dal mondo delle reti di calcolatori.

In passato, molti costruttori hanno sviluppato infrastrutture ad *hoc* per la costituzione di reti che permettessero la comunicazione tra piattaforme e periferiche dello stesso marchio; si pensi ad

esempio alla rete *AppleTalk* dedicata ai sistemi *Macintosh*, a quella *NFS* per i sistemi *Unix*, a *NetBeui* per i PC *Windows* oppure alle reti geografiche *IBM SNA*, *NOVELL/IPX*, etc. Queste soluzioni limitavano notevolmente la potenziale espansione delle reti poiché non permettevano la comunicazione tra macchine che supportavano infrastrutture di rete diverse.

Proprio per questo motivo, vi sono stati tentativi da parte di singole aziende, peraltro falliti, di imporre la propria infrastruttura come unico standard, come nel caso di *Microsoft* al rilascio del sistema operativo *Windows 95*.

Di fatto nessuna infrastruttura ha mai raggiunto un'egemonia sulle altre, principalmente a motivo dello sviluppo di un'infrastruttura innovativa in grado di superare il vincolo della unicità di piattaforma: questa infrastruttura, di ispirazione *ISO/OSI*, è il noto *stack* di protocolli *TCP/IP*. Seguendo lo stesso approccio in contesto domotico, si è pensato di introdurre un'ulteriore nuvola atta a gestire i diversi sistemi domotici al di là delle differenze fra essi, permettendo così la comunicazione tra nodi appartenenti a sottoreti distinte.

Questa nuova infrastruttura, denominata *domoNet*, si basa sui *web service* e fornisce una visione dell'intera topologia della rete secondo un modello *SOA*.

Per garantire l'interoperabilità tra nodi comunicanti attraverso l'infrastruttura *domoNet*, è necessario sviluppare, oltre ad opportuni *gateway*, una grammatica *XML* standard detta *domoML*.

Rispetto alla prima soluzione proposta, questa è certamente più scalabile, in quanto richiede solamente un *gateway* per ogni sistema domotico da collegare a *domoNet*.

Ogni *gateway*, che d'ora in avanti chiameremo *tech manager*, dovrà possedere da un lato un'interfaccia verso il particolare sistema domotico a cui si riferisce, e dall'altro un'interfaccia verso

domoNet **(figura 1.2).**

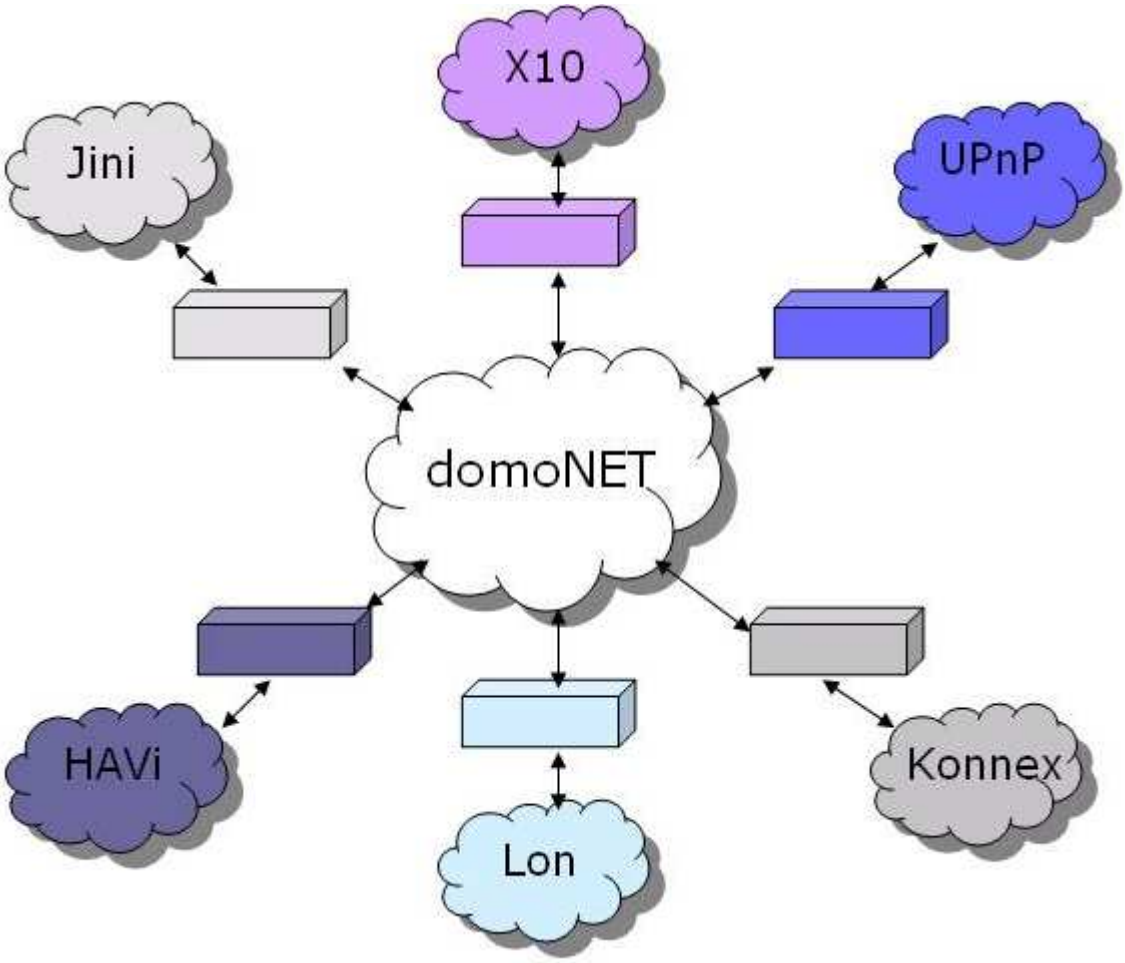

Figura 1.2: Il framework domoNet.

1.2 Un primo sguardo all'architettura domoNet

L'architettura *domoNet* è composta da un insieme di server (*web service*) e da dei possibili *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*.

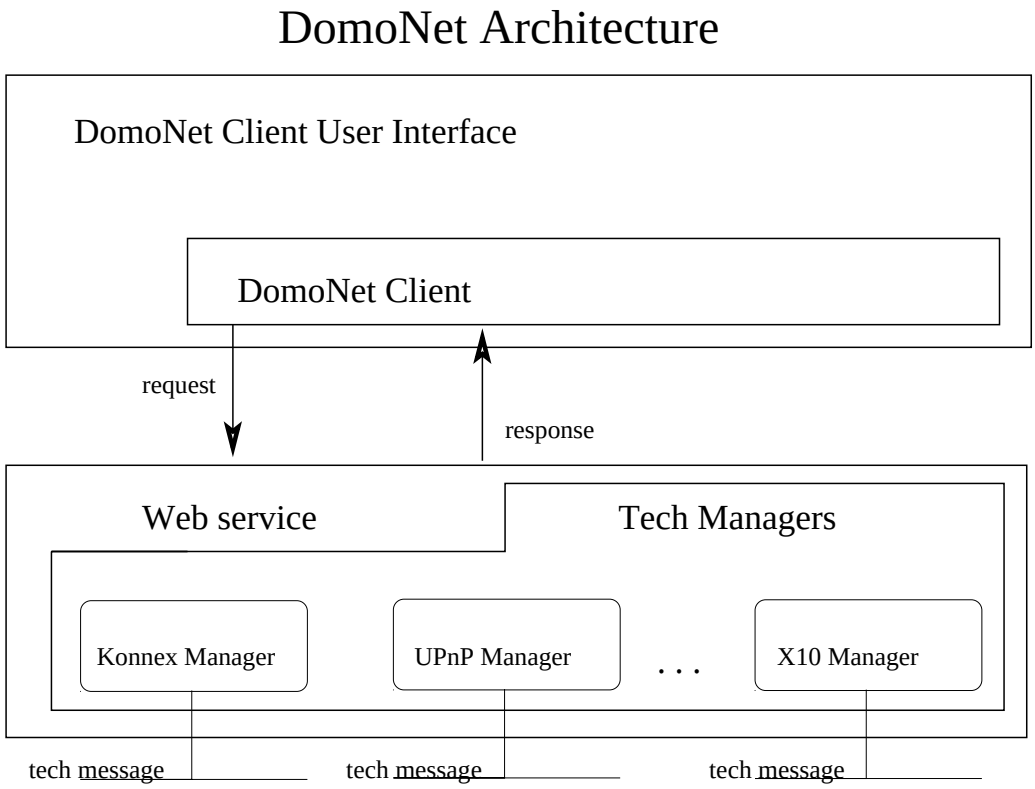

Figura 1.3: L'architettura domoNet

In figura 1.3 viene dato un primo sguardo all'architettura *domoNet*. L'architettura mostrata è composta da una parte *client* (il *domoNetClient + domoNetClientUI*) che, connessa alla parte *server* (*web service + tech mangers*) è in grado di effettuare richieste tramite *TCP/IP* affinché 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.

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 in grado di avere 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*.

1.3 Terminologia

- ***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* può 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*.**

Capitolo 2

DomoML

DomoML **è un un linguaggio per rappresentare in maniera compatta i dispositivi domotici: i relativi servizi offerti e le relative interazioni, astraendo dalla tecnologia di appartenenza.**

DomoML**, usato con questa architettura, è un *middle language* da e verso quale tradurre le rappresentazioni dei dispositivi e dei pacchetti. Tutta la parte logica dell'architettura usa *domoML* e solamente le interazioni fisiche con i dispositivi per costruire i corrispondenti *domoDevice* e per l'esecuzione dei servizi, all'interno dei moduli (*tech manager*), usano il linguaggio specifico proprio di ogni tecnologia.**

DomoML **si occupa attraverso i *tag*:**

- ***domoDevice*****: alla descrizione astratta dei dispositivi fatte secondo *domoML*;**
- ***domoMessage*****: alla descrizione astratta delle interazioni da e verso i dispositivi fatte secondo *domoML*.**

2.1 *DomoDevice*

Il *domoDevice* ha lo scopo di creare un meccanismo semplice, compatto ed efficace per rappresentare i dispositivi astraendo dalla

tecnologie.

Un *domoDevice* può essere rappresentato attraverso un albero largo e poco profondo (la grammatica è altamente attributata e con pochi figli). L'albero ha al massimo 4 livelli:

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`:
     usato per prendere e generare il `domoDeviceId`. Identifica il *web service* che tiene di dispositivo.

   Tutti i campi sono opzionali eccetto per l'`id`, `URL` e `tech` perché permettono di identificare e usare il 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 non va messo se non è previsto alcun valore di ritorno;
   - `outputDescription`:
     una descrizione in linguaggio naturare per l'attributo `output`. Se non è presente l'attributo `output` neanche questo attributo deve essere messo;
   - `prettyName`:
     una etichetta in 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 può 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` **dell'input atteso;**

- `linkedService`:
  il servizio da associare quando questo servizio viene eseguito. Può essere invocato un servizio dello stesso o di un qualsiasi altro *domoDevice*. L'associazione di serivizi non ha vincoli semantici ovvero è possibile combinare un servizio con qualsiasi altro servizio a patto di riuscire a dare dei valori ragionevoli ai `linkedInput` **(discusso successivamente). Questo *tag* è il cuore della tesi in quanto permette di risolvere il problema della cooperazione tra dispositivi reali eterogenei. Questo tag è opzionale. 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** `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 preso;**

Un esempio di *domoDevice* per una semplice lampadina è:

```
<device description="lampada a risparimo energetico"
 id="0" manufacturer="pholips"
 postionDescription="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>
```

In questo esempio è possibile vedere che il *domoDevice* descrit-

to è una lampadina a risparmio energetico prodotta dalla ditta `pholips` **e che si trova sul comodino accanto al letto. Usa la tecnologia *Konnex* e fa parte della categoria** `lampada`**. La lampada è 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***
`url="http://www.altrowebservice.it/servizio"`**, con** `id=3` **e con valore di input** `TRUE`**.**

2.2 *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` può 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`:
   i 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;**

Un esempio di *domoMessage* per accendere una semplice lampadina è:

```
<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:

```
<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 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.

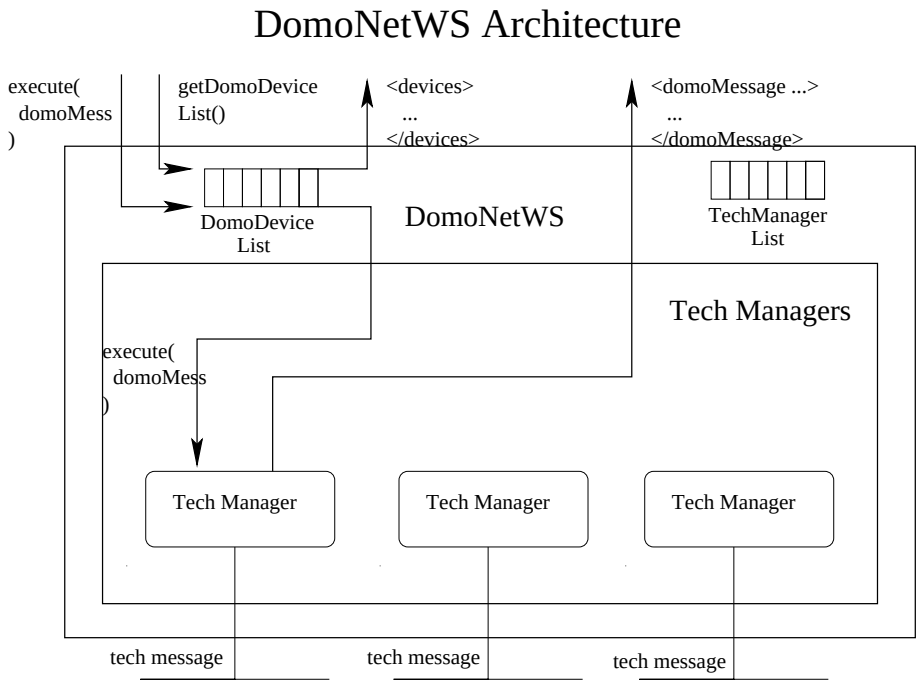

Figura 3.1: L'architettura del domoNetWS (web service).

Il *web service*, al momento in cui viene montato, inizializza i *tech*

manager **elencati nella** `techManagerList` **dai quali riceve la lista dei** ***device*** **disponibili. In un** ***web service*** **ci possono essere più** ***tech manager*** **differenti.**

Ogni dispositivo riconosciuto da ogni ***tech manager*** **viene convertito usando il formalismo** ***domoDevice*** **e memorizzato nella** `DomoDeviceList` **dove viene indicizzato assegnandogli un** ***domoAddress*** **composto dalla** ***url*** **del** ***web service*** **di appartenenza, e da un** ***id*** **(un numero progressivo intero che permette di identificarlo all'interno).**

3.1 Eseguire un *domoMessage*

La richiesta di eseguire un ***domoMessage*** **può 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 essere eseguito, 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`**, individuato il** ***tech mamanger*** **di competenza al quale forwardare 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.**

3.2 La cooperazione tra differenti *tech manager*

A questo livello, la cooperazione tra dispositivi appartenenti a diverse tecnologie funziona come mostrato (figura 3.2):

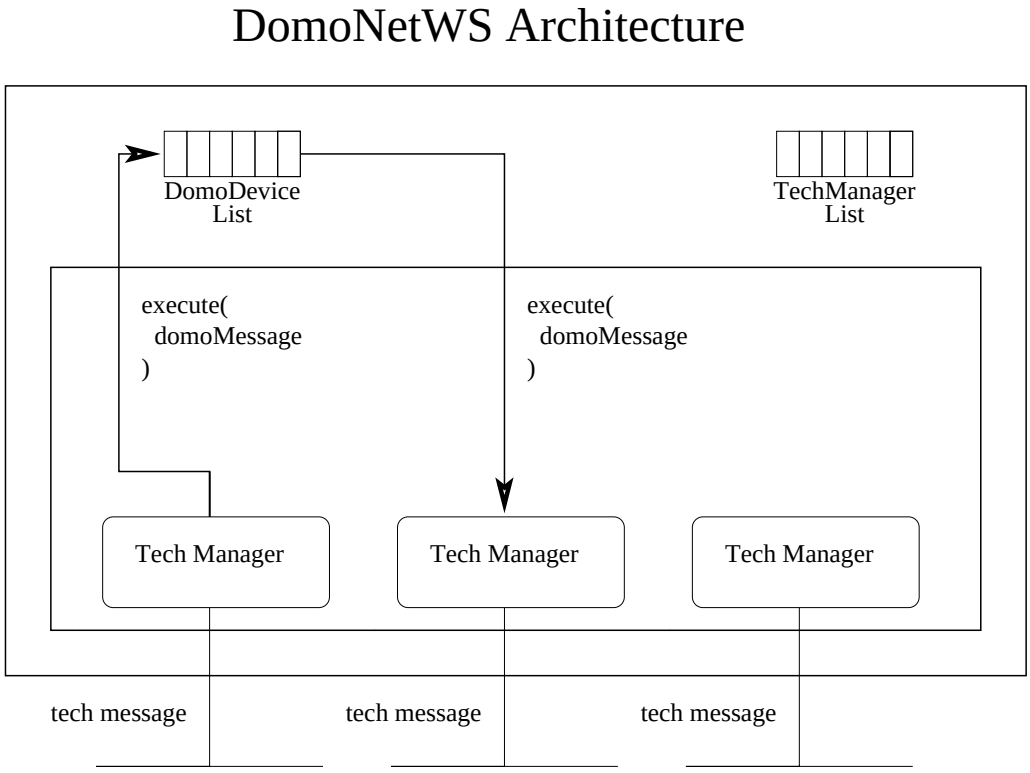

Figura 3.2: L'architettura del domoNetWS (web service): la cooperazione.

Il *tech manager* riceve un *tech message* dalla propria rete ed individua, analizzandolo, il *domoDevice* rappresentante il mittente ed il servizio coinvolto. I dati trovati vengono analizzati dal *web service* che provvede a verificare l'esistenza di `linkedService` 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 dispo-

sitivi appartenenti ad altri *web service* sfruttando la rete Internet e comunicando direttamente con il *domoNetWS* interessato. All'interno della `domoDeviceList` 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 (figura 3.3).

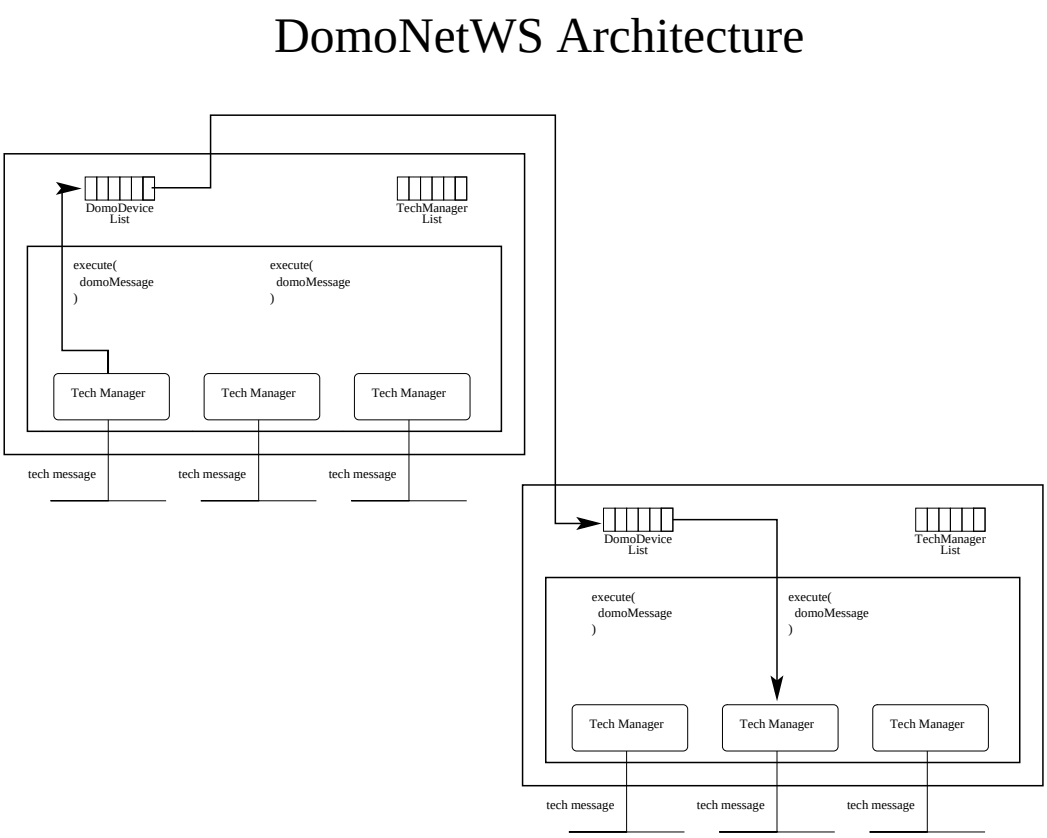

Figura 3.3: L'architettura del domoNetWS (web service): la cooperazione tra diversi web service*.*

3.3 Il *tech manager*

Il *tech manager* è un modulo (o driver) del *web service* che ha la funzione di interfacciarsi fisicamente con i dispositivi appartenenti ad una determinata tecnologia come *Konnex*, *UPnP* etc.

Il *tech manager* implementa le chiamate, funzioni e tutto quanto necessario alla corretta interazione con i dispositivi.

Le funzioni principali a cui deve essere in grado di adempiere sono:

- **la generazione di una lista di *domoDevice* disponibili;**
- **tradurre un *domoAddress* in un *real address* e vice versa;**
- **tradurre un *domoMessage* in un *tech message* e vice versa.**

In fase di inizializzazione, il *tech manager* deve avere la lista dei dispositivi disponibili ed inizializzare eventuali strutture dati.

3.3.1 Esecuzione di un *domoMessage*

Quando il *techManager* riceve un *domoMessage*, esso ricava da quest'ultimo il *real address* del dispositivo destinatario del messaggio e, usando i dati del messaggio, genera il corrispettivo *tech message*.

Questo significa che il *tech manager* deve verificare se il *domoMessage* è di tipo `COMMAND`, e capire l'azione da eseguire interpretando l'attributo `message`. Deve convertire i tag `input` (se presenti) usando gli attributi `datatype` e `value` in modo da poter tradurre correttamente i valori da propagare. A questo punto il messaggio (ora *tech message*) può essere inviato usando il mezzo di comunicazione previsto per la specifica tecnologia in uso.

L'esecuzione di un *tech message* può (e non deve) coinvolgere altri messaggi come ad esempio una risposta al messaggio con un valore o con un *ack*. Se è prevista una risposta, deve essere convertita da *tech message* in *domoMessage* considerando ancora i *datatype* ed i valori. Queste informazioni vengono memorizzate all'interno del *domoMessage* di tipo `SUCCESS` o `FAILURE`. Se non è prevista una risposta, il *tech manager* restituisce un *domoMessage*

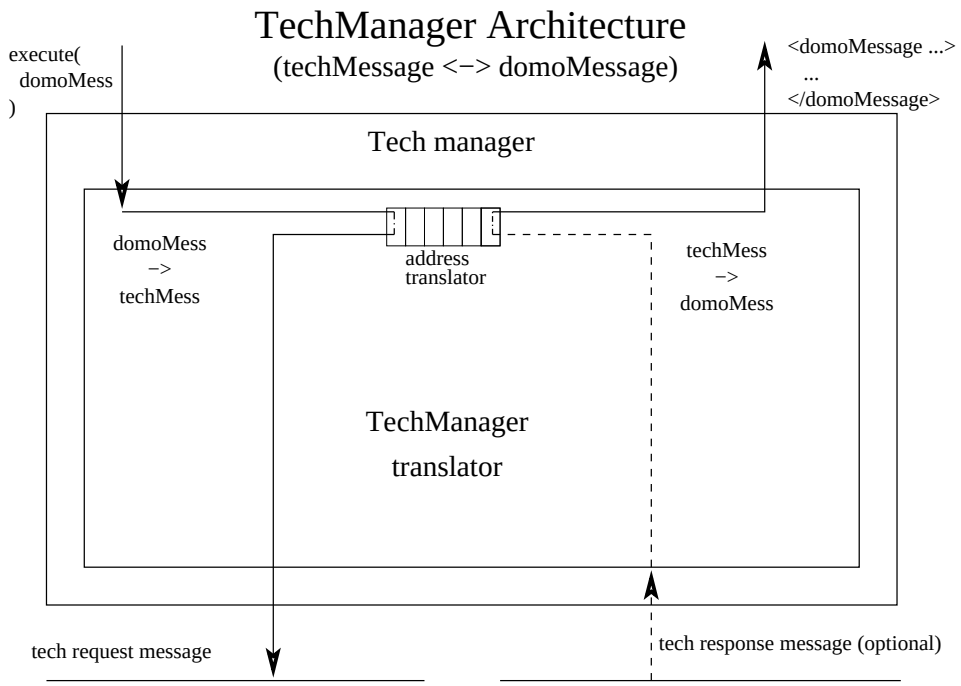

Figura 3.4: L'architettura del techManager: l'esecuzione di un domoMessage.

vuoto comunque sempre di tipo `SUCCESS` o `FAILURE` in dipendenza dallo stato della rete, dal rilevamento di eccezioni, etc.

Per fare tutto ciò, il *tech manager* deve avere al suo interno un traduttore da *tech message* a *domoMessage* e vice versa.

Per tradurre gli indirizzi, il *tech manager* usa una `DoubleHash`, una *hashMap* speciale che permette di usare il *domoDeviceId* o il *real address* come chiave per ottenere l'altro. Per ogni *domoDeviceId* è possibile associare più di un *real address* e per ogni *real address* è possibile associare un solo *domoDeviceId*.

Capitolo 4

Lato client: il domoNetClient

Il *DomoNetClient* esegue il controllo dei dispositivi in remoto indipendentemente dalla loro tecnologia.

Attua la connessione ad uno o più *web service* contemporaneamente prendendo la lista dei *domoDevice* attraverso il metodo `getDeviceList()`, e invia comandi *domoMessage* attraverso il metodo `execute(dm)`. Per qualche dispositivo è previsto anche un parametro di ritorno.

Tutti i *domoDevice* sono memorizzati nella `domoDeviceList`.

Il *domoNetClient* gestisce solamente la parte logica del lato client. Per poter interagie con essa, è necessario implementare una *user interface* tramite un'applicazione *web*, testuale o grafica.

4.1 Interazione tra il *domoNetClient* ed il *domoNetClientUI*

In figura 4.1 sono mostrate le due possibili interazioni tra la parte logica e l'interfaccia utente:

- al momento della connessione ad un *web service*:
  la *user interface* richiede la lista dei dispositivi disponibili in-

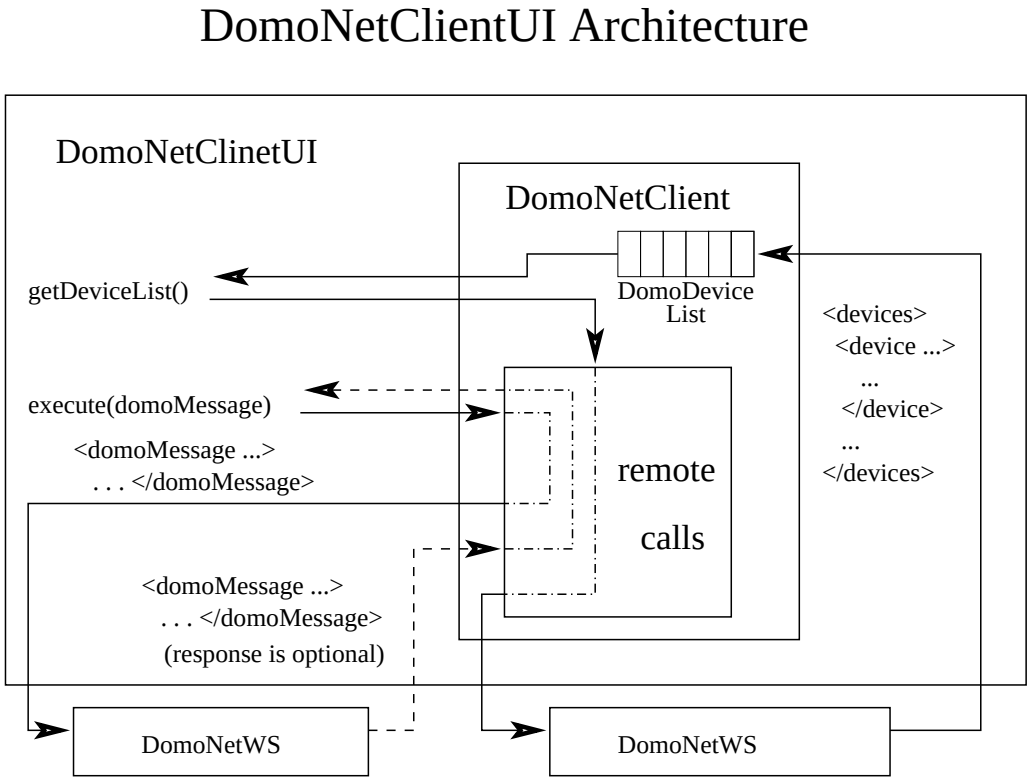

Figura 4.1: L'architettura del domoNetClient.

vocando il metodo `List getDeviceList()` **sul *domoNetClient*. Il *domoNetClient* esegue una chiamata remota indirizzata verso il *web service* richiesto creando una copia locale dei *domoDevice* nella** `DomoDeviceList` **indicizzabile tramite il *domoDeviceId*. Sia il *domoNetClient* che il *domoNetClientUI* condividono questa struttura in sola lettura; Ottenuta la lista, il *domoNetClientUI* provvede a mostrarla all'utente.**

- **per l'esecuzione di un *domoMessage*:**
  la *user interface* richiede l'esecuzione di un servizio generando il relativo *domoMessage* ed invocando il metodo `DomoMessage execute(DomoMessage domoMessage)` **sul *domoNetClient*. Il *domoNetClient* esegue una chiamata remota indirizzata verso il *web service* coinvolto e rimane in attesa di**

una risposta. Ottenuta la risposta, questa viene passata alla *domoNetClientUI* che provvede a mostrarla all'utente.

Capitolo 5

Il prototipo

5.1 Strumenti di lavoro

Il prototipo è stato realizzato grazie alla mia collaborazione con il Laboratorio di Domotica presso l'Istituto di Scienza e Tecnologie dell'Informazione ISTI "Alessandro Faedo" del CNR di Pisa. Il laboratorio mi ha messo a disposizione i dispositivi domotici sui quali è stato effettuato lo studio, le prove ed i test finali.

In particolare al laboratorio ho lavorato con il *middleware Konnex* così costituito (figura 5.1):

- bus Konnex/EIB su TP1;
- BCU (interfaccia tra PC e bus via porta seriale RS232);
- BCU (interfaccia tra PC e bus via TCP/IP);
- attuatore Siemens a 4 uscite, a cui sono state collegate altrettante lampadine;
- dimmer ABB;
- termostato Theben RAM713;
- altro termostato;

- **interruttore/regolatore;**
- **interruttore doppio + 1SA;**
- **alimentatore.**

Figura 5.1: Pannello Konnex.

Inoltre si sono dovuti utilizzare strumenti software che consentissero la configurazione del sistema. Per la configurazione della rete si è utilizzato il software *ETS3* della famiglia *ETS*.

Per *UPnP* non è stato necessario l'acquisto di componenti o di dispositivi fisici per creare l'infrastruttura di rete. Poiché questo *middleware* si basa sui protocolli dello *stack TCP/IP*, è stato sufficiente sviluppare il software su un PC provvisto di una tradizionale interfaccia di rete *Ethernet*. Per la fase di studio, prove e test finali è stato usato il simulatore di dispositivi domotici *UPnP* incluso

come esempio nelle librerie *Cyberlink*. In particolare il simulatore mi ha permesso di lavorare con i seguenti dispositivi (figura 5.2):

- **dispositivo di aria condizionata;**
- **orologio digitale con data;**
- **lampadina;**
- **televisione;**
- **lavatrice;**
- **dispositivo per il controllo remoto (telecomando).**

Figura 5.2: I dispositivi domotici UPnP *simulati utilizzati.*

5.2 Lo sviluppo

Il prototipo è un software *open source* rilasciato sotto licenza *GNU*.

Il software è stato scritto in *Java* ed usa esclusivamente strumenti *open source*:

- **il *web server Tomcat*[1];**
- **il contenitore di *web service axis*[2];**
- **la libreria per la gestione dell'*XML Xerces*[3];**
- **le librerie per l'interazione con i dispositivi domotici *Calimero*[4] e *Cyberlink*[5].**

Per promuovere a pieno titolo lo sviluppo del software *open source* è stato iniziato lo studio di piattaforme *Java* alternative a quella *Sun*[6] ed altrettanto valide come l'*ecj*: il compilatore *Java open source* di *Eclipse*[7].

Di seguito c'è la descrizione dei *package* sviluppati rimandando per eventuali approfondimenti ai *javadoc* autogenerati dai sorgenti.

5.2.1 Il *domoML*

***DomoML* è un un linguaggio contenuto in una libreria *XML* basata su *Xerces*[8].**

Struttura del *package domoML*

La struttura del *package domoML* che implementa la libreria è:

[1] http://www.apache.org
[2] http://ws.apache.org/axis
[3] http://xerces.apache.org/xerces2-j
[4] http://calimero.sourceforge.net
[5] http//sourceforge.net/projects/cgupnpjava
[6] http://java.sun.com
[7] http://www.eclipse.org
[8] http://xerces.apache.org/xerces2-j

- `DomoMLDocument`:
  classe che implementa l'interfaccia `org.w3c.dom.Document`. Ereditandola, permette di creare un nuovo *Document XML* compatibile con tutte le funzionalità di *domoML*. La classe, oltre ai metodi previsti dall'interfaccia, implementa:
  - `DataType`:
    enumerato contenente i *datatype* previsti per *domoML*;
  - `Element createElement(String tagName)`:
    metodo per aggiungere al *document* un elemento compatibile *domoML* e con tag `tagName`. Restituisce l'*Element* creato;
  - `java.lang.String toString()`:
    restituisce una *String* contenente il codice *XML* rappresentante il documento.
- `DomoMLElement`:
  classe che implementa l'interfaccia `org.w3c.dom.Element`. Rappresenta un elemento appartenente ad un *DomoMLDocument*. Ogni `createElement()` crea un oggetto che eredita da questa classe.
- `domoDevice`:
  contiene le classi che descrivono il *domoDevice*:
  - `DomoDevice`:
    classe che eredita da `DomoMLDocument`. Rappresenta il *document* della descrizione astratta dei dispositivi fatte secondo *domoML*. I metodi e i campi principali implementati per la classe sono:
    * `DomoTech`:
      enumerazione delle tecnologie supportate dai *domoDevice*;

    * `DomoDeviceService addService(...)`:
      aggiunge un servizio al *DomoDevice*;
    * `DomoDeviceService getService(String serviceName)`:
      restituisce il servizio del *domoDevice* con nome `serviceName`;
    * `java.util.List getServices()`:
      restituisce la lista di tutti i servizi del *domoDevice*;

- `DomoDeviceId`:
  permette di identificare univocamente, tramite l'`url` e l'`id`, un *domoDevice*. La classe viene usata per indicizzare principalmente *HashMap*;
- `DomoDeviceIdWithDescription`:
  come il `DomoDeviceId` ma con in più un attributo contenente anche la descrizione del dispositivo;
- `DomoDeviceSAXParser`:
  parser per ottenere da uno *strem XML* una lista o un solo *domoDevice*;
- `DomoDeviceService`:
  classe che eredita da `DomoMLElement`. Descrive il servizio appartenente ad un *domoDevice*. I metodi e i campi principali implementati per la classe sono:
    * `DomoDeviceServiceInput addInput(...)`:
      aggiunge un campo input al servizio;
    * `java.util.List getInputs()`:
      restituisce la lista di tutti gli input legati al servizio;
    * `DomoDeviceServiceInput addLinkedService(...)`:
      aggiunge un campo per implementare la interoperabilità tra *domoDevice*;

    * `java.util.List getLinkedSerivices()`:
      restituisce la lista di tutti i servizi legati ad altri *domoDevice*;

- `DomoDeviceServiceInput`:
  classe che eredita da `DomoMLElement`. Descrive il tipo di input legato ad un servizio appartenente ad un *domoDevice*; I metodi e i campi principali implementati per la classe sono:

    * `DomoDeviceServiceInputAllowed addAllowed(...)`:
      aggiunge un valore ammesso per il tipo di *input*;
    * `java.util.List getAllowed()`:
      restituisce la lista i valori ammessi per l'*input*;
    * `DomoDeviceServiceInput addLinkedService(...)`:
      aggiunge un campo per implementare la interoperabilità tra *domoDevice*;
    * `java.util.List getLinkedSerivices()`:
      restituisce la lista di tutti i servizi legati ad altri *domoDevice*;

- `DomoDeviceServiceInputAllowed`:
  classe che eredita da `DomoMLElement`. Descrive un input ammesso per il servizio;

- `DomoDeviceServiceLinkedService`:
  classe che eredita da `DomoMLElement`. Descrive come operare per eseguire l'interoperabilità tra *domoDevice*; I metodi e i campi principali implementati per la classe sono:

    * `DomoDeviceServiceLinkedServiceInput addLinkedServiceInput(...)`:

aggiunge un'associazione tra gli *input* dei servizi in gioco;

* `java.util.List getInputs()`:
restituisce la lista delle associazioni degli *input*;

– `DomoDeviceServiceLinkedServiceInput`:
classe che eredita da `DomoMLElement`. Descrive l'associazione degli *input* tra i servizi in gioco;

- `domoMessage`:
contiene le classi che descrivono il *domoMessage*:

  – `DomoMessage`:
  classe che eredita da `DomoMLDocument`. Rappresenta il *document* della descrizione interazioni tra i dispositivi fatte secondo *domoML*. I metodi e i campi principali implementati per la classe sono:

    * `MessageType`:
    enumerazione dei tipi di messaggi ammessi;
    * `DomoMessageInput addInput(...)`:
    aggiunge un valore da allegare al contenuto del *domoMessage*;
    * `DomoMessageInput getInput(String inputName)`:
    restituisce l'*input* del *domoMessage* con nome `serviceName`;
    * `java.util.List getInputs()`:
    restituisce la lista di tutti gli *input* del *domoMessage*;

  – `DomoMessageInput`:
  classe che eredita da `DomoMLElement`. Descrive il tipo di input legato ad un servizio appartenente ad un *domoMessage*;

– `DomoMessageSAXParser`:
parser per ottenere da uno *strem XML* un *domoMessage*.

Costruire un *domoDevice*

È possibile costruire un nuovo *domoDevice* invocando i costruttori della classe in due maniere differenti:

1. **attraverso righe di codice:**

```
// crea un tag chiamato "device"
DomoDevice domoDevice = new DomoDevice();
// riempie il tag
domoDevice.setDescription(
                "lampada a risparmio energetico");
domoDevice.set...
...
```

oppure in una riga:

```
DomoDevice domoDevice =
           new DomoDevice(DomoDevice.DomoTech.KNX,
                                "lampada", ...);
```

per poi continuare con:

```
// aggiungo il servizio per settare lo status
DomoDeviceService service = domoDevice.addService(
            "SET_STATUS", "Set the status",
            "status");
// e l'unico input che ha
DomoDeviceServiceInput input =
            service.addInput("status", "The value",
                      DomoDevice.DataType.BOOLEAN);
```

```
// con i possibili valori
input.addAllowed("TRUE");
input.addAllowed("FALSE");
...
```

2. **invocare il costruttore passando una stringa contenente il codice xml:**

```
DomoDevice domoDevice =
      new DomoDevice("<device description=\"lampada
          a risparimo energetco\" id=\"0\",
          manufacturer=\"pholips\",
          postionDescription=\"comodino accanto al
          letto\", serialNumber=\"xxxxxxxxx\"
          tech=\"KNX\" type=\"lampada\" url=\"\">
    <service description=\"Get the status\"
     name=\"GET_STATUS\" output=\"BOOLEAN\"
     outputDescription=\"The value\"
     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>
    </service>
</device>");
```

Costruire un *domoMessage*

È possibile costruire un nuovo *domoMessage* invocando i costruttori della classe in due maniere differenti:

1. **attraverso righe di codice:**

```
// crea un tag chiamato "device"
DomoMessage domoMessage = new DomoMessage();
// riempie il tag
domoMessage.setMessage("SET_STATUS");
domoMessage.setMessageType(
            DomoDevice.MessageType.COMMAND);
domoMessage.set...
...
```

   oppure in una riga:

```
DomoMessage domoMessage = new DomoMessage(
         "http://www.altrowebservice.it/servizio",
         "1",
         "http://www.questowebservice.it/servizio",
         "0",
         "SET_STATUS",
         DomoDevice.MessageType.COMMAND);
```

   per poi continuare con:

```
// aggiungo l'input
domoMessage.addInput("status","TRUE",
         DomoMessage.DataType.BOOLEAN);
```

2. **invocare il costruttore passando una stringa contenente il codice xml:**

```
DomoMessage domoMessage =
  new DomoMessage("<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>");
```

5.2.2 Il *web service*

Il *web service* è implementato usando il *web container Tomcat* ed il contenitore di servizi *axis*, entrambi *open source*.

Struttura del *package domoNetWS*

La struttura del *package domoNetWS* che implementa il *web service* è:

- `DomoNetWS`:
  classe che implementa un *web service*. I metodi principali implementati sono:
  - `deviceList`:
    ***HashMap* che contiene la lista dei *domoDevice* indicizzati per `DomoDeviceId`;**
  - `managerList`:
    ***HashMap* che contiene la lista dei *tech manager* indicizzati per `domoML.domoDevice.DomoDevice.DomoTech`;**

  - `DomoDeviceId addDomoDevice(DomoDevice domoDevice)`:
    aggiunge un *domoDevice* alla lista dei *domoDevice* assegnando un `DomoDeviceId` univoco;
  - `java.lang.String execute(String messageString)`:
    esegue un *domoMessage* riconoscendo il *domoDevice* destinatario del messaggio;
  - `void finalize()`: **azioni da eseguire al termine allo *shutdown* del *web service*;**
  - `DomoDevice getDomoDevice(DomoDeviceId domoDeviceId)`:
    restituisce un *DomoDevice* dandone il descrittore;
  - `java.lang.String getDomoDeviceList()`:
    restituisce uno *stream XML* rappresentante la lista dei *domoDevice* raggiungibili dal *web service*;
  - `void printLicenceNote()`:
    mostra la licenza del *software*;
  - `boolean removeDomoDevice()`:
    rimuove un *domoDevice* dalla lista dei *domoDevice* disponibili;
  - `void searchAndExecuteLinkedServices(...)`:
    ricerca eventuali *tag* per l'interoperazione tra i *domoDevice*.

- `techManager`:
  contiene le classi che implementano i *tech manager*.
  - `DoubleHash`:
    classe che implementa una *HashMap* bidirezionale (può

essere indizzato sia tramite un *real address* per ottenere un *domoDeviceId* e vice versa.);

- `TechManager`:
  classe con metodi astratti dalla quale devono ereditare tutti i *tech manager* in modo che siano compatibili con *domoNet*. I metodi astratti previsti sono:
  - `void addDevice(DomoDevice domoDevice, String address)`:
    aggiunge un *domoDevice* al *tech manager* fornendo il *real address*;
  - `DomoMessage execute(DomoMessage domoMessage)`:
    esegue un *domoMessage* e restituisce una risposta al termine;
  - `void finalize()`:
    azioni da compiere alla chiusura del *tech manager*;
- `knxManager`:
  classe che eredita da `TechManager`. contiene le classi che implementano le funzionalità per la tecnologia *Konnex*.
  - `KNXManager`:
    implementa le funzionalità per interagire con i dispositivi *Konnex*;
  - `KNXManagerConfigurationSAXParser`:
    classe per parsare il file di configurazione di *ETS*;
- `upnpManager`:
  classe che eredita da `TechManager`. contiene le classi che implementano le funzionalità per la tecnologia *UPnP*.
  - `UPNPManager`:
    implementa le funzionalità per interagire con i dispositivi *UPnP* e col *web service*;

* `UPNPManagerPoint`:
  listener* per gestire i dispositivi *UPnP;

Come implementare un nuovo *tech manager*

Per implementare un nuovo modulo *tech manager*, è necessario estendere la classe `domoNetWS.techManager.TechManager` e creare una nuova istanza della nuova classe nella `domoNetWS.DomoNetWS.managerList` usando come chiave l'identificatore della tecnologia (essa deve essere contenuta in `domoML.domoDevice.DomoDevice.DomoTech`). Il nuovo modulo deve implementare i metodi astratti che sono:

- `public void addDevice(final DomoDevice domoDevice, String address)`:
  per aggiungere un dispositivo alla lista dei *domoDevice* del *web service*; deve chiamare i metodi `doubleHash.add()` e `addDomoDevice()` del *web service*;

- `public DomoMessage execute(DomoMessage domoMessage) throws Exception`:
  per eseguire un `domoML.domoMessage.DomoMessage`; deve essere in grado di convertire il *domoMessage* in *tech message* e vice versa;

- `public void finalize()`:
  azioni da eseguire quando viene chiuso il *web service*.

I *tech manager* implementati

Questi sono i *tech manager* implementati e che possono essere presi come esempio per quelli nuovi.

KNXManager

Il *KNXManager* è il modulo che gestisce la tecnologia *Konnex* e si basa sulle librerie *open source Calimero*[9].

Il costruttore del modulo prende come parametro la *url* e il numero di porta del servizio che è connesso ai dispositivi.

Al tempo di *startup* esegue la connessione al server e l'inizializzazione delle strutture che si occupano della conversione dei *datatype*, esattamente da `domoML.DomoDevice.domoDevice.DataType` agli identificatori `Major` e `Minor`, usati per convertire il *tag* `input` del *domoMessage* in un valore fruibile per la tecnologia. Dopo le inizializzazioni il modulo carica i dispositivi disponibili parsando un file *XML* generato dall'applicazione *ETS 3*[10] già presentata nel capitolo riguardante *Konnex*.

Il file di configurazione contiene le descrizioni e tutte le funzionalità che il sistema *Konnex* può offrire.

Come esempio, di seguito è mostrato un pezzo di file di configurazione per il servizio di un dispositivo:

```
<row>
  <colValue nr="1">0/0/1 Comando Uscita A</colValue>
  <colValue nr="2">
    0: Comando,tasto sinistro superiore-Commutazione
  </colValue>
  <colValue nr="3">
    1.1.4 16196.. Pulsante 4 canali
  </colValue>
  <colValue nr="4">S</colValue>
  <colValue nr="5">-</colValue>
  <colValue nr="6">C</colValue>
```

[9] http://calimero.sourceforge.net

[10] *ETS* (EIB Tools Software) è un software distribuito dalla *Konnex Association* per settare e configurare dispositivi.

```
  <colValue nr="7">-</colValue>
  <colValue nr="8">W</colValue>
  <colValue nr="9">T</colValue>
  <colValue nr="10">U</colValue>
  <colValue nr="11">
    16196.. Pulsante 4 canali
  </colValue>
  <colValue nr="12">
    16196 On-Off-Dimmer-Tapparelle-Led
  </colValue>
  <colValue nr="13">1 bit</colValue>
  <colValue nr="14">Basso</colValue>
  <colValue nr="15">0/0/1</colValue>
</row>
```

Da questo pezzo, e da ogni tag `row` presente nel file, è possibile individuare:

- **nel secondo `colValue`: il nome del servizio;**
- **nel terzo `colValue` (la parte iniziale): il *real address* del dispositivo;**
- **nel settimo `colValue`: se il campo è leggibile (settato a `R`);**
- **nell'ottavo `colValue`: se il campo è scrivibile (settato a `W`);**
- **nel nono `colValue`: se il campo è trasmissibile (settato a `T`);**
- **nell'undicesimo `colValue`: il nome del device che offre il servizio;**
- **nel dodicesimo `colValue`: la descrizione del servizio;**
- **nel quindicesimo `colValue`: l'indirizzo di gruppo associato al servizio;**

Gli altri `colValue` **non sono significativi allo scopo proposto dal *KNXManager*. Interessano i servizi che sono trasmissibili e almeno o leggibili o scrivibili.**

Un dispositivo viene descritto in tutti i suoi servizi unendo insieme tutti gli undicesimi `colValue` **con la stessa etichetta.**

Questo esempio produce il seguente *domoDevice*:

```
<device description="" id="" manufacturer=""
 postionDescription="" serialNumber="" tech="KNX"
 type="16196.. Pulsante 4 canali" url="">
    <service description="Get the status"
     output="BOOLEAN" outputDescription="The value"
     prettyName="Get status" name="GET_STATUS" />
    <service name="0/0/1"
     description="16196 On-Off-Dimmer-Tapparelle-Led"
     prettyName="16196 On-Off-Dimmer-Tapparelle-Led">
        <input description="The value" name="value"
         type="BOOLEAN">
            <allowed value="TRUE" />
            <allowed value="FALSE" />
        </input>
    </service>
</device>
```

***Come catturare i* tech message *dal bus* Konnex**
Tutti i messaggi che transitano sul bus *Konnex* vengono catturati dal *KNXManager* attraverso l'uso di un *listener* in ascolto sul *bus*. È interessante monitorare il *bus* per due scopi:

1. **in attesa di risposta derivata dall'esecuzione di un *domoMessage*:**
   in questo caso è stato implementato un sistema a semafori dove i produttori sono i dispositivi che scrivono sul bus e il

tech manager **è il consumatore. Il *real address* del mittente di ogni messaggio che transita sul bus viene confrontato con quello di cui siamo in attesa. Se l'indirizzo viene riconosciuto in un limite di tempo prefissato viene generato con i dati del *tech message* il *domoMessage* di risposta altrimenti viene generato un *domoMessage* di fallimento. Un esempio tipico di questo meccanismo è l'interrogazione di un dispositivo per conoscerne lo stato;**

2. **per implementare la cooperazione:**
   ogni messaggio che transita sul bus viene catturato dal *listener*. Il *tech message* estrapola il *real address* del dispositivo mittente e riconosce il servizio tramite l'indirizzo di gruppo a cui è rivolto il comando. Con questi dati, è possibile ricavare il *domoDevice* e la descrizione del servizio. Il *web service* può quindi verificare se è impostato il tag `linkedService` ed agire di conseguenza.

UPnPManager **L'*UPnPManager* è il modulo che gestiche la tecnologia *UPnP* e si basa sulla libreria *Cyberlink*[11].**

A tempo di *startup*, il l'*UPnPManager* inizializza il `ManagerPoint`: il *listener*, cuore del manager, che permette di aggiungere, rimuovere, testare l'*hartbeat* e amministrare i pacchetti dei dispositivi. Il `ManagerPoint` provvede ad inizializzare le strutture dati per la conversione da `domoML.DomoDevice.domoDevice.DataType` ai simboli usati per questa tecnologia e vice versa.

Quando viene aggiunto un dispositivo alla rete *UPnP*, questo si annuncia dando la propria descrizione e quella dei servizi che può offrire. Queste informazini sono memorizzate in diversi file *XML* che la libreria provvede a parsare e a darne l'albero corrispondente.

[11] http//sourceforge.net/projects/cgupnpjava

A questo punto diventa facile ottenere le informazioni cercate. Un esempio di descrizione del dispositivo è:

```
<device>
 <deviceType>urn:schemas-upnp-org:device:light:1
 </deviceType>
 <friendlyName>CyberGarage Light Device</friendlyName>
 <manufacturer>CyberGarage</manufacturer>
 <manufacturerURL>http://www.cybergarage.org
 </manufacturerURL>
 <modelDescription>CyberUPnP Light Device
 </modelDescription>
 <modelName>Light</modelName>
 <modelNumber>1.0</modelNumber>
 <modelURL>http://www.cybergarage.org</modelURL>
 <serialNumber>1234567890</serialNumber>
 <UDN>uuid:cybergarageLightDevice</UDN>
 <UPC>123456789012</UPC>
 <iconList>
  ...
 </iconList>
 <serviceList>
  ...
 </serviceList>
 <presentationURL>http://www.cybergarage.org
 </presentationURL>
</device>
```

Da questo pezzo di codice si trova:

- **dal tag `friendlyName`: la descrizione del dispositivo;**
- **dal tag `deviceType`: il *real address* del dispositivo;**

- **dal tag** `modelName`**: il tipo del dispositivo;**
- **dal tag** `serialNumber`**: il serial number.**

Quello dei servizi offerti (chiamati `actions`**) è:**

```
<actionList>
 <action>
  <name>SetPower</name>
  <argumentList>
   <argument>
    <name>Power</name>
    <relatedStateVariable>
       Power
    </relatedStateVariable>
    <direction>in</direction>
   </argument>
   <argument>
    <name>Result</name>
    <relatedStateVariable>
       Result
    </relatedStateVariable>
    <direction>out</direction>
   </argument>
  </argumentList>
 </action>
 <action>
  <name>GetPower</name>
  <argumentList>
   <argument>
    <name>Power</name>
    <relatedStateVariable>Power</relatedStateVariable>
    <direction>out</direction>
```

```
   </argument>
  </argumentList>
 </action>
</actionList>

<serviceStateTable>
 <stateVariable sendEvents="yes">
  <name>Power</name>
  <dataType>boolean</dataType>
  <allowedValueList>
   <allowedValue>0</allowedValue>
   <allowedValue>1</allowedValue>
  </allowedValueList>
  <allowedValueRange>
   <maximum>123</maximum>
   <minimum>19</minimum>
   <step>1</step>
  </allowedValueRange>
 </stateVariable>
 <stateVariable sendEvents="no">
  <name>Result</name>
   <dataType>boolean</dataType>
 </stateVariable>
</serviceStateTable>
```

Da questo pezzo di codice si deduce:

- **dal tag** `action - name`**: il nome e il *pretty name* del servizio;**
- **dal tag** `argument - name`**: il nome del parametro di input;**
- **dal tag** `relatedStateVariable`**: la chiave da cercare nella** `serviceStateTable` **per trovare le altre informazioni del parametro;**

- **dal tag** `direction`**: se è** `out` **è un parametro di output quindi è aspettato un valore di ritorno all'esecuzione del servizio; se è** `in` **è un parametro di input;**
- **dal tag** `datatype`**: il datatype del parametro;**
- **dal tag** `allowedValue`**: i valori permessi per quel parametro.**

Per ogni `action` **è previsto un set di** `argument`**. Ogni** `argument` **ha un** `datatype` **da essere poi convertito in** `domoML.DomoDevice.domoDevice.DataType`**. Di ogni** `argument` **occorre verificare il tag** `direction` **per sapere se la variabile di stato deve essere usata come parametro di *input* o di *output*.**

Il corrispondente *domoDevice* risultante è:

```
<device description="CyberGarage Light Device" id=""
 manufacturer=""  positionDescription=""
 serialNumber="1234567890" tech="UPNP" type="Light">
   <service description="" name="SetPower"
    output="BOOLEAN" prettyName="SetPower">
     <input description="" name="Power"
      type="BOOLEAN">
       <allowed value="0" />
       <allowed value="1" />
     </input>
   </service>
   <service description="" name="GetPower"
    output="BOOLEAN" prettyName="GetPower" />
</device>
```

***Come catturare i* tech message *dalla rete* UPnP**

L'`UPnPManagerPoint` **implementa anche le funzioni di *listener* in attesa di messaggi che transitano sulla rete. Quando un nuovo messaggio transita, ne viene calcolato il codice di sottoscrizione del**

servizio e il *domoDeviceId* del dispositivo mittente. Con queste informazioni, passate al *web service*, è possibile verificare la presenza di `linkedService` associati al servizio invocato.

5.2.3 Il *DomoNetClient*

il *DomoNetClient* permette di controllare in remoto i dispositivi domotici dislocati nella rete *DomoNet* all'interno dei *web service*.

Struttura del *package domoNetClient*

La struttura del *package domoNetClient* che implementa il *client* è:

- `DomoNetClient`:
  classe che implementa la parte logica del lato *client*. Per poter essere utilizzata occore utilizzare una classe che implementa una *user interface* grafica, testuale, web etc. I metodi e i campi principali implementati per la classe sono:
  - `domoDeviceList`:
    ***HashMap* rappresentante una copia locale dei *domoDevice* scaricati al momento della connessione ai *web service* interessati;**
  - `void connectToWebService(String URL, String description)`:
    si connette ad un *web service* richiedendo la lista dei dispositivi disponibili;
  - `java.lang.String execute(DomoMessage domoMessage)`:
    esegue un *domoMessage* individuando il *web service* interessato.

- **domoNetClientUI:**
  contiene le classi che descrivono il *domoNetClientUI*, il *client* grafico:
  - `DomoNetClientUI`:
    classe che costruisce il *client* grafico da attaccare al `DomoNetClient`;
  - `DefaultWebServicesDescriptorsSAXParser`:
    parser del file di configurazione del *client* grafico.

Un *domoNetClientUI* grafico

Questa interfaccia implementa un client grafico da attaccare al motore *DomoNetClient*.

Il client è fornito da un file di configurazione basato su *XML* dal quale prende le informazioni sui possibili *web service* disponibili. Gli indirizzi dei *web service* sono mostrati in alto contestualmente ad una breve descrizione. La barra dell'indirizzo può essere comunque editata.

Selezionato o editato l'indirizzo del *web service*, cliccando sul bottone `Connect` si esegue materialmente la connessione al *web service*. Il *web service* risponde dando una lista dei *domoDevice* attualmente disponibili (l'insieme dei dispositivi di tutte le tecnologie supportate).

I *domoDevice* vengono mostrati divisi per *web service* con una struttura ad albero (figura 5.3).

Cliccando su uno di essi viene aperta una nuova finestra costruita a *run time* parsando la descrizione del *domoDevice*. La finestra mostra delle informazioni generali sul dispositivo ed i servizi disponibili.

Per ogni servizio viene mostrato il bottone etichettato con il `pretty name`. Alla sua destra è presente un campo testuale se, all'esecuzione del servizio, è atteso un valore di ritorno. Alla sua

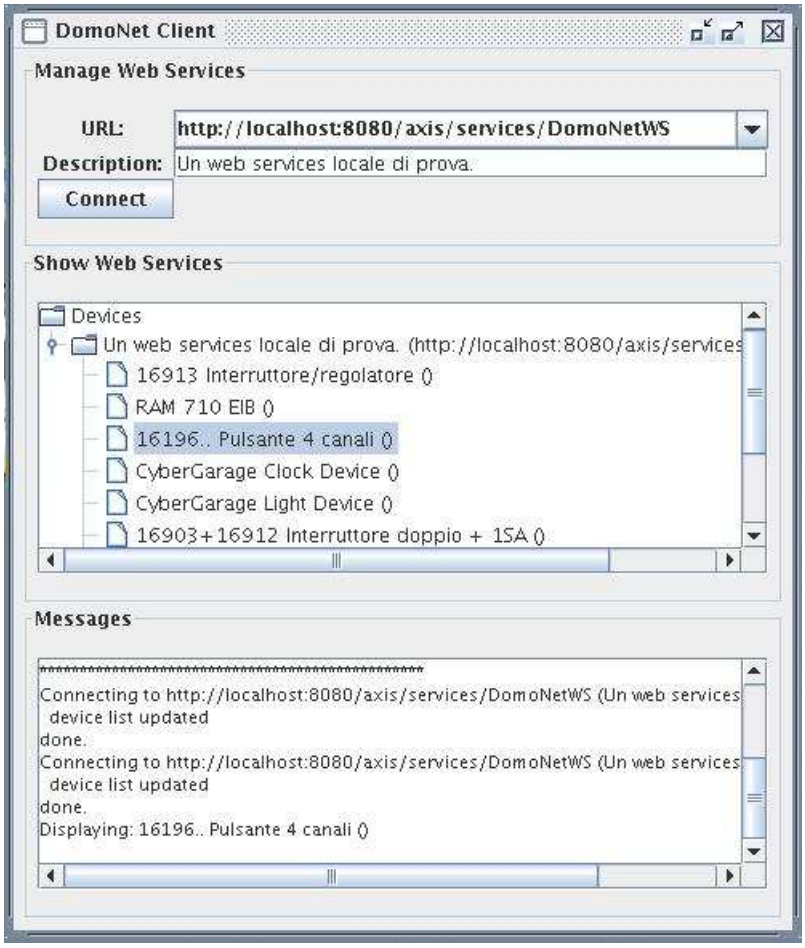

Figura 5.3: Snapshot del domoNetClientUI (finestra principale).

sinistra esistono tanti campi testuali quanti parametri di input richiesti per quel servizio. Accanto ad ogni campo testuale è presente una descrizione del tipo di dato atteso.

Riempiendo correttamente i parametri di input richiesti e cliccando sul bottone del servizio, il client genera il *domoMessage* corrispondente da inviare al *domoNetClient* il quale provvede ad instradarlo al *web service* di competenza.

In figura 5.4 è mostrata la finestra dei servizi relativi ad un televisore:

- `SetPower`: **per settare lo stato del televisore. É necessario un**

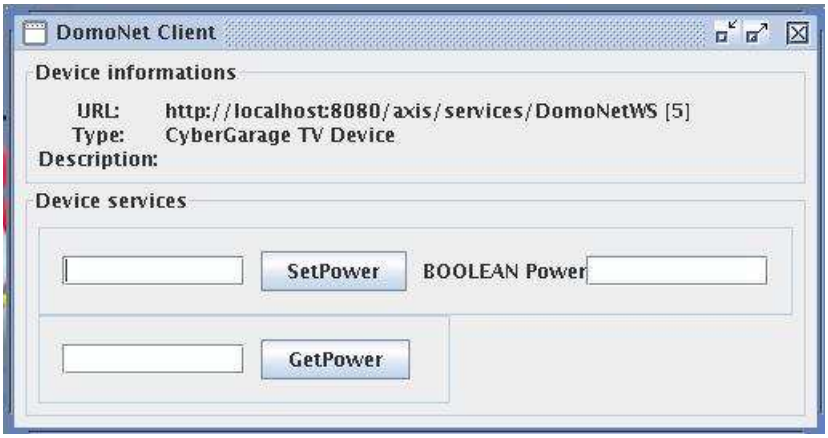

Figura 5.4: Snapshot del domoNetClientUI (finestra dei servizi 1).

parametro ***booleano*** in ***input*** e atteso un parametro di ***output*** come risposta;

- `GetPower`: restituisce lo stato attuale del televisore con un parametro di ***output***.

In figura 5.5 è mostrata la finestra dei servizi di una pulsantiera a quattro canali con un servizio per pulsante. Su di essa è possibile invocare solamente servizi che determinano la commutazione dei pulsanti tramite un valore ***booleano*** "0" o "1".

5.3 Il test

Sono stati effettuati numerosi ***test*** di valutazione per verificare il corretto funzionamento del ***software*** sia per quanto riguarda il controllo remoto sia per quanto riguarda l'interoperabilità e non sono stati riscontrati problemi.

Per brevità viene riportato solamente il caso più interessante e semplice tra i ***test*** effettuati: l'interoperazione tra un interruttore ***Konnex*** per accendere una lampada ***UPnP*** e l'interoperazione tra un interruttore ***UPnP*** per accendere una lampada ***Konnex***.

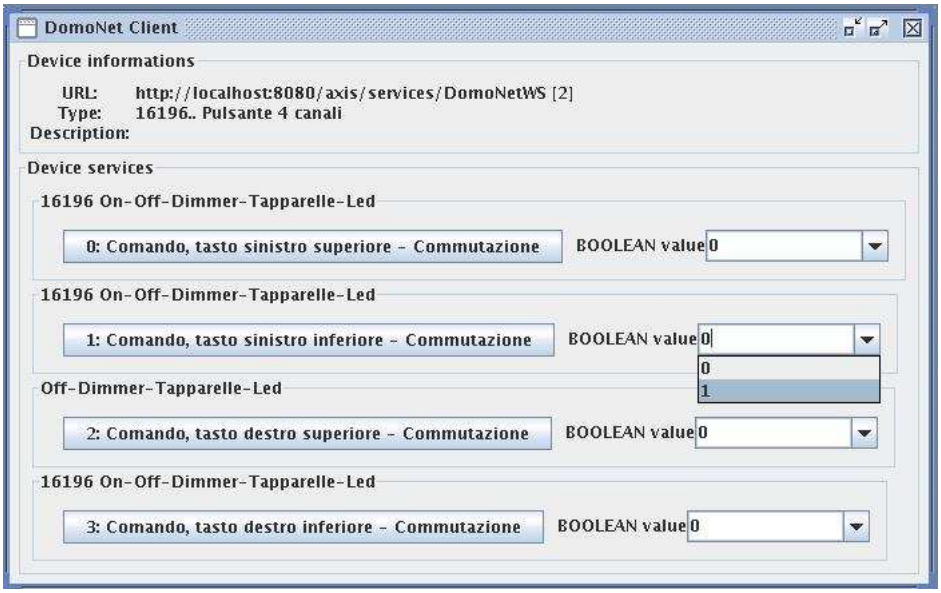

Figura 5.5: Snapshot del domoNetClientUI (finestra dei servizi 2).

5.3.1 Interruttore *Konnex* con lampada *UPnP*

Come interruttore viene usato l'attuatore *Siemens* a 4 uscite e la lampada *UPnP* simulata. Il rispettivo *domoDevice* dell'attuatore è:

```
<device serialNumber="" tech="KNX"
 id="0" manufacturer="" positionDescription=""
 description="16196 On-Off-Dimmer-Tapparelle-Led"
 type="16196.. Pulsante 4 canali" url="">
   <service
    description="16196 On-Off-Dimmer-Tapparelle-Led"
    name="0/0/1"
    prettyName="0: Comando,tasto sinistro superiore">
      <input description="" name="value"
       type="BOOLEAN">
        <allowed value="0" />
        <allowed value="1" />
```

```
    </input>
  </service>
  <service
   description="16196 On-Off-Dimmer-Tapparelle-Led"
   name="0/0/2"
   prettyName="1: Comando,tasto sinistro inferiore">
    <input description="" name="value"
     type="BOOLEAN">
      <allowed value="0" />
      <allowed value="1" />
    </input>
    <linkedService id="1" service="SetPower" url="">
      <linkedInput from="value" to="Power" />
    </linkedService>
  </service>
  <service
   description="16196 On-Off-Dimmer-Tapparelle-Led"
   name="0/0/3"
   prettyName="2: Comando, tasto destro superiore">
    <input description="" name="value"
     type="BOOLEAN">
      <allowed value="0" />
      <allowed value="1" />
    </input>
  </service>
  <service
   description="16196 On-Off-Dimmer-Tapparelle-Led"
   name="0/0/4"
   prettyName="3: Comando, tasto destro inferiore">
    <input description="" name="value"
     type="BOOLEAN">
```

```
        <allowed value="0" />
        <allowed value="1" />
      </input>
    </service>
</device>
```

Con la pressione del tasto sinistro inferiore dell'attuatore (figura 5.6)viene implementata l'interoperabilità con il *domoDevice* sullo stesso *web service* e con *id* "1".

Figura 5.6: Pulsantiera Konnex *a 4 canali.*

Il rispettivo *domoDevice* della lampada è:

```
<device description="" id="1" manufacturer=""
 positionDescription="" serialNumber="1234567890"
 tech="UPNP" type="CyberGarage Light Device">
```

```
   <service description="" name="SetPower"
    output="BOOLEAN" prettyName="SetPower">
     <input description="" name="Power"
      type="BOOLEAN" />
   </service>
   <service description="" name="GetPower"
    output="BOOLEAN" prettyName="GetPower" />
</device>
```

Alla pressione del pulsante la lampada si accende (figura 5.7) e alla ripressione la lampada si spegne.

Figura 5.7: Accensione lampada UPnP *alla pressione del tasto della pulsantiera* Konnex.

5.3.2 Interruttore *UPnP* con lampada *Konnex*

Per questo *test* invece di interoperare con lo stato di un interruttore, interoperiamo con lo stato della lampada *UPnP* e con l'attuatore *Konnex* del paragrafo precedente. Il risultato atteso è l'accensione e lo spengimento contemporaneo della lampada *UPnP* e *Konnex*, dimostrando così la flessibilità del software sviluppato.

Il *domoDevice* della lampada *UPnP* è:

```
<device description="CyberGarage Light Device" id=""
 manufacturer=""  positionDescription=""
```

```
 serialNumber="1234567890" tech="UPNP" type="Light">
   <service description="" name="SetPower"
    output="BOOLEAN" prettyName="SetPower">
     <input description="" name="Power" type="BOOLEAN">
       <allowed value="0" />
       <allowed value="1" />
     </input>
     <linkedService id="0" service="0/0/2" url="">
       <linkedInput from="Power" to="value" />
     </linkedService>
   </service>
   <service description="" name="GetPower"
    output="BOOLEAN" prettyName="GetPower" />
</device>
```

Il *domoDevice* della pulsantiera è analoga al paragrafo precedente ad eccezione dell'assenza del tag `linkedService`.

Cliccando sul tasto *Light Power* del telecomando *UPnP* (figura 5.8)

le lampade si accendono (figura 5.9) e alla ripressione le lampade si spengono.

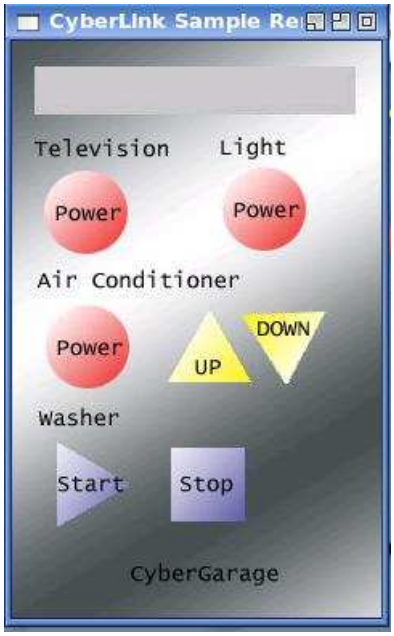

Figura 5.8: Dispositivo di controllo remoto UPnP.

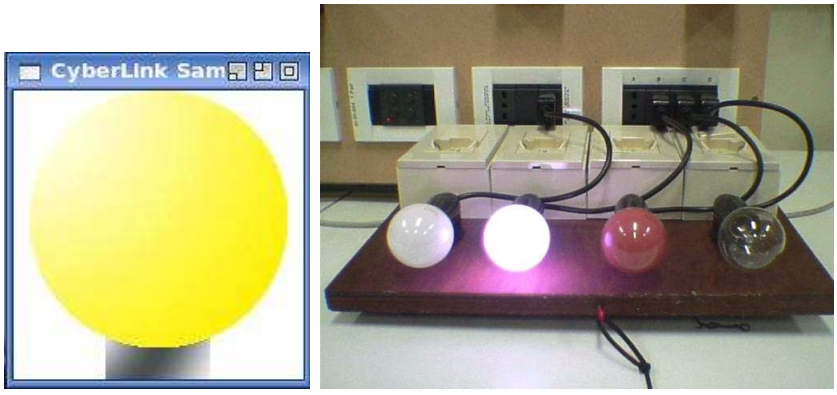

Figura 5.9: Accensione lampade Konnex *e* UPnP.

Capitolo 6

Conclusioni

6.1 Realizzazioni sperimentali e valutazioni

I test sul software effettuati al Laboratorio di Domotica presso l'Istituto di Scienza e Tecnologie dell'Informazione ISTI "Alessandro Faedo" del CNR di Pisa, hanno dimostrato l'efficacia e la generalità del motore del software. Il software è stato in grado di importare correttamente tutti i dispositivi domotici sottoposti alla prova come *dimmer*, termostati, interruttori, televisioni, lavatrici e lampadine. Il lato client è riuscito ad interagire correttamente con i dispositivi sottoposti, mostrando tutti i servizi disponibili, costruendo una corretta interfaccia per ognuno di essi. Per la cooperazione, il punto cruciale è l'individuazione del messaggio che passa sulla via di comunicazione della tecnologia in oggetto, il riconoscimento del dispositivo mittente (per trovare il corrispettivo *domoDevice*) ed il corrispondente servizio invocato. Per quel che riguarda le tecnologie sviluppate, queste informazioni sono sempre state individuate ed elaborate correttamente quindi è stato sempre possibile verificare la presenza di richieste di cooperazione ed eventualmente soddisfarle.

Da un punto di vista personale, l'esperienza presso il Labora-

torio di Domotica ha contribuito a fornirmi nuovi ed interessanti bagagli tecnici in un settore che, nonostante l'auspicabilissima espansione, stenta ancora a decollare e mantiene di fatto un ruolo di nicchia nel campo dell'informatica moderna. Confrontarmi con i continui interrogativi, sia teorici che pratici, che inevitabilmente sorgono in un progetto di questo calibro, ha rappresentato inoltre una sfida con me stesso: dove le conoscenze e gli strumenti già accessibili finiscono, lì si attivano l'intelligenza e la creatività, e ogni piccolo risultato diventa una conquista e un nuovo stimolo. Gli obiettivi prefissati si possono dire complessivamente raggiunti e, malgrado la modestia e l'umiltà non ci portino ad affermare che la nostra soluzione al problema dell'interoperabilità tra i *middleware* domotici sia in assoluto quella giusta, con un pizzico di convinzione credo almeno di aver dato un qualche contributo.

6.2 Integrazione con Internet

Il modello proposto si integra in maniera perfetta con Internet in quanto alla base del suo funzionamento giocano un ruolo fondamentale i protocolli e gli strumenti *standard* e *open* della rete, come *XML* ed i *web service*. Tali strumenti sono utilizzati per le comunicazioni tra i diversi componenti architetturali e più precisamente tra *domoNetClient* e *domoNetWS* per implementare il controllo remoto (figura 1.3) e tra *domoNetWS* diversi per l'interoperabilità (figura 3.3).

Inoltre, essendo il prototipo rilasciato sotto i termini di licenza *GNU* ed appoggiandosi esclusivamente a *software open source*[1], può essere pubblicato e distribuito sulla rete senza alcun vincolo.

[1] vedere il capitolo riguardante il prototipo

6.3 Direzioni future di ricerca

La domotica, allo stato attuale, è tuttora ostacolata dal tentativo di ciascuna industria di imporre il proprio standard sugli altri. La presenza di un così vasto numero di standard domotici, oramai ampiamente affermati, indica che difficilmente vi sarà la definitiva consacrazione di uno solo di essi.

Il software realizzato rappresenta una soluzione praticabile che possa garantire la definitiva affermazione della domotica in quanto risolve in maniera definitiva, con un'architettura semplice e pulita, l'ostacolo posto.

Occore ancora però molto lavoro. In primo luogo occorre continuare lo sviluppo del software. Questo implementa solamente le funzionalità di base atte a dimostrare l'efficacia dell'architettura presentata ma ancora lontano da poter sfruttare tutte le potenzialità offerte. Non di minore importanza ricopre lo sviluppo di altri *tech manager* in modo da poter ampliare il parco delle tecnologie supportate.

Il software presentato, inoltre, rappresenta solo un esempio di applicazione per architettura proposta. Aumentando ulteriormente il grado di generalità dell'architettura è sicuramente possibile trovare ulteriori ambiti d'utilizzo dove è necessaria una cooperazione tra dispositivi e sensori di qualunque tipo di fatto diversi e incompatibili.

Bibliografia

- **Per l'introduzione alla domotica**
  - **Bianchi Bandinelli R., Lucidi del corso di Domotica (CLS Informatica, a.a. 2005/06)**
    www.di.unipi.it/ bandinel/domotica.html
  - **Jini Community, http://www.jini.org**
  - **HAVi Organization, http://www.havi.org**
  - **Echelon Co., "Introduction to the LonWorks Platform: An Overview of Principles and Practices, v2.0", 2001**
  - **X10 Home Page, http://www.x10.com**
- **Per Konnex**
  - **Konnex Association, http://www.konnex.org**
  - **Konnex Standard System Specification: Architecture Vol.3 Part I (Konnex Association, 2001)**
    (non di dominio pubblico)
- **Per UPnP**
  - **UPnP Forum, http://www.upnp.org**
  - **UPnP Device Architecture 1.0 (UPnP Forum, 2003)**
    http://www.upnp.org/resources/documents/
    CleanUPnPDA101-20031202s.pdf

- Per i Web Services
  - Kreger H., IBM Web Services Conceptual Architecture 1.0 (IBM Software Group, 2001)
  - Newcomer E., Understanding Web Services: XML, WSDL, SOAP and UDDI (Addison Wesley Professional, 2002)
  - Snell J., Programming Web Services with SOAP (O'Reilly, 2001)
  - Cerami E., Web Services Essential (O'Reilly, 2002)
  - http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf
- Per lo sviluppo
  - Tokunaga E. et al., A Framework for Connecting Home Computing Middleware (Department of Information and Computer Science Waseda University - Tokyo - IEEE, 2002)
  - Chemishkian S., Lund J., Experimental Bridge LonWorks/UPnP 1.0 (Consumer Communications and Networking Conference, 2004)
  - Papazoglou, M.P. and Georgakopolous, D. "Service Oriented Computing". In *Communications of the class just created and obtain a relative ACM 46, 10.* (2003), 42-47.
  - Lee, C.E. and Moon, K.D. "Design of a Universal Middleware Bridge for Device Interoperability in Heterogeneous Home Network Middleware". CCNC Conference. (2005).
  - F. Jammes, H. Smit, "Service-Oriented Paradigms in Industrial Automation", IEEE Transactions on Industrial Informatics, Vol.1 (1), pp. 62-70, February 2005.

- Per la realizzazione del prototipo
  - Mahmoud Q.H., Registration and Discovery of Web Services Using JAXR (2002)
  - Topley K., Java Web Services in a nutshell (O'Reilly, 2003)
  - Java Web Services Tutorial (Sun Microsystems, 2004)
  - McLaughlin B., Java & XML 2nd Edition: Solutions to Real-World Problems (O'Reilly, 2001)
  - Harold E.R., XML Bible 2nd Edition (John Wiley & Sons, 2001)
  - Harold E.R, Java Network Programming (O'Reilly, 2000)
  - Hyde P., Java Thread Programming: The Authoritative Solution (SAMS, 1999)
  - JNI tutorial (sun url)
  - Horstmann C.S., Cornell G., Core Java 2 Volume II: Advanced Features - capitolo 11 (Prentice Hall, 1999)
  - http://www.netbeans.org/kb/articles/form_getstart40.html
  - http://www.devleap.it
  - http://java.sun.com/developer/technicalArticles/WebServices/jaxrws/
  - http://java.sun.com/webservices/tutorial.html
  - http://java.sun.com/docs/books/tutorial/native1.1
  - http://www.eiba.com/en/software/falcon/samples.html
  - Nichols, B.A. Myers, M. Higgins, J. Hughes, T.K. Harris, R. Rosenfeld and M. Pignol, "Generating Remote Control Interfaces for Complex Appliances", Proceedings of

the 15th annual ACM symposium on user interface software and technology, ACM Press, pp. 161-170, October 2002.

– EIBA Software, http://www.eiba-software.com/

– F. Furfari, C. Soria, V. Pirrelli, O. Signore, R. Bandinelli, "NICHE: Natural Interaction in Computerized Home Environment", Ercim News no. 58, pp. 66-67, July 2004.

– L. Tarrini, V. Miori, "LIGHT: XML-Innovative Generation for Home Networking Technologies", Ercim News no. 62, pp. 48-49, July 2005.