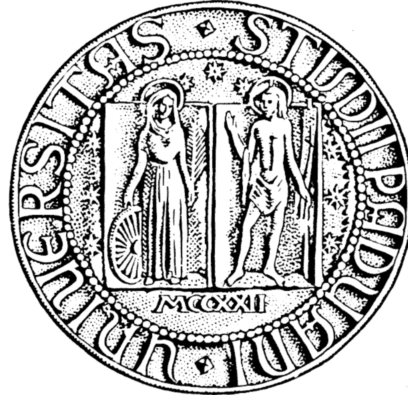


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica

Euclide: sviluppo di un framework per il controllo push-pull di sistemi remoti

Laureando
Alberto Boccato

Relatore
Sergio Congiu

Anno Accademico 2009/2010

SOMMARIO

Euclide è il nome del progetto che Workteam, l'azienda presso cui ho svolto il mio tirocinio, porta avanti da qualche anno. Si tratta di un sistema real time per il monitoraggio di sistemi d'automazione, domotica, sicurezza, antintrusione, antincendio basato interamente su Internet, sia nell'interoperabilità dei servizi monitorati che di quelli rivolti all'utente.

Obiettivo del tirocinio è stato lo sviluppo di un sistema di controllo remoto che si inserisse appunto in Euclide. Il componente doveva permettere di prendere il controllo di un insieme di computer da una postazione remota e, attraverso l'esecuzione di comandi o script, automatizzare una vasta gamma di operazioni. Queste possono essere semplici procedure di configurazione o backup, oppure l'esecuzione di particolari comandi in risposta a segnalazioni o allarmi derivanti da altri componenti di Euclide.

Nella realizzazione del progetto sono stati diversi gli strumenti e i linguaggi di programmazione usati. La piattaforma Java è stata scelta per la realizzazione dei componenti di rete. Il linguaggio XML ha permesso la definizione di un protocollo di comunicazione fra questi e la creazione di file che fungessero da database. Sopra questa struttura è stata poi sviluppata un'interfaccia grafica via browser estremamente dinamica basata sulle tecnologie Servlet e Ajax.

In questa relazione verranno descritti innanzitutto gli strumenti, i linguaggi di programmazione e le tecnologie appena citati analizzandone vantaggi e svantaggi. Successivamente si illustreranno le fasi di analisi che hanno portato alla definizione del progetto vero e proprio, dall'idea di partenza alla progettazione del software. Verranno poi delineate le caratteristiche di ciascun componente, come questi siano stati realizzati e come comunichino tra loro.

RINGRAZIAMENTI

Ringrazio tutta la Workteam ed in particolare il mio tutor aziendale Enrico Conte per avermi accolto e seguito nel corso del tirocinio.

Ringrazio il mio relatore, Sergio Congiu, per essersi dimostrato disponibile, seguendo e correggendo il mio lavoro.

Ringrazio infine i miei compagni di corso, con i quali ho avuto modo di discutere e confrontarmi durante la mia carriera universitaria, e tutti coloro che hanno contribuito alla mia formazione in questi anni.

INDICE

1 INTRODUZIONE 1
- 1.1 L'azienda 1
- 1.2 Il progetto Euclide 1
- 1.3 Obiettivi 2
- 1.4 Campo applicativo 2

2 ANALISI DEL PROGETTO 3
- 2.1 Panoramica 3
- 2.2 Componenti 5
- 2.3 Comunicazione tra componenti 6
- 2.4 Interazione con l'utente 6

3 STRUMENTI E TECNOLOGIE UTILIZZATI 9
- 3.1 Java 9
  - 3.1.1 Apache Xerces2 Java 10
  - 3.1.2 Javamail 10
- 3.2 Eclipse 10
- 3.3 XML 10
  - 3.3.1 Cos'è l'XML 10
  - 3.3.2 La sintassi XML 11
  - 3.3.3 Documenti ben formati e documenti validi 12
  - 3.3.4 Diversi tipi di parser 14
  - 3.3.5 Il modello Document Object Model 15
- 3.4 La tecnologia Servlet 16
  - 3.4.1 Che cos'è una servlet 16
  - 3.4.2 Servlet container 16
- 3.5 Linguaggi lato client 17
  - 3.5.1 Tecnologia AJAX 17
  - 3.5.2 JavaScript e jQuery 19
  - 3.5.3 Il formato JSON 19

4 REALIZZAZIONE 21
- 4.1 Definizione di un protocollo di trasmissione 21
- 4.2 Il server 23
  - 4.2.1 Gestione descrittori e loro struttura 25
  - 4.2.2 Gestione delle connessioni 28
  - 4.2.3 Sicurezza e autenticazione 29
  - 4.2.4 Logging delle operazioni svolte 31
  - 4.2.5 Supporto all'interfaccia grafica 31

4.3 Messaggi scambiati 33
4.4 L'agente 35
4.4.1 Gestione delle connessioni 35
4.4.2 Traduzione delle richieste e loro esecuzione 37
4.5 L'interfaccia grafica 37
4.5.1 La servlet 37
4.5.2 Le pagine web dinamiche 41

5 MANUALE UTENTE 45
5.1 Requisiti di sistema 45
5.2 Modalità di avvio 45
5.3 Esportazione del progetto da Eclipse 46
5.4 Problemi noti 46

6 CONCLUSIONI 47
6.1 Diagramma di Gantt delle fasi del tirocinio 47
6.2 Considerazioni sul software sviluppato 47
6.3 Considerazioni personali sull'esperienza 48

Bibliografia 51

ELENCO DELLE FIGURE

Figura 1 situazione fisica 4
Figura 2 architettura push-pull 5
Figura 3 interazione utente 7
Figura 4 parsing DOM 14
Figura 5 Esempio struttura DOM 15
Figura 6 Struttura servlet container 17
Figura 7 Modello sincrono e asincrono 18
Figura 8 Struttura oggetto JSON 20
Figura 9 Struttura array JSON 20
Figura 10 protocollo di trasmissione 23
Figura 11 moduli di cui si compone il server 25
Figura 12 diagramma delle attività del server 29
Figura 13 SSH Tunneling 30
Figura 14 Componenti interfaccia grafica e loro comunicazione 32
Figura 15 Diagramma delle attività dell'agente 36
Figura 16 Esempio della pagina manageAgent.jsp . . . 41
Figura 17 Esempio form inserimento 42
Figura 18 Esempio visualizzazione log 42
Figura 19 Diagramma di Gantt delle fasi del tirocinio . 47

1 INTRODUZIONE

Indice

1.1 L'azienda 1
1.2 Il progetto Euclide 1
1.3 Obiettivi 2
1.4 Campo applicativo 2

1.1 L'AZIENDA

La Workteam di San Donà di Piave è un'azienda di Internet e Business Integrator. Offre connettività e integrazione di sistemi, sicurezza contro intrusioni, tutela dati sensibili, software gestionale, procedure per la gestione dei progetti e processi e soluzioni Internet avanzate. Fra le altre cose si occupa dello sviluppo di portali web e portali tecnologici. In questo contesto nasce il progetto Euclide.

1.2 IL PROGETTO EUCLIDE

Euclide come si è detto è un sistema real time per il monitoraggio di sistemi d'automazione, domotica, sicurezza, antintrusione, antincendio, e controllo video basato interamente su Internet, sia nell'interoperabilità dei servizi monitorati che di quelli rivolti all'utente.

Proprio per la sua natura fortemente aperta agli standard internazionali, "Euclide" permette a sistemi di origine diversa di interfacciarsi e di riversare i propri dati in archivi dalla enorme disponibilità, presentandosi con interfacce e servizi sul mondo Internet.

Il sistema "Euclide" nasce su midleware IBM Websphere e database Db2 sinomimi di alta affidabilità ed efficienza. Euclide possiede una bacheca per la gestione delle informazioni fra utenti diversi, segnalazioni con gestione dello storico e assegnazione delle risorse, coadiuvato da un workflow basato su ruoli e profili. Possiede un sistema di allarmi automatici e/o manuali con

notifiche tramite e-mail o SMS. Inoltre è basato su tecnologia Java per questo altamente scalabile ed interfacciabile con sistemi periferici eterogenei.

1.3 OBIETTIVI

L'idea di realizzare un sistema di controllo remoto (d'ora in poi per comodità verrà chiamato SCR) nasce inizialmente come progetto a se stante. Lo scopo è quello di poter eseguire una serie di script o comandi sulla shell del computer remoto che si desidera controllare monitorando lo stato della loro esecuzione attraverso la verifica dei risultati ritornati. Sulla base di questo framework si vuole poi sviluppare un'interfaccia grafica che permetta all'utente una semplice gestione del sistema.

Successivamente SCR è stato integrato come componente del progetto Euclide. Quest'ultimo, come si è detto, possiede un sistema di segnalazioni e allarmi automatici in risposta a determinati eventi. Ciò che ancora non era implementato all'inizio del tirocinio, era un modo per intraprendere una qualche operazione in funzione di queste notifiche. Con lo sviluppo di un framework per la gestione di sistemi remoti anche questo tassello è stato aggiunto al puzzle ed in seguito si vedrà il motivo di tale affermazione.

1.4 CAMPO APPLICATIVO

Oggigiorno sono moltissime le aziende, anche di piccole dimensioni, che affidano la gestione dei propri dati e procedure ad un sistema informatico. A dover seguire tale sistema è spesso una ditta esterna a cui vengono richieste manutenzioni, aggiornamenti, backup, nuove configurazioni ecc.. È questo il caso di Workteam, ed è proprio in questo contesto che il framework per il controllo di sistemi remoti trova la sua principale utilità. Le operazioni citate precedentemente possono così essere svolte direttamente dall'ufficio. Si immagini ad esempio di installare una rete aziendale e dopo qualche giorno ricevere la richiesta di modificare una particolare impostazione in una delle macchine. Attraverso questo software è possibile farlo senza doversi muovere dalla propria scrivania con il conseguente risparmio in termini di tempo e conseguentemente dei costi.

2 ANALISI DEL PROGETTO

Indice

2.1 Panoramica 3
2.2 Componenti 5
2.3 Comunicazione tra componenti 6
2.4 Interazione con l'utente 6

2.1 PANORAMICA

Dopo una serie di incontri di braistorming ed analisi delle esigenze è emerso come il sistema avrebbe dovuto essere strutturato. La situazione di partenza, dal punto di vista fisico, è quella descritta in figura 1.

La maggior parte delle aziende clienti di Workteam dispone di una propria rete interna dotata di macchine connesse ad Internet e protette da firewall. L'idea è quella di sviluppare un software che permetta a Workteam di controllare tali postazioni semplicemente depositando i comandi da eseguire su un proprio server. Saranno successivamente le macchine remote a connettersi al server e richiedere la lista dei comandi da lanciare. In questo modo tutte quelle operazioni che non richiedono un'interazione real-time possono essere automatizzate. La fase di login non sarà necessaria e l'utente si troverà ad un livello di astrazione più alto rispetto ai classici sistemi di controllo remoto, infatti non dovrà preoccuparsi di alcun dettaglio relativo alla connessione ma semplicemente definire quali siano le operazioni che intende eseguire.

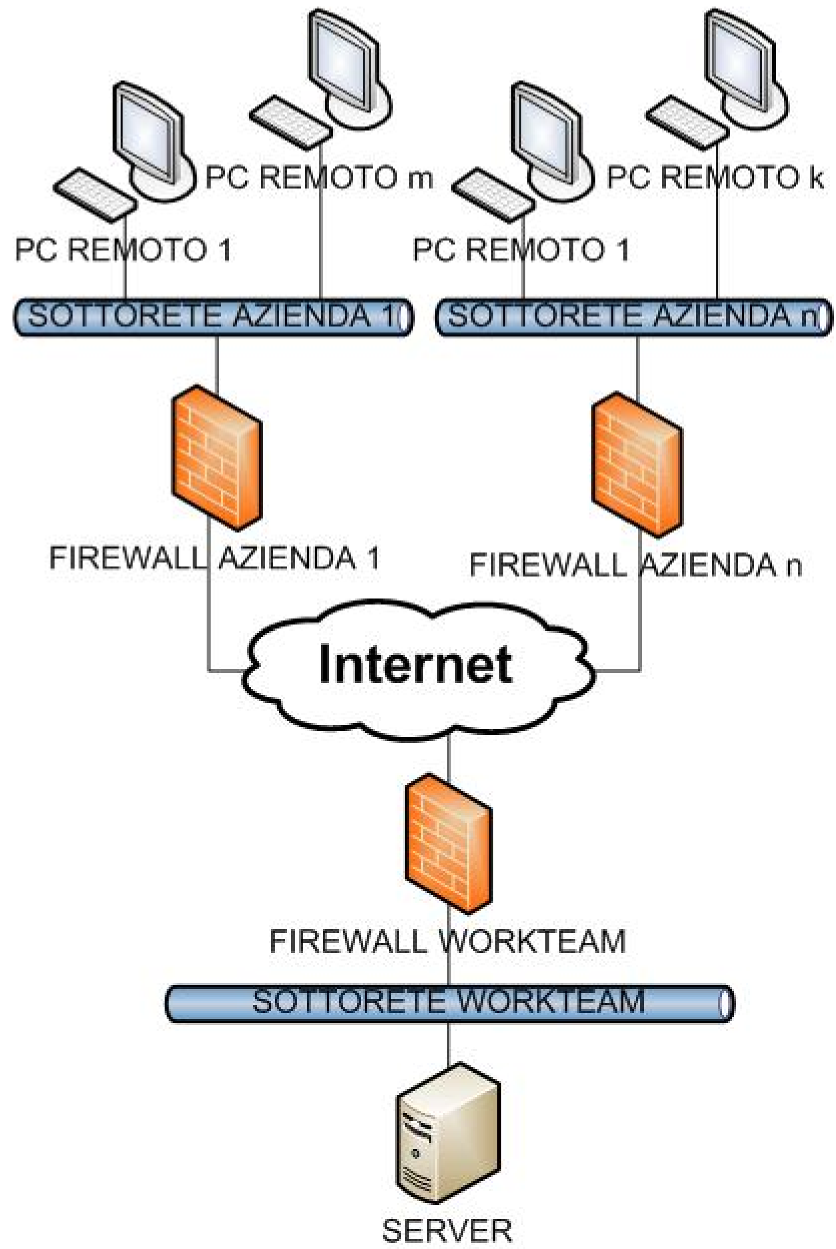


Figura 1: situazione fisica

L'architettura push-pull

Da queste considerazioni deriva il termine push-pull intendendo con esso un sistema dove l'utente deposita le proprie richieste in una sorta di contenitore presente sul server aziendale, da cui ciascuna postazione remota andrà poi a recuperarle.

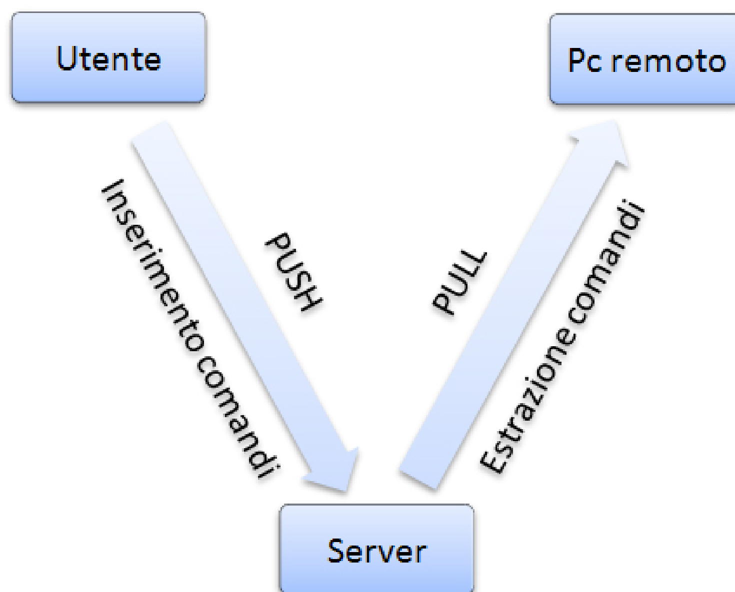


Figura 2: architettura push-pull

Vantaggi

L'architettura push-pull offre inoltre il vantaggio di non sovraccaricare l'unità centrale, la quale non deve preoccuparsi di distribuire il lavoro in quanto sono le macchine client, nel momento che ritengono più opportuno, a verificare la presenza di qualche operazione da svolgere. Strutture di questo tipo infatti, vengono spesso usate in sistemi software di grosse dimensione, ad esempio IBM Tivoli.

2.2 COMPONENTI

A questo punto è possibile definire le componenti che, una volta messe in comunicazione, costituiranno il nostro framework.

L'agente

Nelle macchine client da controllare sarà installato un agente (agent), ossia un software sempre attivo in background, incaricato di connettersi periodicamente al server, verificare la presenza di qualche richiesta, eseguirla e ritornare i risultati. Poiché ci si prefigge l'obiettivo di controllare diverse tipologie di sistemi

è necessario che questo componente sia estremamente flessibile ed adattabile alle varie architetture.

Il server

L'applicazione collocata sul server, invece, si occuperà di attendere connessioni da parte degli agenti, verificarne l'identità, inviar loro le richieste di esecuzione di comandi, script, codice eseguibile, ecc. se presenti, e ricevere i risultati. Il server dovrà dunque possedere una lista degli agenti conosciuti, che chiameremo descrittore degli agenti. In questo modo sarà possibile identificarli e memorizzare le richieste pendenti per ciascuno di essi. Oltre a questo, sarà necessario un descrittore dei comandi per tener traccia di tutte le possibili operazioni da eseguire. Più avanti si vedrà come questi descrittori sono stati realizzati.

L'interfaccia grafica

Infine per permettere all'utente una semplice ed efficace gestione del sistema verrà implementata un'interfaccia grafica via browser.

2.3 COMUNICAZIONE TRA COMPONENTI

Una volta individuati gli elementi costitutivi del sistema è necessario definire il modo in cui questi comunicheranno. La trasmissione dei dati avverrà su protocollo TCP/IP sfruttando la diffusa tecnologia dei socket. Su questa base sarà necessario studiare poi un particolare protocollo di comunicazione che specifichi ai vari componenti il modo con cui la connessione viene stabilita e terminata, la tipologia di messaggi scambiati e il loro flusso. Da questo punto di vista si decide da subito di adottare lo standard XML per strutturare i messaggi con i conseguenti vantaggi in termini di flessibilità, compatibilità con i diversi linguaggi di programmazione e portabilità dato che i sistemi da monitorare sono di diverso tipo.

2.4 INTERAZIONE CON L'UTENTE

Come si è detto il progetto è finalizzato ad un uso interno di Workteam di conseguenza i reali utilizzatori saranno dei tecnici esperti. Partendo da questo presupposto si permetterà all'utente di agire su due livelli:

- Attraverso un'interfaccia grafica che mascheri i dettagli implementativi del sistema e semplifichi la sua gestione;

- Direttamente sulle impostazioni del software lato server. Si tratta di una gestione più completa ma allo stesso tempo più rischiosa;

Vantaggi dell'interazione a più livelli

Il principale vantaggio una suddivisione di questo genere è rappresentato dalla conseguente modularità del software. Il framework, infatti, sarà in grado di funzionare anche in assenza dell'interfaccia grafica permettendo anche una gestione "a basso livello".

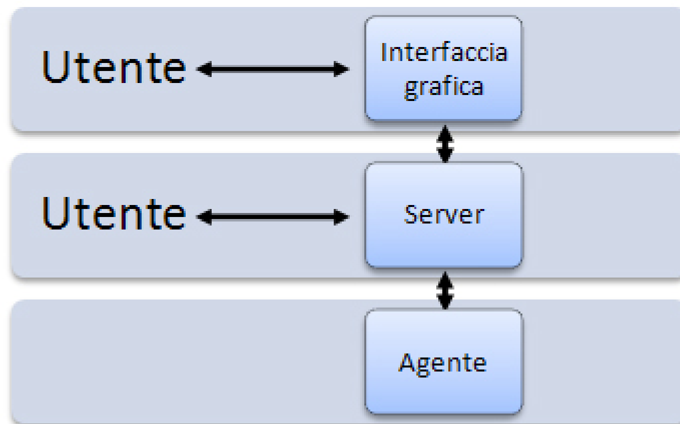


Figura 3: interazione utente

Per quanto riguarda l'interfaccia grafica, l'idea iniziale era di svilupparne una di tipo stand-alone che risiedesse sulla macchina dove sarebbe stato installato il server. Nel corso del lavoro però, si è osservato che un'interfaccia via browser avrebbe garantito una maggiore flessibilità permettendo di spostare questa componente su un computer diverso e di accedervi da qualsiasi posizione attraverso internet. Inoltre con la scelta di questa seconda soluzione sarebbe stato possibile integrare facilmente il sistema di controllo remoto con Euclide, essendo anch'esso fortemente orientato al web.

3 STRUMENTI E TECNOLOGIE UTILIZZATI

In questo capitolo vengono illustrati gli strumenti, i linguaggi e le tecnologie di cui si è fatto uso nel corso del tirocinio, in modo che anche il lettore meno esperto in materia, possa comprendere cosa si è effettivamente realizzato.

Indice

3.1 Java . . . 9
3.1.1 Apache Xerces2 Java . . . 10
3.1.2 Javamail . . . 10
3.2 Eclipse . . . 10
3.3 XML . . . 10
3.3.1 Cos'è l'XML . . . 10
3.3.2 La sintassi XML . . . 11
3.3.3 Documenti ben formati e documenti validi . . . 12
3.3.4 Diversi tipi di parser . . . 14
3.3.5 Il modello Document Object Model . . 15
3.4 La tecnologia Servlet . . . 16
3.4.1 Che cos'è una servlet . . . 16
3.4.2 Servlet container . . . 16
3.5 Linguaggi lato client . . . 17
3.5.1 Tecnologia AJAX . . . 17
3.5.2 JavaScript e jQuery . . . 19
3.5.3 Il formato JSON . . . 19

3.1 JAVA

Java è la piattaforma di sviluppo che si è scelto di usare per la realizzazione di gran parte del progetto. Questa scelta nasce dal fatto che, come si è detto nel capitolo 2, il sistema deve essere il più flessibile possibile ed adattabile a diverse architetture. Oltre a ciò, Java fornisce numerose librerie utili ai nostri scopi: da quelle per la manipolazione di file XML a quelle per l'invio di messaggi di posta elettronica.

3.1.1 Apache Xerces2 Java

http://xerces.apache.org/

Apache Xerces2 Java è un processore per il parsing, la validazione e la manipolazione di file XML. In altre parole un software che permette di accedere a file XML in modo molto più comodo rispetto ad una lettura "manuale" con Java. Le librerie necessarie per il suo funzionamento sono già incluse in JRE 6.

3.1.2 Javamail

Si tratta di un'insieme di API che forniscono un framework multipiattaforma per la creazione di e-mail e applicazioni di messaggistica. Attraverso l'uso di questa libreria è possibile creare software Java in grado di spedire messaggi di posta elettronica secondo diversi protocolli. Javamail non è inclusa nella piattaforma Java SE mentre è già presente all'interno della Java EE.

3.2 ECLIPSE

Eclipse è l'IDE utilizzato per l'implementazione dell'intero software. Si tratta di un ambiente di sviluppo integrato multilinguaggio e multipiattaforma dotato fra l'altro di strumenti di evidenziazione, compilazione e debug integrati. Eclipse non è un editor: non si può, ad esempio, aprire e compilare un qualunque file. Per usare questo ambiente di sviluppo bisogna organizzare il proprio lavoro in progetti. Un progetto è l'insieme di file e impostazioni relativi ad un programma che si sta sviluppando. Nel caso di Java, ad esempio, il progetto conterrà non solo i file sorgenti .java ma anche l'indicazione di quale JDK usare, le librerie esterne (file .jar) e altro ancora.

3.3 XML

3.3.1 Cos'è l'XML

XML è l'acronimo di eXtensible Markup Language. Si tratta di un metalinguaggio che permette la definizione di altri linguaggi che seguono regole precise e rigorose per la struttura, la sintassi e la semantica, come stabilito dal W3C.

XML consente la memorizzazione dei dati in forma di testo semplice: qualsiasi applicazione (o qualsiasi essere umano) che possa leggere un file di testo può leggere un documento XML. Non è necessario quindi avere un programma specifico per accedere ai dati, è sufficiente un semplice editor di testo. Questo aspetto può sembrare una banalità ma in realtà non lo è: i formati proprietari dei dati sono diventati così complessi che di frequente la nuova versione di un'applicazione non può essere letta da una versione precedente della stessa applicazione. XML offre dunque un metodo per inserire dati strutturati in un file di testo in modo che questi siano:

- Privi di ambiguità
- Estendibili
- Indipendenti dalla piattaforma

Estendibilità significa che il linguaggio può essere ampliato per soddisfare esigenze specifiche. XML non è infatti basato su un insieme finito di marcatori, e si possono creare marcatori descrittivi adatti alle proprie necessità.

3.3.2 La sintassi XML

Vediamo ora un esempio di documento XML per comprendere meglio come questo possa organizzare delle informazioni:

rubrica.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rubrica xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="schemaRubrica.xsd">
    <persona>
        <nome>Luca</nome>
        <cognome>Rossi</cognome>
    </persona>
    <persona>
        <nome>Marco</nome>
        <cognome>Bianchi</cognome>
    </persona>
</rubrica>
```

L'XML, come l'HTML, utilizza dei marcatori chiamati tag per assegnare una semantica al testo. I tag presenti in questo documento sono: rubrica, persona, nome e cognome. Come si può

vedere questi elementi hanno nomi semplici e descrittivi: XML è leggibile da una macchina, ma è comprensibile anche dagli esseri umani. Basta rispettare alcune regole per i nomi di questi tag:

- Il nome di un elemento in XML deve iniziare con una lettera, un segno di sottolineatura (_) o un segno di due punti (:);
- Dopo il primo carattere, il nome di un elemento può contenere lettere, cifre, trattini (-), segni di sottolineatura (_), punti (.) o due punti (:);
- I nomi degli elementi non possono iniziare con XML o le sue varianti perché questi nomi sono tutti proprietà intellettuale del W3C;

Osserviamo inoltre come la sintassi della prima riga sia leggermente diversa da quella degli elementi. Si tratta di una dichiarazione XML ossia un'istruzione di elaborazione che fornisce all'analizzatore sintattico informazioni particolari come la versione e codifica del documento.

3.3.3 Documenti ben formati e documenti validi

Documenti ben formati

Requisito indispensabile di ogni documento XML è il suo essere ben formato, ovvero che la sua forma testuale rispetti l'apertura e la chiusura dei tag e il loro annidamento. Questo è indispensabile affinché i processori XML (parser), possano interpretare correttamente il documento e metterlo a disposizione delle applicazioni.

Documenti validi

La strutturazione dei dati in un file di testo comporta però molto più del rispetto della sola sintassi. Un documento, per esempio potrebbe essere perfetto sintatticamente, ma contenere elementi disposti in ordine errato, oppure escludere elementi indispensabili. Per questo motivo è possibile associare ad un documento XML un secondo documento, chiamato XML Schema, che ne definisca la struttura. La definizione formale dello schema a cui deve sottostare il documento XML rappresenta una fonte di riferimento precisa per il documento stesso, ma anche leggibile e comprensibile dagli esseri umani.

Riprendendo l'esempio precedente vediamo quale sia lo schema da associare al documento affinché un processore XML possa eseguirne la validazione.

schemaRubrica.xsd

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="rubrica">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="persona" type="voceRubrica"
                     minOccurs="1" maxOccurs="unbounded"/>
          </xsd:sequence>
            </xsd:complexType>
          </xsd:element>

  <xsd:complexType name="voceRubrica">
    <xsd:sequence>
      <xsd:element name="nome" type="xsd:string"
                   minOccurs="1" maxOccurs="1"/>
      <xsd:element name="cognome" type="xsd:string"
                   minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

Questo nuovo documento XML, come si può vedere, fa uso di particolari tag, che permettono di definirne la struttura. Cerchiamo di capirne il significato.

L'elemento *<schema>* è la radice di ogni XML schema e può contenere diversi attributi. In questo caso il frammento

```
xmlns:xsd=http://www.w3.org/2001/XMLSchema
```

sta ad indicare che gli elementi e i tipi di dato che verranno usati nello schema derivano dal namespace *http://www.w3.org/2001/XMLSchema* e dovranno usare il prefisso *xsd* per essere richiamati. Un namespace è semplicemente un contenitore di nomi definito da un unico identificatore, in modo da ovviare eventuali ambiguità nel caso ci fossero documenti diversi aventi tag omonimi.

La comprensione del resto del documento è abbastanza intuitiva: definisce un tag *<rubrica>* all'interno del quale possono essere presenti uno o più elementi *<persona>* di tipo *"voceRubrica"*. Il tipo *"voceRubrica"* si compone poi di una sequenza di due tag, chiamati *<nome>* e *<cognome>*, che devono contenere una stringa ed essere sempre presenti rispettando il loro ordine.

A questo punto è necessario modificare il file *rubrica.xml* in modo da associarlo allo schema definito. Il tag <*rubrica*> deve dunque essere sostituito con il seguente:

```
<rubrica xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="schemaRubrica.xsd">
```

3.3.4 Diversi tipi di parser

Dunque chi si occupa dell'analisi sintattica e della validazione sono i parser. Esistono fondamentalmente due tipi di parser per l'XML:

- SAX (Simple API for XML)
- DOM (Document Object Model)

Parser SAX

I primi sono basati sulle "sequenze di eventi": scorrono sequenzialmente i file XML senza tener traccia di tutta la sua struttura ma solamente degli eventi di interesse per il programmatore eseguendo solamente le azioni ad essi relative.

Parser DOM

I secondi invece, leggono i documenti come se fossero delle strutture gerarchiche ad albero. Con DOM un file viene interamente caricato nella memoria, dopo di che sarà possibile effettuare le operazioni volute.

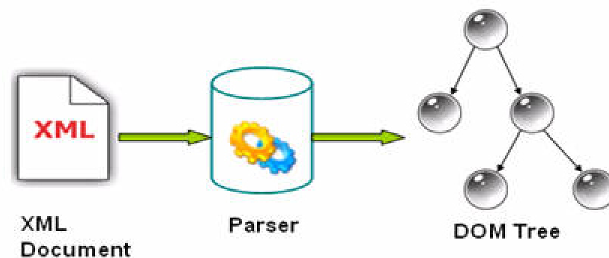


Figura 4: parsing DOM

Vantaggi del DOM

Da queste considerazioni si evince come un parser SAX sia adatto in caso di documenti XML di grandi dimensioni e non soggetti a modifiche. Un parser DOM invece, memorizzando la totalità del documento, ne permette la manipolazione e l'accesso casuale senza doverlo rileggere ogni volta. È questo il principale motivo per cui si è scelto di usare un parser DOM per interpretare il contenuto dei descrittori.

3.3.5 Il modello Document Object Model

Il risultato dell'interpretazione di un documento XML attraverso un parser DOM è, come si è detto, una rappresentazione ad albero del documento stesso. Essa è compatibile con i più noti linguaggi di programmazione ed incapsula ogni elemento caratteristico di XML (elementi, attributi, commenti, . . .) in un oggetto specifico, che ne fornisce un'interfaccia di manipolazione. Esistono varie versioni del DOM, strutturate in livelli:

- Livello 1: definisce gli elementi DOM di base con interfacce contenenti i metodi e gli attributi di uso più comune;
- Livello 2: modifica alcuni metodi del livello 1, e introduce il supporto ai namespaces e alla clonazione dei nodi
- Livello 3: introduce nuovi metodi e interfacce per una navigazione più rapida nel documento, per il supporto dei tipi di nodo, per la serializzazione e per la formattazione del documento

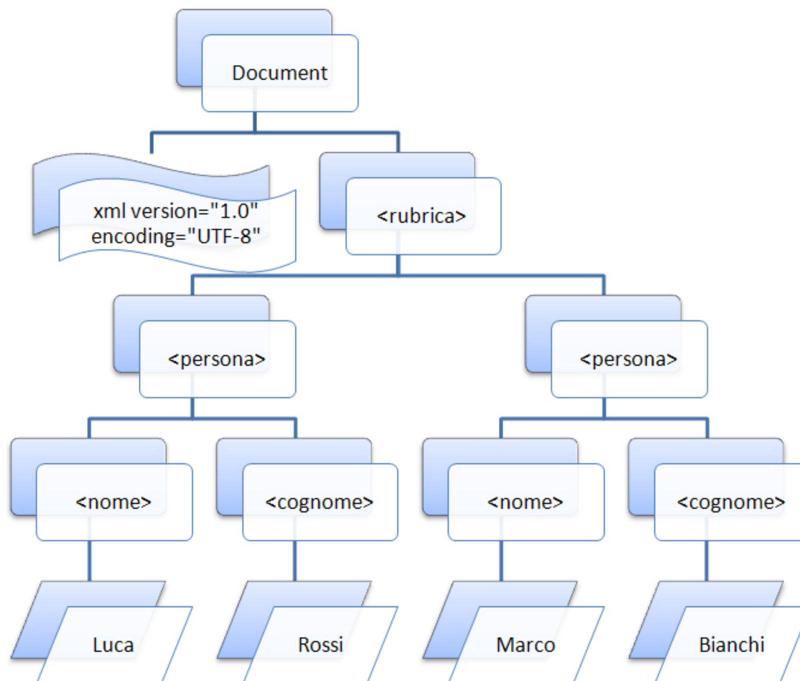


Figura 5: Esempio struttura DOM

3.4 LA TECNOLOGIA SERVLET

3.4.1 Che cos'è una servlet

Una Java servlet è una componente web, fondamentalmente una classe Java, che può essere caricata dinamicamente al fine di espandere le funzionalità di un server. Le servlet sono comunemente usate con i web server, dove possono prendere il posto degli script CGI nella generazione di pagine web dinamiche ed interagire con i client (tipicamente i browser degli utenti) in accordo ad un modello request/response.

Caratteristiche

Le servlet vengono eseguite da una Java Virtual Machine (JVM) sul server, all'interno di un servlet container. Sono sicure e portabili: sia tra i vari sistemi operativi, sia tra i vari web server. Ogni servlet viene mappata in un'URL e gestisce ciascuna richiesta con un thread separato.

3.4.2 Servlet container

Per costruire un'applicazione Web basata sull'uso di Servlet è necessario, come si è detto, un Web Container (o Servlet Container o Servlet Engine). Si tratta del componente di un web server che interagisce con le servlet: è responsabile della gestione del loro ciclo di vita, della loro mappatura nelle varie URL e supporta il protocollo HTTP per gestire il flusso request-response.

Le Request rappresentano la chiamata al server effettuata dal client. Le Request sono caratterizzate da varie informazioni:

- Chi ha effettuato la Request
- Quali parametri sono stati passati nella Request
- Quali header sono stati passati

Le Response rappresentano le informazioni restituite al client in risposta ad una Request:

- Dati in forma testuale (es. html, text) o binaria (es. immagini)
- HTTP headers, cookies, ...

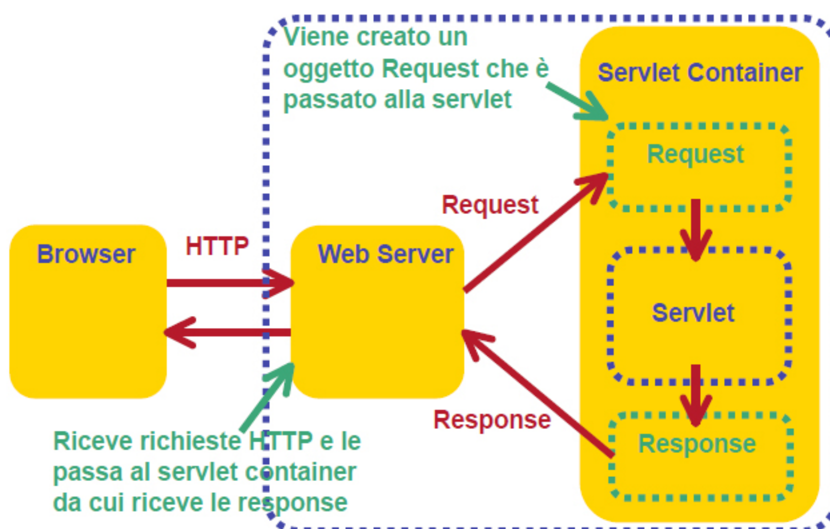


Figura 6: Struttura servlet container

Compatibilità fra servlet container diversi

Nel corso del tirocinio ho avuto modo di cimentarmi con web server e servlet container diversi, in particolare Apache Tomcat e IBM WebSphere. Bisogna dire che l'affermazione fatta in precedenza riguardo alla portabilità delle servlet è stata verificata sul campo in quanto la servlet progettata, di cui si parlerà nel capitolo 4, è stata sviluppata all'interno di Tomcat ed è stata poi trasferita senza alcun problema su un diverso sistema operativo in cui era installato WebSphere Application Server.

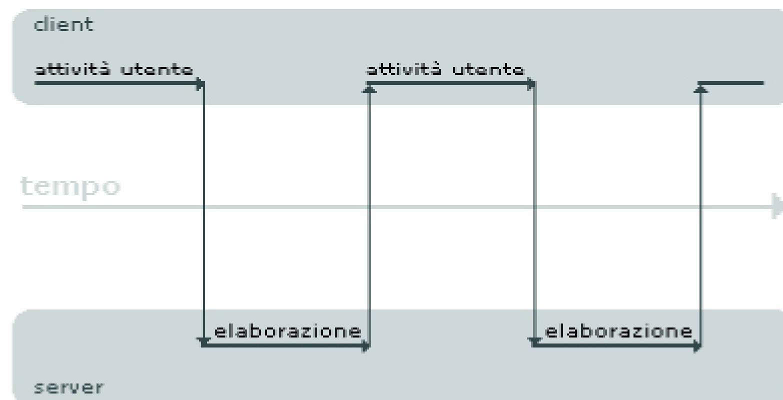
3.5 LINGUAGGI LATO CLIENT

3.5.1 Tecnologia AJAX

AJAX, acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo per la realizzazione di applicazioni web interattive (Rich Internet Application). Lo sviluppo di applicazioni con AJAX si basa su uno scambio di dati in background fra web browser e server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente. Tuttavia, e a dispetto del nome, l'uso di JavaScript e di XML non è obbligatorio, come non è necessario che le richieste di caricamento debbano essere necessariamente asincrone.

Modello sincrono e asincrono

La differenza tra una richiesta sincrona e una asincrona sta nel fatto che nel primo caso l'esecuzione del codice viene interrotta bloccando il browser temporaneamente fino al completamento della richiesta mentre nel caso asincrono non occorre attendere che la richiesta sia ultimata per effettuare altre operazioni, stravolgendo sotto diversi punti di vista il flusso tipico di una applicazione web come si può vedere in figura.



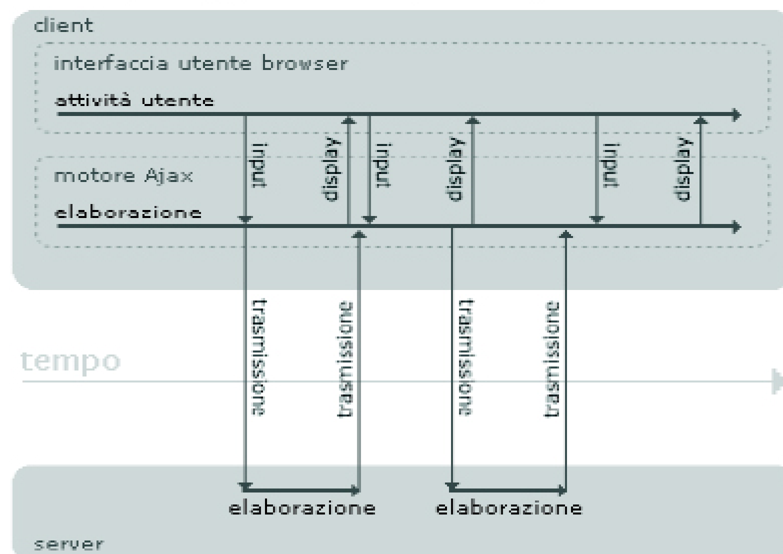


Figura 7: Modello sincrono e asincrono

Nel modello classico ad ogni chiamata del client si ottiene una risposta dal server che si traduce in una pagina web aggiornata.

Nel modello asincrono invece è possibile scambiare dati con il server (in formato XML, JSON o altro) e fare in modo che sia il motore AJAX a decidere quando ricaricare la pagina. Il principale vantaggio di un'architettura di questo tipo consiste in una massiccia diminuzione del traffico tra client e server in quanto non è necessario inviare nuovamente i dati dell'intera pagina. In conseguenza di ciò, è possibile realizzare interfacce utenti molto più veloci.

3.5.2 JavaScript e jQuery

Javascript è un linguaggio di scripting a oggetti pensato per interagire con HTML, estendendo il controllo sul browser, migliorando le prestazioni delle pagine web e rendendole più dinamiche ed efficienti. Con JavaScript è possibile per esempio gestire gli eventi generati dall'interazione fra utente e browser oppure manipolare i moduli compilati verificando la validità di quanto è stato inserito.

Uno dei maggiori problemi di JavaScript sono le sue numerose implementazioni che variano da browser a browser creando problemi di incompatibilità. Per ovviare questo problema si è fatto uso di una libreria JavaScript chiamata jQuery. Essa si propone come obiettivo quello di astrarre ad un livello più alto la programmazione lato client definendo un vasto set di funzioni cross-browser per:

http://www.jquery.com/

- Manipolare il DOM delle pagine web
- Manipolare fogli di stile
- Gestire in modo semplice gli eventi
- Creare effetti grafici
- Gestire la comunicazione tra client e server secondo il modello AJAX

3.5.3 Il formato JSON

JSON (JavaScript Object Notation) è un semplice formato per lo scambio di dati. Per le persone è facile da leggere e scrivere, mentre per le macchine risulta facile da generare e analizzarne la sintassi. Si tratta del formato che è stato usato per ritornare i dati nelle chiamate AJAX in sostituzione all'XML in quanto più leggero e di semplice interpretazione rispetto a quest'ultimo.

http://www.json.org/

JSON è basato su due strutture:

- Un insieme di coppie nome/valore. In diversi linguaggi, questo è realizzato come un oggetto, un record, uno struct, un dizionario, una tabella hash, un elenco di chiavi o un array associativo.
- Un elenco ordinato di valori. Nella maggior parte dei linguaggi questo si realizza con un array, un vettore, un elenco o una sequenza.

Queste sono strutture di dati universali. Virtualmente tutti i linguaggi di programmazione moderni li supportano in entrambe le forme. E' sensato che un formato di dati che è interscambiabile con linguaggi di programmazione debba essere basato su queste strutture. In JSON, assumono queste forme:

- Un oggetto è una serie non ordinata di nomi/valori. Un oggetto inizia con { (parentesi graffa sinistra) e finisce con } (parentesi graffa destra). Ogni nome è seguito da : (due punti) e la coppia di nome/valore sono separata da , (virgola).

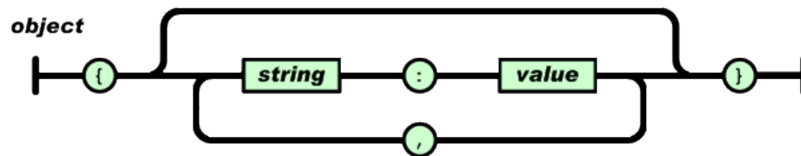


Figura 8: Struttura oggetto JSON

- Un array è una raccolta ordinata di valori. Un array comincia con [(parentesi quadra sinistra) e finisce con] (parentesi quadra destra). I valori sono separati da , (virgola).

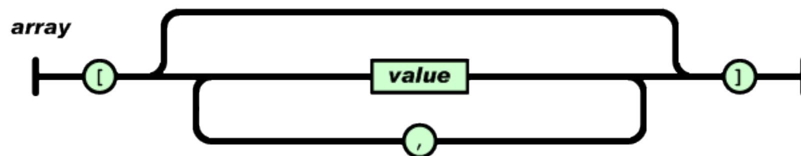


Figura 9: Struttura array JSON

Un valore può essere una stringa tra virgolette, o un numero, o vero o falso o nullo, o un oggetto o un array. Queste strutture possono essere annidate.

4 REALIZZAZIONE

In questo capitolo si descriverà come il framework per la gestione di sistemi remoti è stato implementato effettivamente, sulla base delle considerazioni e dei vincoli raccolti in fase di analisi.

Indice

4.1 Definizione di un protocollo di trasmissione . 21
4.2 Il server 23
4.2.1 Gestione descrittori e loro struttura . 25
4.2.2 Gestione delle connessioni 28
4.2.3 Sicurezza e autenticazione 29
4.2.4 Logging delle operazioni svolte 31
4.2.5 Supporto all'interfaccia grafica 31
4.3 Messaggi scambiati 33
4.4 L'agente 35
4.4.1 Gestione delle connessioni 35
4.4.2 Traduzione delle richieste e loro esecuzione 37
4.5 L'interfaccia grafica 37
4.5.1 La servlet 37
4.5.2 Le pagine web dinamiche 41

4.1 DEFINIZIONE DI UN PROTOCOLLO DI TRASMISSIONE

Prima di analizzare come le varie componenti del software sono state realizzate è necessario spiegare come si è pensato di gestire la loro comunicazione in quanto questo ha un notevole impatto sulla loro struttura. Le regole che permettono ad agent e server di comunicare sono le seguenti:

- L'agente stabilisce una connessione con il server su una determinata porta e invia un messaggio di acknowledge, di cui si vedrà più avanti la struttura;
- Il server, che è costantemente in attesa di qualche messaggio di acknowledge, accetta la connessione e analizzando il contenuto della segnalazione stabilisce l'identità dell'agente;

- Se l'autenticazione va a buon fine è il server ad instaurare una seconda connessione con l'agent su una porta definita nel messaggio di acknowledge;
- L'agent una volta acettata la connessione inizia ad inviare, se ve ne sono, le risposte a comandi precedentemente eseguiti ma che non sono pervenute al server in seguito ad un suo crash o problemi di connessione. Una volta terminate invia una direttiva "stop" che fa terminare questa fase;
- Il server riceve i vecchi risultati finchè incontra la direttiva "stop" dopodiché inizia ad inviare i comandi che si intendono far eseguire all'agent uno alla volta. Dopo ogni invio attende il risultato dell'esecuzione prima di passare alla richiesta successiva. Quando queste sono terminate manda una direttiva "stop" che fa disconnettere l'agent;
- L'agent, specularmente, riceve uno ad uno i comandi ed invia i risultati prodotti dall'esecuzione. Infine quando riceve la direttiva "stop" chiude la comunicazione.

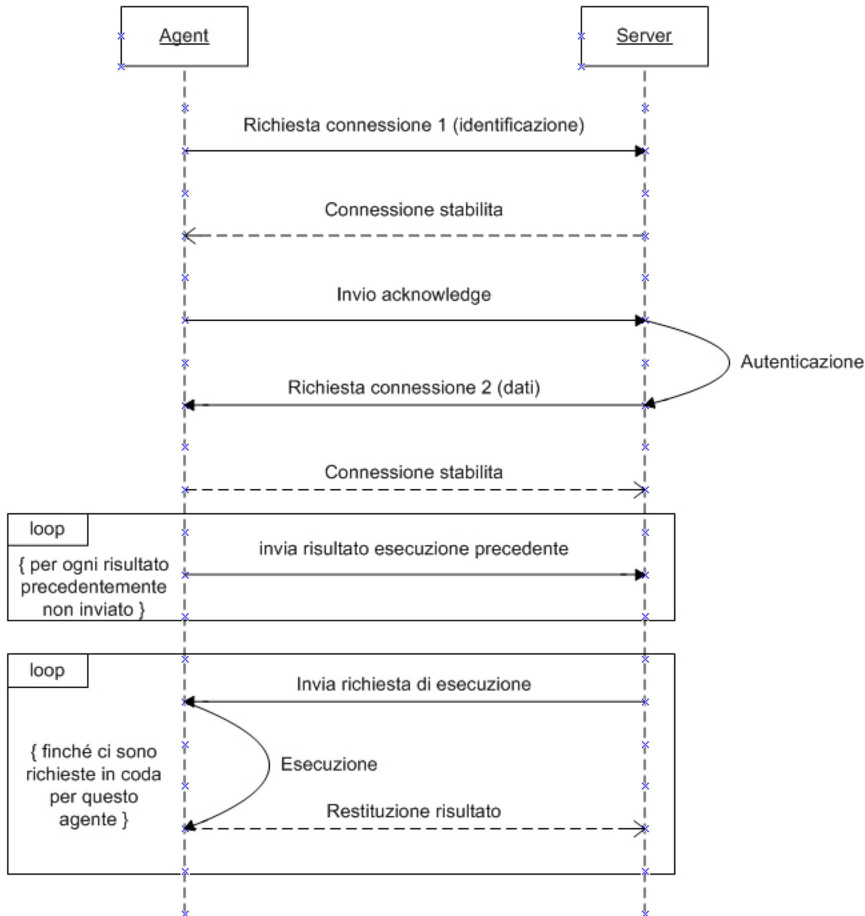


Figura 10: protocollo di trasmissione

Si osservi come la comunicazione avviene su due canali TCP distinti in analogia al protocollo FTP standard. Nel nostro caso la prima connessione rimane in vita solamente il tempo necessario all'invio di un segnale di acknowledge da parte dell'agente. La seconda, invece, è quella su cui viaggiano i dati veri e propri: richieste di esecuzione in un senso e risultati nell'altro.

4.2 IL SERVER

Inizieremo l'esposizione dall'applicazione server, la componente più complessa e significativa che rappresenta il cuore del sistema stesso. Esternamente si presenta come un elemento dotato di due interfacce:

- Una verso gli agenti, con cui deve comunicare scambiando messaggi che, come si è detto, hanno la struttura di un file XML e rispettano un ben preciso protocollo di comunicazione;

- Una verso l'utente, attraverso la quale si possono visualizzare le informazioni relative allo stato degli agenti e caricare i comandi o script che si intendono far eseguire a ciascuno di essi;

Scendendo poi in profondità e analizzando il server dal punto di vista interno si possono individuare i moduli principali di cui è costituito:

- Gestione dei descrittori: si occupa di recuperare ed aggiornare le informazioni relative agli agenti e ai comandi noti. Come si è detto i descrittori fungono da database e sono realizzati con dei file XML;

- Gestione delle connessioni: definisce il protocollo di trasmissione tra server ed agent, ossia l'insieme di regole che permettono il corretto scambio dei messaggi. Specifica la struttura della comunicazione, come viene stabilita e rilasciata, quali sono i processi interessati ed il loro ciclo di vita;

- Sicurezza e autenticazione: permette di determinare l'identità dell'agente che tenta di connettersi, verificarne le relative informazioni nel descrittore e così decidere se stabilire o meno la comunicazione;

- Logging delle operazioni svolte: tiene traccia di tutte le azioni intraprese dal server e delle eventuali segnalazioni di errore. Memorizza inoltre i risultati dei comandi eseguiti dagli agent;

- Supporto all'interfaccia grafica: definisce il processo che si occupa della comunicazione con l'interfaccia utente. Fondamentalmente risponde alle interrogazioni provenienti da quest'ultima e, sfruttando il modulo di gestione dei descrittori ne permette la modifica in funzione delle richieste;

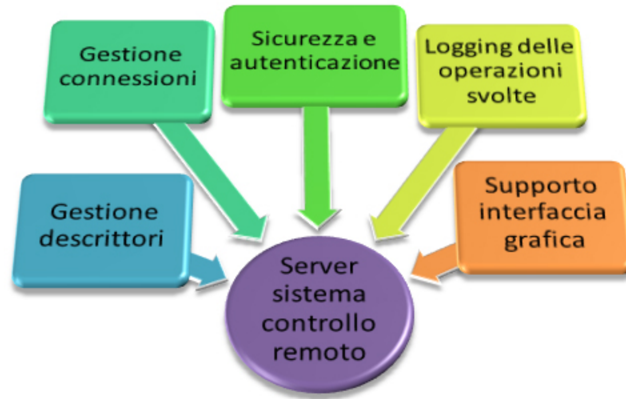


Figura 11: moduli di cui si compone il server

4.2.1 Gestione descrittori e loro struttura

A questo punto è necessario specificare quali sono le informazioni di cui il server necessita per funzionare e in che modo queste vengono memorizzate. Come si è accennato in precedenza esistono due descrittori che fungono da database, uno per gli agenti e uno per i comandi. Il primo tiene traccia di tutte le informazioni relative agli agenti. Per ognuno di questi si ha:

Descrittore degli agenti

- id: il codice che identifica univocamente l'agente all'interno del descrittore;
- un nome o una descrizione che ricordi la macchina su cui è stato installato;
- l'indirizzo ip su cui è stato installato;
- la coda contenente gli id dei comandi di cui l'utente ha richiesto l'esecuzione;

Il secondo memorizza le informazioni relative ai possibili comandi da eseguire sulle macchine remote. Esse sono:

Descrittore dei comandi

- id: il codice che identifica univocamente il comando all'interno del descrittore;
- la tipologia del comando ossia se si tratta di un comando di sessione che interessa il sistema (per esempio la direttiva di disconnessione) oppure un comando runtime da eseguire sulla shell dell'agente;

- l'effettivo comando che si intende eseguire cioè quello che verrà lanciato sulla riga di comando;
- la direttiva da applicare in caso di errori (per esempio ritentare l'esecuzione o disconnettersi);

Poiché si è scelto di usare dei file XML per rappresentare tutti questi dati vediamo attraverso un esempio come tali file sono stati progettati.

AgentDescriptor.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Agents xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="AgentDescriptorSchema.xsd">
    <Agent>
        <Id>WORKTEAM_1</Id>
        <Name>workteam ufficio 1</Name>
        <Ip>192.168.252.161</Ip>
    </Agent>
    <Agent>
        <Id>WORKTEAM_2</Id>
        <Name>workteam ufficio 2</Name>
        <Ip>192.168.252.162</Ip>
        <Command>IPCONFIG</Command>
        <Command>SCRIPT_BACKUP</Command>
    </Agent>
</Agents>
```

CommandDescriptor.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Commands xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="CommandDescriptorSchema.xsd">
    <Entry>
        <Id>IPCONFIG</Id>
        <Type>cmd</Type>
        <Command><![CDATA[ipconfig]]></Command>
        <OnError>retry</OnError>
    </Entry>
    <Entry>
        <Id>SCRIPT_BACKUP</Id>
        <Type>cmd</Type>
        <Command><![CDATA[#!/bin/bash
                SRCD="logs/"
                TGTD="./"
                OF=logs-$(date +%Y%m%d).tgz
                tar -cf $TGTD$OF $SRCD]]>
          </Command>
        <OnError/>
    </Entry>
</Commands>
```

Come si può vedere i tag XML sono auto esplicativi e il risultato è la creazione di una struttura chiara e semplice. L'agente *WORKTEAM_1* non ha alcuna richiesta pendente mentre per *WORKTEAM_2* sono stati depositi in coda due comandi i cui id sono *IPCONFIG* e *SCRIPT_BACKUP*. Affinché il sistema possa poi inviare all'agent tutte le informazioni necessarie, relative a tali comandi, è indispensabile che i loro id compaiano nel descrittore dei comandi come nell'esempio.

Riguardo a questo secondo file si noti come la sigla *"cmd"* del campo <*Type*> stia ad indicare un comando o uno script da lanciare sulla shell dell'agente. Ciò che viene effettivamente eseguito è il contenuto del tag <*Command*>, il quale, poichè al suo interno potrebbero essere presenti caratteri non ammessi dalla sintassi XML, è incapsulato in una sezione CDATA.

Va ricordato inoltre che ad ogni descrittore è associato un XML Schema che ne descrive il contenuto. Si osservi infatti come la seconda riga degli esempi definisca la posizione di tali schemi. In questo modo è possibile validare i file XML al momento del parsing ed eliminare eventuali errori strutturali oltre che sintattici.

Il modulo di gestione dei descrittori deve dunque fornire le primitive per inserire, modificare, cancellare e recuperare le voci dai file XML oltre alle funzionalità di parsing e validazione degli stessi. All'avvio del server i descrittori vengono caricati e vengono salvate in RAM le loro strutture DOM. Da quel momento in poi l'applicazione lavora su questi oggetti aggiornandoli e sincronizzandoli in determinati momenti con il file presente su supporto di massa.

Modifica manuale dei descrittori

Nella sua gestione priva di interfaccia grafica, il sistema prevede che l'utente faccia terminare il server e vada a modificare manualmente questi file. Una volta depositati in coda i comandi che si intende far eseguire, il server viene riavviato e i descrittori ricaricati.

Modifica automatica dei descrittori

Al contrario con l'uso dell'interfaccia grafica questo non è necessario, in quanto è il software che va ad agire direttamente sull'immagine del descrittore presente in RAM, senza la necessità di ricaricare il file.

4.2.2 Gestione delle connessioni

L'applicazione server, come si è visto, è strutturata in diversi moduli. Affinché la loro gestione avvenga in modo efficiente si sceglie di seguire il paradigma della programmazione multithreading. In questo modo è possibile suddividere le funzionalità in singoli thread con un ben preciso compito. È questo il caso, per esempio, del processo di gestione delle connessioni con gli agenti. Vediamo ora come ciò che è stato definito nel protocollo di trasmissione viene implementato. All'avvio del server viene creato un thread chiamato AgentListener che si mette in attesa di qualche messaggio di acknowledge da parte degli agenti. Nel momento in cui riceve una di queste segnalazioni, la cui struttura sarà spiegata più avanti, avvia un ulteriore thread denominato AgentManager e si rimette in ascolto. In questo modo tutte le operazioni di gestione relative ad un agente attivo vengono delegate ad un AgentManager specifico. In questo modo tutte le fasi di autenticazione, ricezione vecchi risultati, invio dei comandi da eseguire, ricezione delle risposte, ecc. possono essere svolte contemporaneamente per agenti diversi. Una organizzazione single-thread sarebbe stata improponibile in quanto per prendere il controllo di una macchina remota si avrebbe dovuto attendere la conclusione delle operazioni di tutte quelle già connesse.

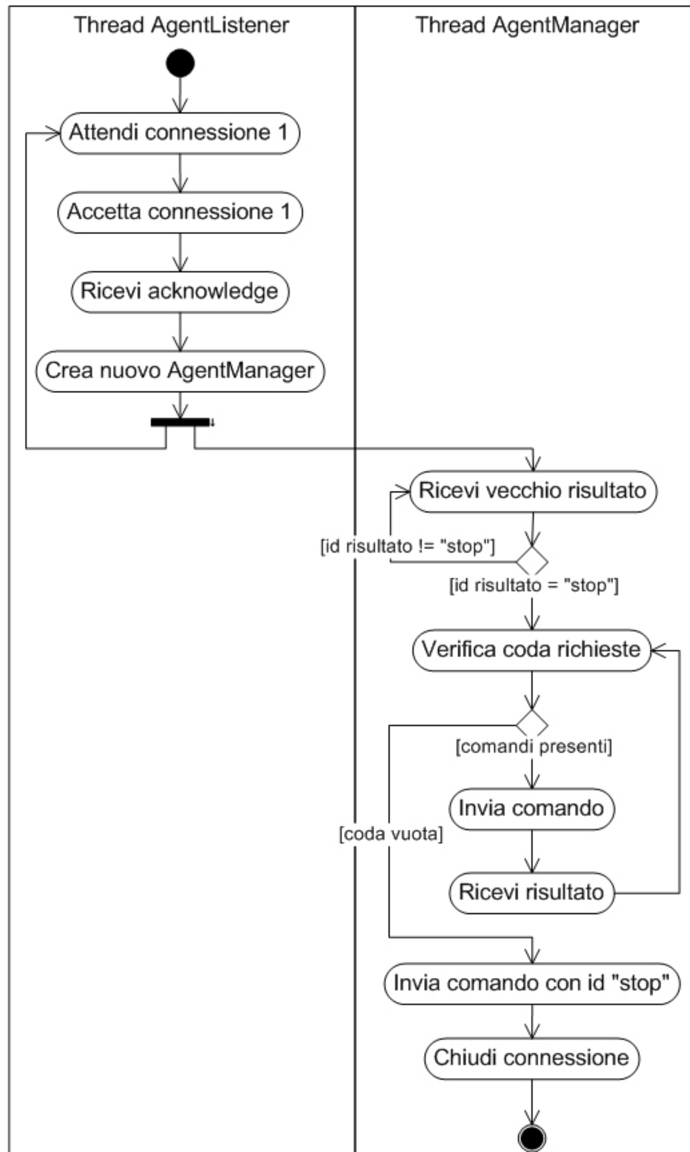


Figura 12: diagramma delle attività del server

4.2.3 Sicurezza e autenticazione

SSH tunneling

Per quanto concerne la sicurezza, il server non prevede di stabilire connessioni criptate fra server ed agent. Questo potrebbe sembrare a prima vista un baco enorme per un software che permette di eseguire dei comandi su una shell remota. In realtà il server può trasmettere tranquillamente in chiaro in quanto si è scelto di demandare il compito di cifratura ad un altro

software. Ssh permette infatti di stabilire un tunnel sicuro in cui far passare tutto il traffico generato dalle componenti del framework.

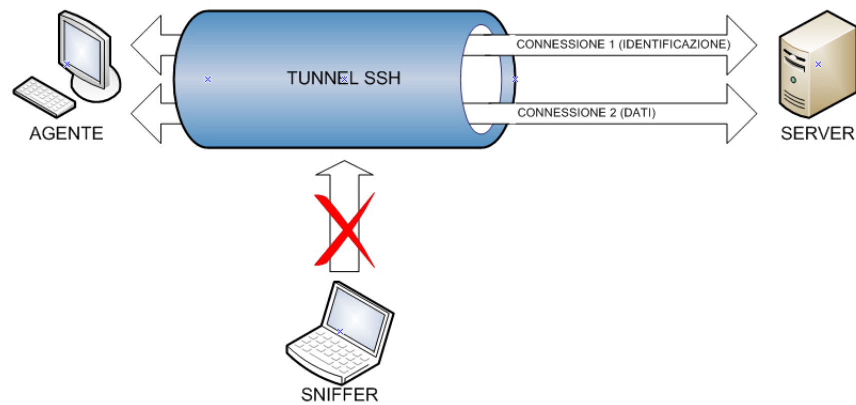


Figura 13: SSH Tunneling

Autenticazione

Esiste invece una fase di autenticazione che permette al server di verificare l'identità dell'agente prima di inviargli i comandi. Come si è detto, appena viene stabilita la prima connessione, l'agente invia un messaggio di acknowledge. Esso contiene l'id dell'agente seguito dal numero di porta su cui questo si aspetta di instaurare la seconda connessione. Il server può così recuperare dal descrittore degli agenti l'ip associato e verificare che corrisponda a quello di provenienza del messaggio. Se così non fosse l'agente viene rifiutato e nessuna richiesta inviata. Allo stesso modo se l'id contenuto nell'acknowledge non ha un riferimento nel descrittore, l'agente viene considerato sconosciuto e di conseguenza la seconda connessione, quella dal server verso l'agent, non viene mai stabilita.

Vantaggi dell'autenticazione

Oltre ai vantaggi in termini di sicurezza questo meccanismo evita all'utente di eseguire dei comandi sul pc remoto sbagliato. Si immagini il caso in cui ad un agente, al momento della configurazione, venga assegnato l'id di un altro agente. Se il server verificasse semplicemente l'id, quell'agente eseguirebbe tutti i comandi indirizzati invece ad un'altra macchina senza che l'utente se ne accorga. Controllando anche l'ip invece, è possibile essere certi che l'agente si trovi esattamente sulla macchina remota che si vuole gestire.

4.2.4 Logging delle operazioni svolte

Un altro aspetto importante per l'applicazione server è la fase di logging delle operazioni svolte. Per operazioni svolte si intendono sia le richieste che sono state eseguite ed hanno prodotto un risultato, sia le varie azioni di routine compiute dal server (ad esempio caricamento descrittori, apertura connessione, ecc.). Per questo motivo esiste un file di log per ogni agente in cui vengono registrate tutte le azioni svolte dal thread AgentManager ad esso associato.

Logging via e-mail

Il sistema di logging prevede inoltre una diversa modalità di gestione dei risultati attraverso l'invio di messaggi di posta elettronica. È infatti possibile configurare il server in modo che, alla ricezione di un risultato di esecuzione, lo spedisca attraverso una e-mail ad un indirizzo interno a Workteam oltre che salvato sul disco locale. Il motivo per cui è stata implementata una simile gestione dei log si riconduce all'integrazione del nostro framework con il progetto Euclide. Il server di posta dove vengono depositate le e-mail è infatti lo stesso in cui sopraggiungono le notifiche e gli allarmi di Euclide e costituisce dunque l'anello di congiunzione tra i due software. I messaggi inviati dovranno perciò seguire lo stesso standard previsto dal sistema già esistente. In questo modo è possibile interpretarli ed elaborarli trattandoli come una nuova tipologia di segnalazione.

4.2.5 Supporto all'interfaccia grafica

Come si è detto il server può essere gestito manualmente agendo direttamente sui descrittori oppure attraverso un'interfaccia grafica via browser che si appoggia ad una servlet. Il vantaggio di una architettura di questo tipo consiste nella possibilità di avviare l'applicazione server su una macchina diversa da quella in cui risiede il web container. Inoltre attraverso un browser è possibile gestire il sistema di controllo remoto da qualsiasi posizione.

Comunicazione tra server e servlet

Questa scelta implica la comunicazione tra le due parti (server e servlet) che, anche in questo caso, avviene attraverso l'uso delle socket TCP/IP. All'avvio del server, infatti, viene creato un thread chiamato ServletListener che si occupa esclusivamente di rispondere alle richieste provenienti dalla servlet. L'utente, agendo sulle pagine web, comunica alla servlet le operazioni che intende svolgere e queste vengono poi inoltrate al server aprendo

un canale TCP. In questo caso i messaggi scambiati consistono nei tipi di dato elementari (stringhe, valori booleani, ecc.) e non sono incapsulati in file XML in quanto tale scelta avrebbe appesantito la comunicazione.

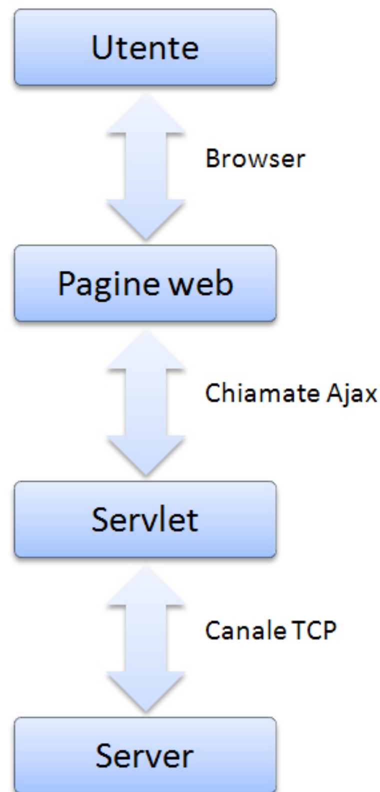


Figura 14: Componenti interfaccia grafica e loro comunicazione

Il thread ServletListener come si è detto si occupa di rispondere alle richieste provenienti dalla servlet. Per assolvere a tale compito è necessaria, nella maggior parte dei casi, la modifica dei due descrittori essendo questo l'unico modo per definire nuovi comandi e caricarli nella coda di un agente. Il ServletListener si appoggia dunque al modulo di gestione dei descrittori sfruttando le primitive che questo mette a disposizione. Se, per esempio, l'utente chiede di aggiungere un certo comando in coda ad un agente, è necessario innanzitutto che la servlet invii al server un messaggio. Questo, una volta tradotto, andrà a modificare il descrittore aggiungendo il tag relativo nella coda dell'agente.

4.3 MESSAGGI SCAMBIATI

Prima di analizzare come è stato realizzato l'agent è necessario definire la struttura dei messaggi che questo scambia con il server. Esistono tre tipologie di messaggio:

1. acknowledge
2. richiesta esecuzione
3. risultato esecuzione

Il primo, come si è detto, viaggia su quella che in precedenza è stata chiamata connessione 1 o connessione identificazione. Si tratta di una brevissima stringa che contiene: l'id dell'agente che invia il segnale e la porta su cui esso si aspetta di ricevere la successiva connessione da parte del server, separati dal carattere ":". Esempio:

WORKTEAM_1:9001

Gli altri due appartengono invece alla connessione 2 ossia ai dati veri e propri. In particolare le richieste di esecuzione viaggiano dal server verso gli agenti, mentre i risultati dagli agenti verso il server. Per la creazione di questi messaggi si è scelto di usare dei file XML, di fatto lo standard per lo scambio dei dati in applicazioni web e non. Vediamo ora nel dettaglio come sono stati strutturati.

Messaggio 2: richiesta esecuzione

```
<Request>
        <Id>IPCONFIG__20-08-2010__14-37-00<Id>
        <Type>cmd</Type>
        <Command><![CDATA[ipconfig]]></Command>
        <OnError>retry</OnError>
</Request>
```

Di fatto ciò che viene mandato all'agent è la voce, estratta dal descrittore dei comandi, relativa al comando che si intende eseguire. A questa viene appesa la data e l'ora al momento dell'invio, in modo da poter poi distinguere i risultati di cui si è avuta una risposta immediata da quelli che l'agent ha memorizzato in locale ed inviato successivamente.

Messaggio 3: risultato esecuzione

```
<Response>
        <Id>IPCONFIG__20-08-2010__14-37-00<Id>
        <Type>result</Type>
        <Value>
        Configurazione IP di Windows

        Scheda Ethernet Connessione alla rete locale (LAN):

        Suffisso DNS specifico per connessione: workteam.local
        Indirizzo IPv6 locale rispetto al collegamento :
        fe80::2947:21e0:f9fe:724%12
        Indirizzo IPv4. . . . . . . . . . . : 192.168.252.160
        Subnet mask . . . . . . . . . . . . : 255.255.255.0
        Gateway predefinito . . . . . . . . : 192.168.252.254
        </Value>
</Response>
```

L'id del risultato ovviamente permette di identificare la richiesta a cui è associato. Il contenuto del campo <*Type*> può assumere due valori, *"result"* oppure *"error"*, e permette di discriminare quali esecuzioni sono andate a buon fine e quali hanno prodotto degli errori. Infine il campo <*Value*> contiene il risultato vero e proprio ossia l'output del comando eseguito oppure il messaggio d'errore se ci sono stati dei problemi.

Direttiva di disconnessione

Si osservi che la direttiva per far terminare la fase di ricezione dei vecchi risultati di cui si parlava in precedenza (vedi 4.1) non è altro che un file XML con la stessa struttura del messaggio 3 il cui id è *"stop"*. Allo stesso modo la direttiva per far terminare l'attesa di nuove richieste da parte dell'agent consiste in un messaggio di tipo 2 avente id *"stop"*.

4.4 l'agente

L'agente è l'applicazione installata nelle macchine da controllare che, congiuntamente al server, permette all'intero framework di funzionare. Come già detto nella fase di analisi, si tratta di un software sempre attivo in background, incaricato di connettersi periodicamente al server, verificare la presenza di qualche richiesta, eseguirla e ritornare i risultati. La scelta di implementarlo in Java permette a questo componente di essere estremamente flessibile ed adattabile alle varie architetture.

4.4.1 Gestione delle connessioni

Analogamente a quanto è stato fatto per il server, analizziamo come l'agente gestisce invio e ricezione dei messaggi. Anche in questo caso si è usata ampiamente la programmazione multithreading assegnando così compiti diversi a processi diversi. All'avvio dell'agent infatti, vengono creati due thread, uno chiamato clientThread e l'altro serverThread. Il primo si occupa solamente di connettersi ad intervalli di tempo prestabiliti al server ed inviare il messaggio di acknowledge (vedi connessione 1 in 4.1). Il secondo, invece, è costantemente in attesa della connessione (vedi connessione 2 in 4.1) da parte del server sulla porta specificata nel segnale di acknowledge. Appena questa viene stabilita il serverThread verifica la presenza di risultati di precedenti esecuzioni memorizzati localmente ed eventualmente li spedisce, concludendo il ciclo con l'invio della direttiva di terminazione (vedi 4.3). A questo punto il thread si mette in attesa dei comandi da lanciare, li esegue e ritorna uno per uno i risultati prodotti finché non sopraggiunge l'ordine di scollegarsi.

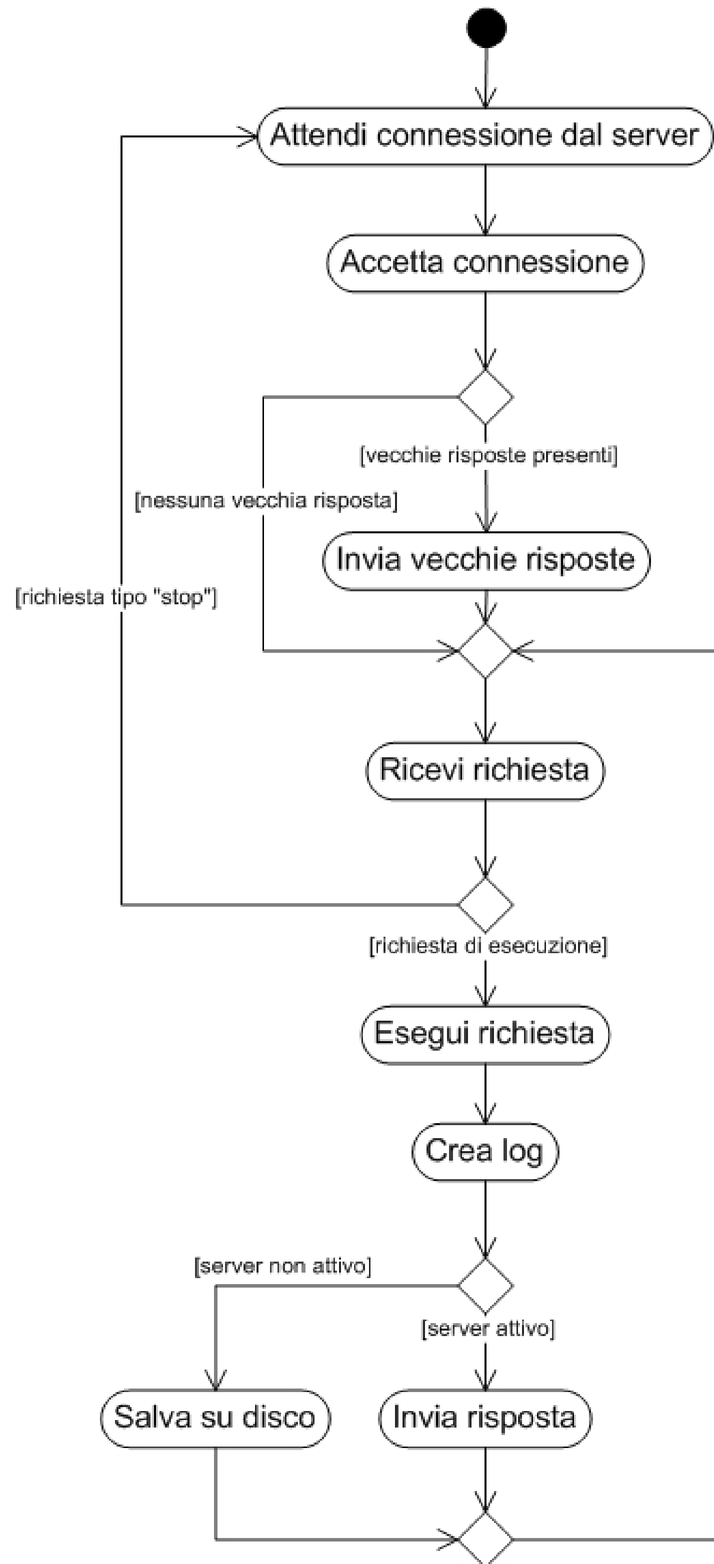


Figura 15: Diagramma delle attività dell'agente

4.4.2 Traduzione delle richieste e loro esecuzione

Ad ogni richiesta ricevuta, l'agent provvede a tradurre il messaggio recuperando dal file XML i relativi campi e decidendo di conseguenza come comportarsi. Per tutti i comandi di tipo *"cmd"* viene aperta una shell, all'interno della quale viene lanciato il contenuto del tag <*Command*>. Lo standard output e lo standard error vengono poi reindirizzati in due file diversi. A questo punto, valutando il contenuto di questi file, è possibile definire il tipo del messaggio di risposta. Se il file degli errori è vuoto significa che l'esecuzione è andata a buon fine dunque il campo <*Type*> del messaggio di risposta conterrà la stringa *"result"* e nel campo <*Value*> verrà copiato il contenuto del file dei risultati. In caso contrario, se si sono verificati degli errori, il campo <*Type*> conterrà la stringa *error"* e nel campo <*Value*> verrà copiato il contenuto del file degli errori.

4.5 L'INTERFACCIA GRAFICA

Una volta realizzati agent e server, il framework è in grado di funzionare nella sua versione manuale, ossia andando a modificare con un editor di testo i descrittori. Affinchè si possa sfruttare il sistema di controllo remoto con maggiore semplicità, un minor rischio di malfunzionamenti e soprattutto evitando di dover di volta in volta terminare l'applicazione server per modificare i file XML, è necessaria l'implementazione di una interfaccia grafica. Come si è detto in precedenza la scelta è stata quella di creare delle pagine web dinamiche che si appoggino ad una servlet.

4.5.1 La servlet

Il primo passo per fornire all'utente questo diverso metodo di interazione con il sistema è stata la realizzazione di una Java servlet, un'applicazione eseguibile nel web server di Workteam che permetta di controllare le pagine web. Essa si occupa di assolvere alle richieste che l'utente effettua, inoltrandole al server e definendo di volta in volta quale debba essere il contenuto delle pagine visualizzate nel browser.

L'idea è quella di interrogare la servlet attraverso richieste http aventi una ben precisa composizione della query string, e di ricevere le risposte in formato JSON. Quest'ultime vengono

poi interpretate dalle pagine web mediante jQuery, componendo così dinamicamente ciò che viene effettivamente visualizzato nel browser.

Interfaccia della servlet verso l'utente

Vediamo ora in che modo è possibile comunicare con la servlet dal lato utente, elencando tutte le operazioni che essa supporta. Per ognuna di queste descriviamo l'interfaccia in ingresso, ossia i parametri da introdurre nella query string per invocarla, e l'interfaccia in uscita ossia l'oggetto in formato JSON ritornato dalla servlet.

- Aggiungere una voce ad un descrittore: oper=A
  - Selezionare il descrittore degli agenti: descriptor=A
    - IN: oper=A&descriptor=A&id=...&name=...&ip=...&commands=...
    - OUT: {"result":"true"} oppure {"result":"false"} a seconda l'operazione abbia avuto successo oppure no
  - Selezionare il descrittore dei comandi : descriptor=C
    - IN: oper=A&descriptor=C&id=...&type=...& command=...&onError=...
    - OUT: {"result":"true"} oppure {"result":"false"} a seconda l'operazione abbia avuto successo oppure no
- Rimuovere una voce ad un descrittore : oper=R
  - Selezionare il descrittore degli agenti: descriptor=A
    - IN: oper=R&descriptor=A&id=...
    - OUT: {"result":"true"} oppure {"result":"false"} a seconda l'operazione abbia avuto successo oppure no
  - Selezionare il descrittore dei comandi : descriptor=C
    - IN: oper=R&descriptor=C&id=...
    - OUT: {"result":"true"} oppure {"result":"false"} a seconda l'operazione abbia avuto successo oppure no
- Aggiornare una voce ad un descrittore : oper=U
  - Selezionare il descrittore degli agenti: descriptor=A
    - IN: oper=U&descriptor=A&id=...&name=...&ip=...&commands=...

    * OUT: {"result":"true"} oppure {"result":"false"} a seconda l'operazione abbia avuto successo oppure no
  - Selezionare il descrittore dei comandi : descriptor=C
    * IN: oper=U&descriptor=C&id=...&type=... &command=...&onError=...
    * OUT: {"result":"true"} oppure {"result":"false"} a seconda l'operazione abbia avuto successo oppure no

- Filtrare le voci del descrittore attraverso l'id : oper=F
  - Selezionare il descrittore degli agenti: descriptor=A
    * IN: oper=F&descriptor=A&id=... se non viene specificato alcun id ritorna tutte le voci del descrittore
    * OUT: oggetto JSON delle voci selezionate
  - Selezionare il descrittore dei comandi : descriptor=C
    * IN: oper=F&descriptor=C&id=... se non viene specificato alcun id ritorna tutte le voci del descrittore
    * OUT: oggetto JSON delle voci selezionate

- Visualizzare un file di log relativo ad un agente : oper=L
    * IN: oper=F&id=...
    * OUT: {"result":"false"} se il file non esiste, in caso contrario al posto del valore "false" è presente il contenuto del file di log

- Verificare se sono state apportate delle modifiche ai descrittori dall'ultima volta che questo controllo è stato eseguito : oper=M
    * IN: oper=M
    * OUT: {"result":"true"} oppure {"result":"false"} a seconda ci siano state modifiche o meno.

Si osserva che il parametro *oper* deve essere sempre presente e definisce la tipologia di operazione. Il parametro *descriptor*, per le operazioni che lo richiedono, descrive invece su quale dei due descrittori si andrà a lavorare. Esso può assumere i valori *'A'* (agenti) o *'C'* (comandi).

Gli altri parametri rispecchiano i campi dei descrittori. Per modificarli è sufficiente costruire la query string a proprio piacimento, sostituendo ai puntini nello schema appena visto gli opportuni valori.

Vediamo ora un esempio per capire meglio come lavora la servlet. Assumiamo di avere i descrittori della sezione 4.2.1.

Query sting:

```
Servlet?oper=F&descriptor=A&id=WORKTEAM_1,WORKTEAM_2
```

Oggetto JSON ritornato:


```
{"col0":"id",col1":"name","col2":"ip","col3":"commands",
"col4":"isConnected","rows":[
{"id":"WORKTEAM_1","name":"workteam ufficio 1",
"ip":"192.168.252.161","commands":"","isConnected":"false"},
{"id":"WORKTEAM_2","name":"workteam ufficio 2",
"ip":"192.168.252.162","commands":"IPCONFIG,SCRIPT_BACKUP",
"isConnected":"false"}]}
```


Comunicazione con l'utente

La notazione JSON come si può vedere è abbastanza intuitiva. L'array *"rows"* contiene le informazioni relative agli agenti selezionati recuperate dal descrittore, eccezion fatta per la chiave *"isConnected"*, un flag applicato dal server che definisce se l'agente è attualmente connesso. La prima parte dell'oggetto, invece, potrebbe sembrare incomprensibile, in realtà si tratta dei nomi da applicare all'intestazione di una tabella. Le pagine web presenteranno infatti i dati in forma tabulare ma la scelta di includere le intestazioni è stata fatta più per compatibilità con altre parti di Euclide che per effettiva necessità.

Comunicazione con il server

Ora che si è spiegato come le pagine web comunicano con la servlet è necessario capire come essa inoltri le richieste al server. Come è stato accennato nella sezione 4.2.5 viene aperto un canale TCP con il server sul quale vengono inviati dei semplici messaggi che consistono in caratteri, stringhe o valori booleani. La scelta di non incapsularli in file XML è stata fatta per questioni di prestazioni, in quanto il contenuto del messaggio in genere è molto breve. In questo modo è sufficiente inviare al server, secondo un ordine prestabilito, i parametri ricevuti sulla query string. La servlet attende poi la risposta e mappa i risultati ottenuti in un oggetto JSON come già visto nello schema precedente.

4.5.2 Le pagine web dinamiche

L'ultimo tassello che manca da esaminare dell'intero sistema è quello che va a completare l'interfaccia grafica. Si tratta di una serie di pagine web dinamiche le quali, sfruttando la tecnologia AJAX e la libreria jQuery comunicano in modo asincrono con la servlet, gestendo le richieste dell'utente e permettendo degli aggiornamenti real time dei contenuti senza la necessità di ricaricare le pagine.

Le pagine di cui ha senso fornire una descrizione sono essenzialmente due: *manageAgent.jsp* e *manageCommand.jsp*. Come si può intuire dal nome, esse gestiscono rispettivamente agenti e comandi ed aggiungono ulteriori funzionalità alla semplice gestione manuale dei descrittori.

manageAgent.jsp si presenta come una lista contenente tutte le informazioni relative agli agenti noti. Il sistema permette di inserire, rimuovere e modificare gli agenti oltre che riconoscere a colpo d'occhio quali siano quelli che stanno lavorando, dato che rimangono evidenziati fino a quando si scollegano.

Figura 16: Esempio della pagina manageAgent.jsp

Inserimento e aggiornamento avvengono attraverso l'uso di finestre di dialogo molto simili realizzate in jQuery. Nel primo caso tale finestra si richiama cliccando sul pulsante relativo, nel secondo cliccando sull'id dell'agente che si intende modificare.

Figura 17: Esempio form inserimento

Come si può vedere attraverso questa schermata è possibile depositare nella coda di un agente tutti i comandi che si vuole vengano eseguiti alla sua prossima connessione. Al momento della conferma, il codice jQuery provvede ad effettuare una chiamata AJAX alla servlet, inviando una richiesta http la cui query string è stata correttamente composta al fine di attuare l'operazione desiderata dall'utente. Su questo stesso principio lavora anche il form di inserimento e visualizzazione dei logs, richiamabile cliccando sui disegni dei file nella colonna "LOGS", di cui vediamo un esempio di seguito.

Figura 18: Esempio visualizzazione log

La pagina *manageCommand.jsp* gestisce invece le informazioni relative al descrittore dei comandi sfruttando la stessa tecnica appena descritta: le voci del descrittore vengono caricate in una tabella ed è possibile interagire con esse per mezzo di finestre

di dialogo realizzare in jQuery, che andranno poi a interfacciarsi con la servlet comunicandole le operazioni da svolgere.

5 MANUALE UTENTE

Indice

5.1 Requisiti di sistema 45
5.2 Modalità di avvio 45
5.3 Esportazione del progetto da Eclipse 46
5.4 Problemi noti 46

5.1 REQUISITI DI SISTEMA

L'agente richiede l'installazione di JRE 6. Lo stesso vale per il server con l'aggiunta della libreria Javamail. La servlet infine deve essere inserita in un web container come Apache Tomcat o IBM WebSphere. Dato che la libreria di Javamail (mail.jar) non è inclusa nella JRE6, essa verrà copiata nella stessa directory del .jar avviabile del server, sotto la cartella /lib. Lo stesso vale per la libreria rt.jar nel caso si usasse una diversa implementazione di JRE6 dove non fosse incluso xerces. Questi percorsi sono definiti nel manifest file del jar avviabile.

5.2 MODALITÀ DI AVVIO

I file avviabili vengono appunto forniti in formato jar. agent.jar e server.jar sono le versioni a riga di comando. Per avviare l'agent è necessario digitare sulla shell:

java –jar agent.jar <id agente> <porta ascolto agent> <ip server> <porta ascolto server> <timer riconnessione>

Per avviare il server invece, essendo dotato di un file di configurazione è sufficiente digitare:

java –jar server.jar

agentGui.jar e serverGui.jar sono le versioni che fanno uso delle librerie Swing per gestire una semplice console grafica e la TrayIcon. Per avviarle è sufficiente lanciare il file .jar. Per l'agente una serie di finestre di dialogo permetterà di definire i vari parametri.

5.3 ESPORTAZIONE DEL PROGETTO DA ECLIPSE

Il progetto Euclide-SCR, all'interno di eclipse, contiene nella sua root il file MANIFEST.MF . Tale file specifica con la voce Class-Path il percorso delle librerie citate nella sezione "requisiti di sistema", aggiuntive rispetto a quelle di ambiente. Inoltre la classe avviabile specificata di default è core.Server, dunque usare il MANIFEST.MF solamente per esportare il server nella sua versione a riga di comando. In caso contrario modificare prima il MANIFEST.MF, definendo come classe avviabile ServerGui.

5.4 PROBLEMI NOTI

Il server si avvia correttamente. Alla prima connessione di un agente non viene trovata la classe javax/mail/Address. Significa che la libreria mail.jar non viene trovata. Potrebbe essere stata fatta una esportazione da eclipse senza definire come manifest file quello presente nel workspace. Soluzione:

- Verificare che essa sia presente all'interno della cartella lib/ nella directory in cui è presente il .jar avviabile;
- Verificare il file MANIFEST.MF all'interno della cartella META-INF del jar avviabile. La voce Class-Path deve risultare come segue: Class-Path: . lib/mail.jar;

Il server si avvia correttamente. Alla prima connessione di un agente non viene trovata la classe DOMImpl. Significa che il sistema usa una diversa implementazione di xerces avendo delle librerie di sistema diverse da quelle della JRE6 (ad esempio quelle di ibm). Soluzione:

- Copiare la libreria rt.jar presente nella root di JRE6 all'interno della cartella lib/ nella directory in cui è presente il .jar avviabile;
- Verificare il file MANIFEST.MF all'interno della cartella META-INF del jar avviabile aggiungendo al Class-Path la seguente voce: lib/rt.jar;

6 CONCLUSIONI

Indice

6.1 Diagramma di Gantt delle fasi del tirocinio . 47
6.2 Considerazioni sul software sviluppato 47
6.3 Considerazioni personali sull'esperienza . . . 48

6.1 DIAGRAMMA DI GANTT DELLE FASI DEL TIROCINIO

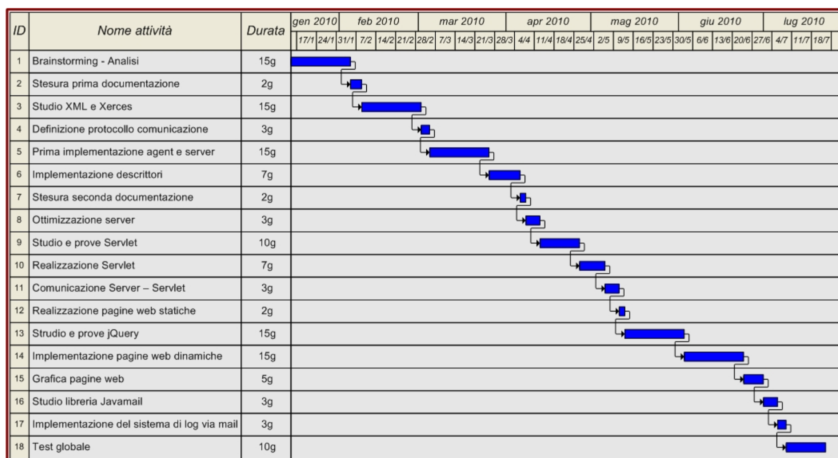


Figura 19: Diagramma di Gantt delle fasi del tirocinio

6.2 CONSIDERAZIONI SUL SOFTWARE SVILUPPATO

Possiamo considerare gli obiettivi più che raggiunti. Il framework per la gestione di sistemi remoti è stato installato e testato presso Workteam permettendo all'azienda di automatizzare tutte quelle operazioni da svolgere sulle macchine dei clienti che non richiedono un'interazione real-time. Tutte le idee e le richieste emerse nella fase di brainstorming sono state analizzate a

fondo, rendendo possibile la realizzazione di un software complesso articolato in diverse componenti. Nella fase di analisi si era inoltre discusso di una possibile integrazione con il progetto Euclide. Anche in questo senso possiamo considerare in gran parte lo scopo raggiunto. Il meccanismo di reportistica via mail permette al sistema già esistente l'interpretazione dei messaggi del nuovo framework. A questo punto, grazie all'interfaccia fornita alla servlet, è possibile automatizzare l'esecuzione di particolari comandi in risposta alle segnalazioni provenienti da Euclide. Nel corso del tirocinio inoltre, ci si è resi conto della potenza e della versatilità di un software di questo tipo. Permettendo l'esecuzione di qualsiasi script o comando sta alla fantasia dell'utente finale decidere che tipo di operazioni far svolgere agli agenti. Per fare un esempio si pensi alla rimozione dei file di log: è una funzione non prevista dal sistema ma uno script pensato in modo intelligente può assolvere a questo compito ampliando ulteriormente le potenzialità del sistema.

6.3 CONSIDERAZIONI PERSONALI SULL'ESPERIENZA

Mi ritengo veramente soddisfatto dell'esperienza svolta e dei risultati ottenuti. Nel corso del tirocinio ho imparato in che modo si affronta un progetto in ambito professionale, dalla sua fase di stesura delle idee, alla loro analisi sino all'effettiva implementazione.

Ho avuto modo di applicare numerosi concetti appresi nel corso dei miei studi come la programmazione multithreading vista in Sistemi Operativi oppure il protocollo TCP/IP affrontato nel corso di reti. Inoltre ho potuto arricchire il mio bagaglio culturale facendo uso di tecnologie e linguaggi di cui, fino ad ora, avevo solo sentito parlare. Oltre all'XML, di cui avevo già qualche conoscenza, mi sono cimentato nella programmazione di servlet Java e di pagine web dinamiche attraverso l'uso della libreria jQuery. In questo ambito ho dovuto studiare inoltre il formato JSON, rispolverando inoltre qualche competenza di html e JavaScript.

Le maggiori difficoltà che ho incontrato sono state la dimensione del progetto e la numerosità di ambiti interessati. Inizialmente il progetto facilmente gestibile in quanto era costituito semplicemente da agent e server. Procedendo nell'implementazione il numero delle componenti è poi aumentato, compor-

tando un aumento del tempo necessario soprattutto per la fase di testing. Riposizionare tutte le parti del sistema sulla relativa macchina ad ogni prova richiedeva infatti un costo non indifferente in termini di tempo.

BIBLIOGRAFIA

[1] Cay Horstmann, *Concetti di informatica e fondamenti di Java*, III edizione, Apogeo.

[2] Andrew S. Tanenbaum, *Reti di calcolatori*, IV edizione, Prentice Hall.

[3] Eric van der Vlist, *XML Schema*, O'Reilly.

[4] Jason Hunter, William Crawford, *Java Servlet Programming*, O'Reilly.

[5] Martin Fowler, *UML distilled*, III edizione, Pearson-Addison Wesley.

[6] *http://www.json.org/*

[7] *http://www.jquery.com/*

[8] *http://www.w3schools.com/*

Alberto Boccato: *Euclide: sviluppo di un framework per il controllo push-pull di sistemi remoti*, relazione di tirocinio, 28 Settembre 2010

e-mail
boccato.alberto@gmail.com