UNIVERSITÀ DEGLI STUDI DI CAMERINO

SCUOLA DI SCIENZE E TECNOLOGIE

Corso di Laurea Specialistica in Informatica 23/S

Master and Butler: un sistema command and control distribuito

---

RELATORE
Prof. Fausto Marcantoni

TESI DI LAUREA DI
Roberta Mancini

Anno accademico 2015/2016

Indice

Capitolo 1
Introduzione

"-Il Lato Oscuro è più forte?
-No! No! No... Più rapido... Più facile... Più seducente..." [1]

Il confronto bene-male ci coinvolge nella vita di tutti giorni e in tutti i campi. Nell'ambito informatico il ruolo del malvagio è affidato all'hacker, che da un lato è disprezzato perché può colpire, indistintamente, utenti e sistemi, da l'altro è coperto da un alone di fascino perché ogni volta deve adottare soluzioni brillanti per aggirare gli ostacoli e far si che il virus prodotto attecchisca. Viene spontaneo chiedersi perché gli hacker riversino le loro conoscenze a scopi maligni, come se davvero fossero attratti dal Lato Oscuro... A volte gli hacker si convertono e trovano impiego proprio nella sicurezza delle reti. Altre volte i loro prodotti sono impiegati a scopi di ricerca per testare le falle della rete o la vulnerabilità dei sistemi. Nel mio caso, proprio un virus informatico, *Gcat* [GCa], è diventato lo spunto per lo sviluppo del presente lavoro.

Gcat è un programma malevolo che, inoculato tramite e-mail nel dispositivo dell'ignaro prescelto, riceve comandi da remoto e li esegue. Il risultato viene poi spedito al committente del virus e il destinatario viene così privato della sua privicy e non solo: il malvagio hacker potrebbe cancellare file, trasferire file, fare uno *screenshot*, bloccare lo schermo, forzare il check in, avviare un *keylogger* per poi fermarlo. Ciò che rende efficace il virus è il protocollo di trasmissione utilizzato, non solo il virus viene trasmesso per posta, ma una volta installato, questo programma comunica con l'hacker tramite *Gmail*. Il sistema perfetto per eludere firewall e restrizioni sulla rete. Le porte utilizzate per la posta elettronica sono sempre aperte. Ecco perché l'idea di sfruttare *Gmail* come protocollo comunicativo in *Master and Butler*, il sistema distribuito oggetto di questa tesi. Utilizzare *Gmail* per implementare del protocollo tra i dispositivi rende l'applicazione flessibile, snella, facilmente configurabile, scalabile e può essere utilizzata anche in condizioni di restrizione delle politiche di reti.

[1] Dialogo tra Luke e Yoda nel film "*Star Wars: The Empire Strikes Back*". USA 1980.

1.1 Presentazione del lavoro

Master and Butler è un sistema *Client/Server* di tipo *Master/Slave* in cui il Master invia un comando ad uno degli *Slave* attivi e il risultato viene riportato al *Master*.

La schermata dell'applicazione è composta da un pannello di controllo contenente uno schedario per ogni *Butler*, all'interno del quale è indicato se il relativo *Butler* è attivo o meno, i possibili comandi da far eseguire e i risultati dei comandi inviati.

I comandi che si possono eseguire sono: *Print screen* della macchina remota e altri comandi da *prompt del DOS*, con l'opzione di far eseguire l'operazione istantaneamente o in un precisato momento, specificando il giorno, mese, anno e ora.

Il protocollo comunicativo, basato sullo scambio di messaggi e-mail, rende l'applicativo *Master and Butler* adatto alle applicazioni di controllo remoto come gestione e controllo dei laboratori informatici, dei cluster, applicazioni sulla domotica e prodotti di sorveglianza.

Le informazioni trasmesse da *Gmail* sono protette da crittografia TLS (*Transport Layer Security*), rendendo *Master and Butler* un'applicazione efficace dal punto di vista della privacy.

L'applicazione è stata sviluppata in Java versione 8 [Java], il linguaggio orientato agli oggetti lanciato dalla Sun Microsystems nel 1995, sviluppato da un team coordinato da James Gosling, diventato di larga diffusione in tutto il mondo. La scelta è ricaduta sul linguaggio Java per la sua flessibilità, per la sua portabilità e per la sua indipendenza dalla piattaforma utilizzata, nonché per la notevole specializzazione nella programmazione distribuita.

1.2 Struttura della tesi

Il capitolo 2: Il sistema *Master and Butler*, illustrerà il sistema con l'aiuto degli schemi UML, verrà spiegato come i componenti del sistema interagiscono tra loro e il protocollo utilizzato, focalizzando l'attenzione sulla programmazione concorrente e distribuita e le classi coinvolte nel protocollo.

Il capitolo 3: I *Butler*, riguarda le caratteristiche dei *Butler,* la loro configurazione e la loro funzione all'interno del sistema: invio segnale di stato attivo, stato di ascolto, intercettazione del comando, esecuzione del comando eventualmente con tempo ritardato e ritorno del risultato del commando prodotto.

Il capitolo 4: Il Master, tratterà l'architettura del *Master*, illustrando la modalità di esecuzione, l'interfaccia grafica con i possibili comandi da inviare per ogni *Butler* e la visualizzazione delle risposte ottenute ed infine i Thread che governano l'aggiornamento dell'interfaccia grafica con le risposte dai *Butler*

Infine, nel capitolo 5 dedicato alle conclusioni ci saranno il riepilogo e alcune osservazioni sul lavoro prodotto, i suggerimenti sulle migliorie e i possibili campi di applicazione.

Capitolo 2
Il sistema Master and Butler

"Il sabotaggio delle comunicazioni può significare solo una cosa: l'invasione."[2]

La comunicazione riveste un ruolo fondamentale fin dagli albori della Terra e ha contribuito allo sviluppo e all'evoluzione dell'uomo fino al giorno d'oggi. Il pianto di un bambino, il semplice scambio di saluti, un ponte radio che mette in collegamento la più remota parte di terra con il resto del mondo, la larga diffusione di Internet, la messaggistica telefonica largamente usata, offrono esempi di comunicazione, senza di essa il mondo sarebbe impensabile e porterebbe l'uomo alla sua involuzione.

L'uomo non ha inventato solo la comunicazione ma, con essa, in modo spontaneo, sono nate anche tutte le regole più o meno formalmente descritte e definite tra una o più entità. L'alzare la mano per prendere la parola in un'assemblea, prendere il numero e aspettare il proprio turno quando si va alla posta, salutare con un "Ciao", un "Buongiorno" o un "Buonasera" a seconda dell'interlocutore e dal momento della giornata, sono tutti semplici esempi di protocolli che fanno parte della vita quotidiana.

In questo capitolo si parlerà del funzionamento generale del sistema incentrando l'attenzione proprio sul protocollo di comunicazione e la programmazione concorrente e distribuita.

2.1 Funzionamento generale del sistema

Master and Butler è un'applicazione *Client/Server* di tipo *Master/Slave* (figura 1) in cui il *Master* invia il comando tramite un pannello grafico ai *Butler*, questi, una volta individuato il tipo di comando, lo eseguono e restituiscono il risultato al *Master* che a sua volta visualizza il risultato sempre sul pannello di controllo. Il *Master* ha un pannello di controllo per ogni *Butler,* all'interno del quale è specificato se il *Butler* è attivo o meno.

[2]Sio Bibble nel film "*Star Wars: Episode I - The Phantom Menace*". USA 1999.

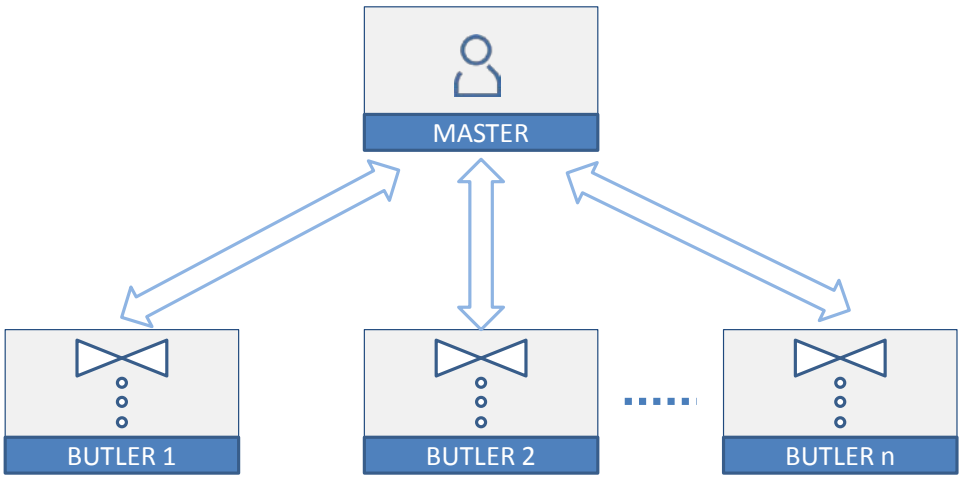

Figura 1 - Architettura di tipo *Master/Slave*

Entrando nel particolare del sistema, la figura 2 mostra come è implementato il mezzo di comunicazione tra il *Master* e i *Butler* impiegando le caselle di posta di *Gmail* del *Master* e dei *Butler*.

Il *Master* comunica con i *Butler* in modo asincrono, infatti spedisce un messaggio, indipendentemente dal fatto che il Butler possa essere attivo o meno, e continua ad effettuare le proprie operazione: invia ulteriori comandi allo stesso *Butler* o ad altri *Butler* o ancora controlla i risultati dei comandi inviati. I comandi possono essere richiesti con effetto immediato o ritardato specificando il giorno, il mese, l'anno e l'ora di esecuzione. Il *Butler* a sua volta, non curante di ciò che accade al *Master* o agli altri *Butler*, esegue i comandi ricevuti al momento che gli sono stati ordinati e rispedisce i risultati al *Master*.

Nelle reti di telecomunicazione ogni dispositivo deve essere individuato il modo univoco, ad esempio nella rete informatica che utilizza il protocollo di rete IP [KuKe], ogni dispositivo è individuato da un indirizzo IP. In Master and Butler, sebbene i

dispositivi condividano un unico indirizzo e-mail, ogni Butler è identificato in modo univoco, specificandolo nel corpo dell'e-mail. Il destinatario del messaggio inviato dal *Master* è scritto prima del comando.

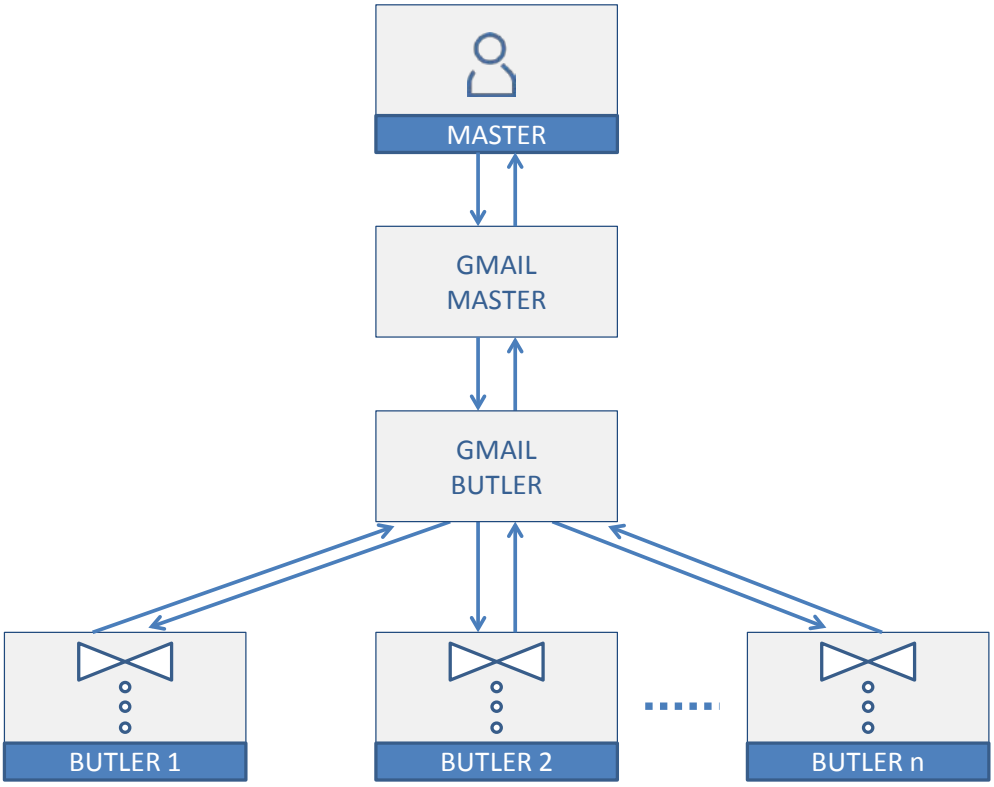

Figura 2 - Scambio messaggi tra Master e Butler tramite Gmail

Analizzando la topologia delle reti, *Master and Butler* implementa una topologia a stella, in cui ogni nodo è collegato in maniera univoca con il nodo centrale. Infatti non c'è dialogo tra i *Butler*, ma solo con il *Master*.

Le primitive di comunicazione indicano come avviene lo scambio dei messaggi inviati e ricevuti: *send* e *receive*. In *Master and Butler* queste funzioni sono assolte dall'invio del messaggio sotto forma di e-mail, in *send* il *Master* specifica il mittente, la data di esecuzione e il comando. In *receive* i *Bulter* leggono i messaggi in arrivo nella cartella a loro dedicata. Un *thread* specifico sposta ogni e-mail dalla cartella principale alla rispettiva cartella relativa al destinatario effettivo; *Butler1* contiene i comandi destinati al *Butler1* e via dicendo. Il *thread* smistatore legge tutte le e-mail, le valida controllando il mittente e il codice dispositivo (un sistema di verifica che permette di testare

la veridicità del messaggio) e poi le sposta nella cartella corrispondente. Analogamente nel *Master* i messaggi vengono letti dopo che sono stati spostati nelle cartelle identificative del mittente: il *Thread* smistatore che legge le e-mail, identifica il mittente e verifica il codice dispositivo e infine sposta l'e-mail nella cartella relativa. Nella cartella *Butler1* ci sono tutte le e-mail spedite da *Butler1* e così via, cosicché ogni *thread* associato al relativo *Butler*, legge solo i messaggi a lui riguardanti e li comunica all'interfaccia grafica del *Master*.

2.1.1 Log del Sistema

Tutte le operazioni sono tracciate all'interno della casella di posta elettronica nella cartella delle e-mail inviate del Master e del Butler. Le e-mail ricevute sono cancellate dopo averle processate per una scelta di ordine e pulizia della casella di posta elettronica.

Qualora si abbia la necessità di avere la tracciabilità di questi dati, basta modificare il codice e spostarle dopo averle processate invece di cancellarle.

2.2 Il protocollo di comunicazione utilizzato

Il protocollo di comunicazione utilizzato da *Master and Butler* è illustrato nella figura 3 dove sono definiti i messaggi che si scambiano il *Master* e i *Butler* al passare del tempo.
Il *Butler* ciclicamente invia un segnale per far sapere al *Master* che è attivo. Indipendentemente da questo, il *Master* invia uno dei suoi comandi: un *print screen* o un comando di linea da eseguire dal *promt* dei comandi del DOS. Il *Butler* riceve il comando, lo elabora e restituisce il risultato al Master.

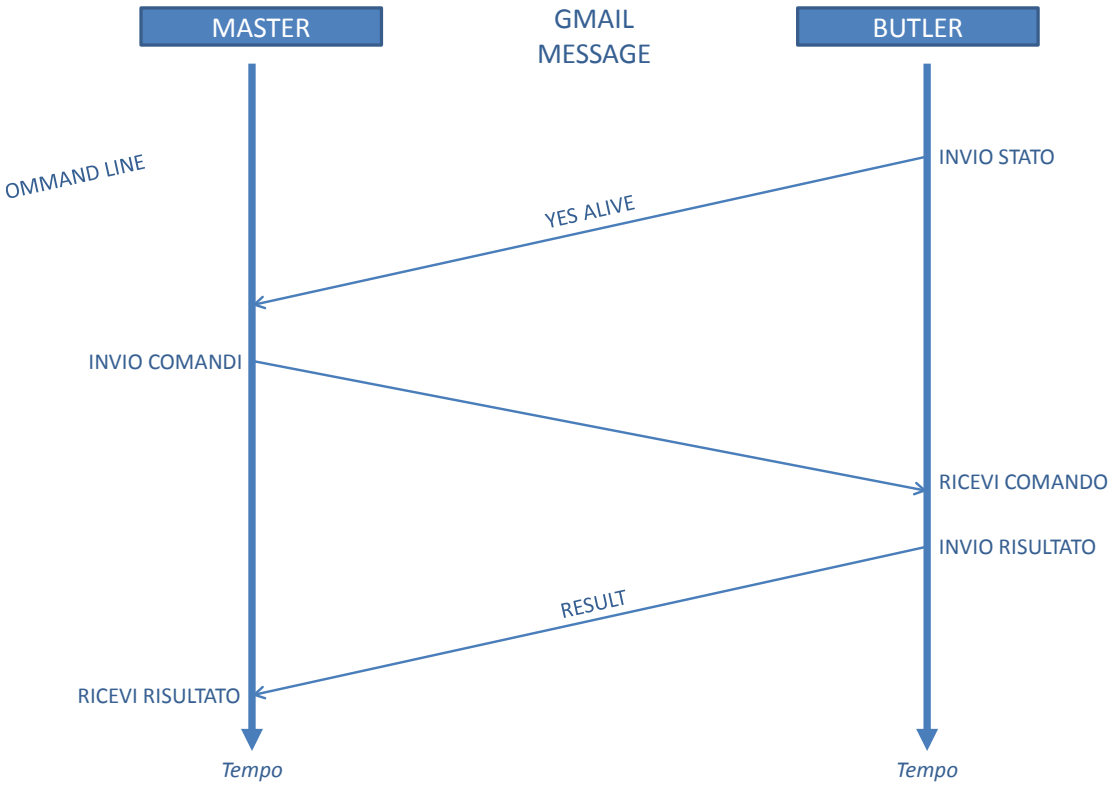

Figura 3 – Scambio messaggi tra *Master* e *Butler*

Il protocollo, come già spiegato, è asincrono, ogni oggetto agisce autonomamente senza aspettare il segnale di avvenuta ricezione dall'interlocutore. Sia il *Master* sia i *Butler* processano le loro operazioni senza preoccuparsi che il destinatario abbia ricevuto o meno il proprio messaggio. Nel caso vi siano problemi con la posta o con la connessione, si deve tenere in considerazione che in ogni caso alla consegna avvenuta verrà processato ugualmente, a patto che sia letto entro i termini di esecuzione richiesti dal *Master*, e verrà spedito il risultato dal lato *Butler*; risultati visualizzati potrebbero corrispondere a vecchie richieste. In ogni caso, nella casella di posta elettronica rimangono i messaggi dai quali si può ricavare lo storico dei comandi e rappresentano i log del sistema. Inoltre i *Butler* spediscono i risultati sotto forma di file allegato all'e-mail che il Master salva in locale e in cui il nome di questo file è la data (anno, mese, giorno, ora, minuti e secondi) dell'esecuzione del comando da parte del *Butler*.

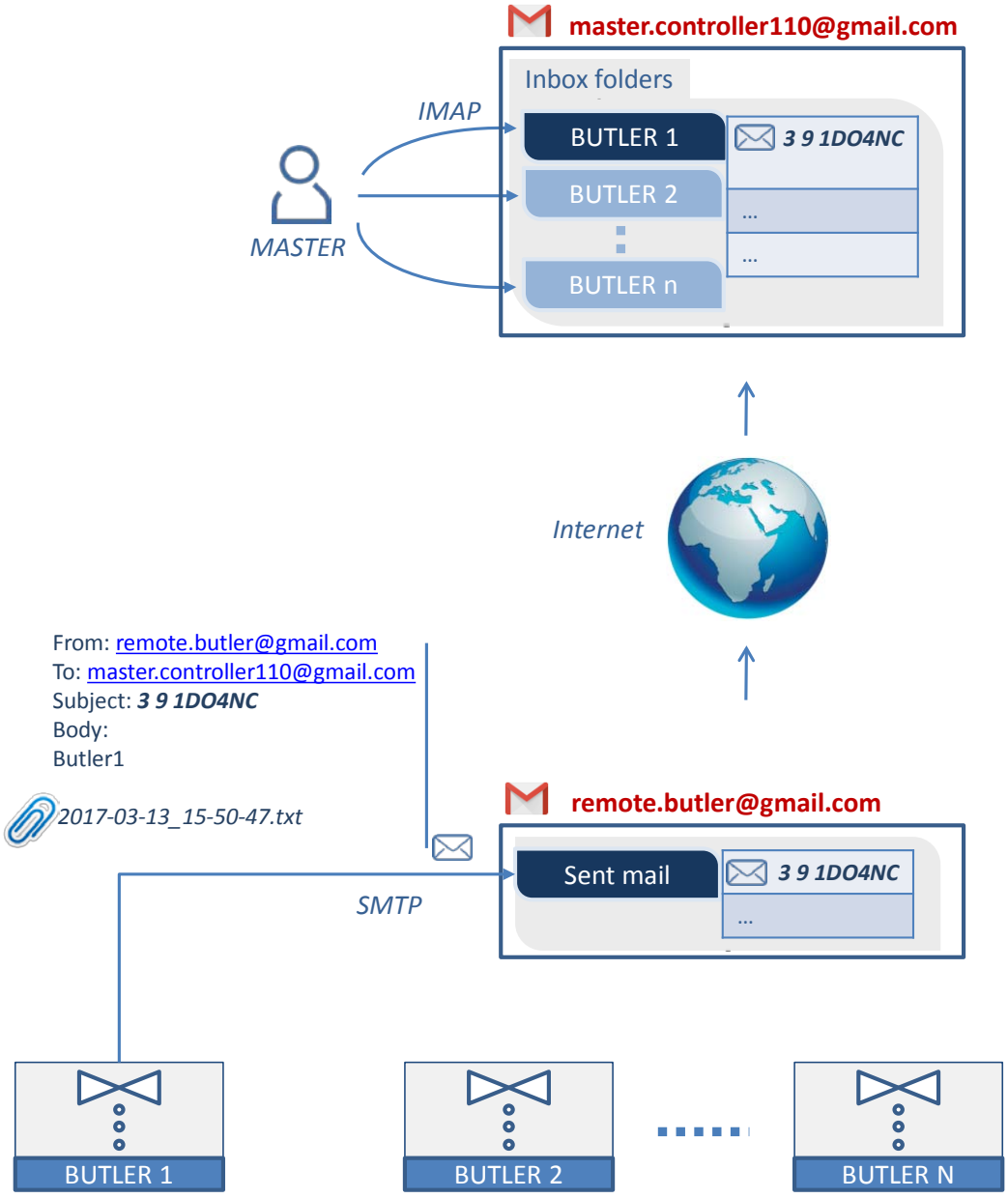

Figura 4 - Flusso generato dall'invio di un messaggio uno dei *Butler* al *Master*

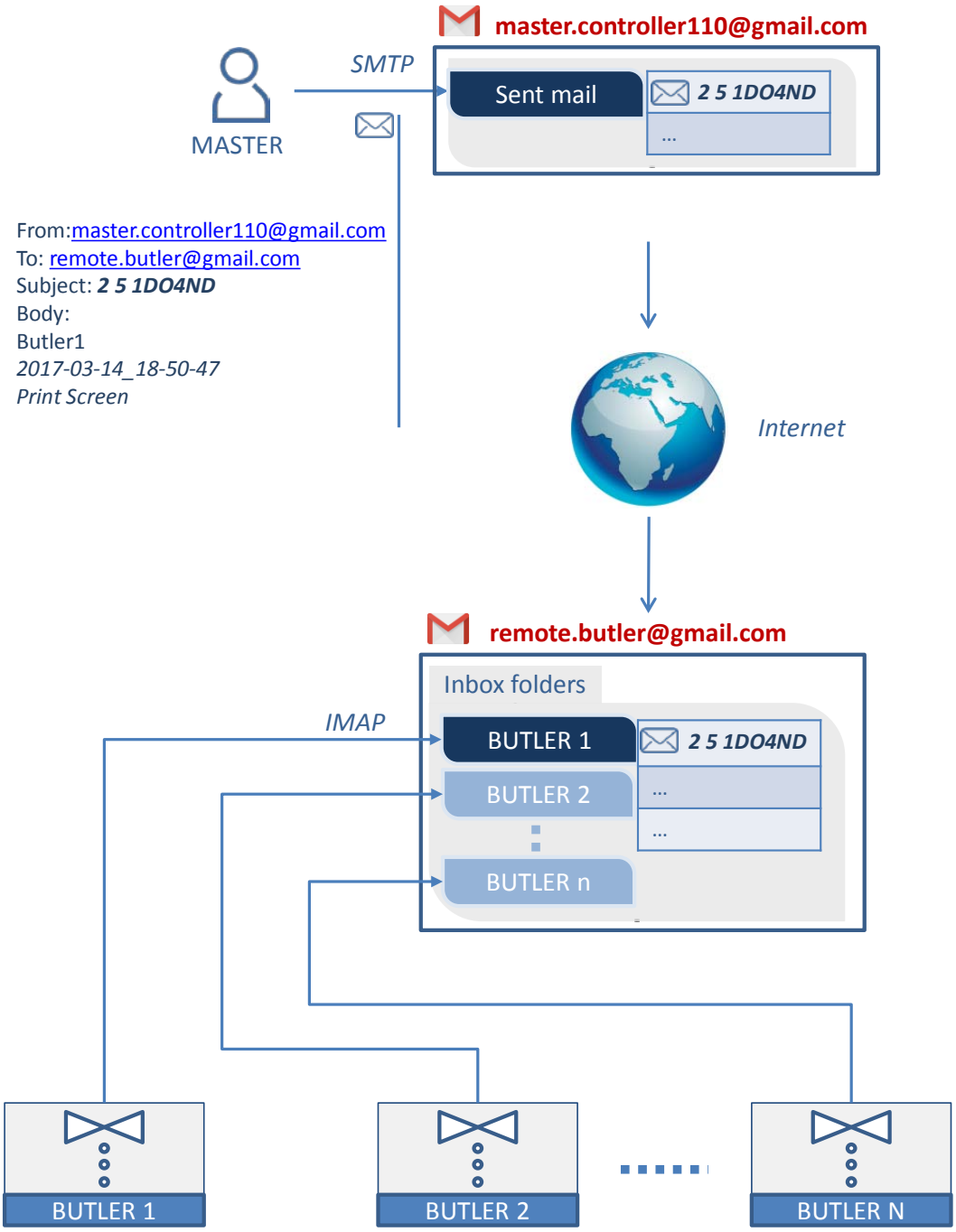

Figura 5 - Flusso generato dall'invio di messaggi dal *Master* a *Butler1*

La figura 4 mostra il flusso generato dall'invio di un messaggio da un *Butler* al *Master* dove si mette in evidenza il protocollo SMTP usato per la trasmissione e quello per la ricezione (IMAP), il server di posta elettronica Gmail condiviso, usato dai *Butler* e l'accesso del *Master* ad ogni cartella relativa ai *Butler* nella propria casella di posta elettronica.

Alcune osservazioni:

- L'accesso al protocollo SMTP si ha solo quando il Butler ha necessità di inviare un messaggio. L'accesso al protocollo IMAP da parte del *Master* si ha indipendentemente dal fatto che si abbia o meno un messaggio.
- Per il *Butler*, la cartella di posta elettronica delle e-mail inviate contiene le e-mail inviate da tutti i *Butler*
- La posta elettronica delle e-mail da leggere del *Master* è suddivisa in cartelle ognuna con le e-mail ricevute dal *Butler* di appartenenza.

Analogamente in figura 5 è descritto il flusso di un messaggio inviato dal *Master* a un *Butler* (nello specifico *Butler1*). Anche in questo caso il protocollo di trasmissione è SMTP e quello di ricezione è IMAP. Il *Master* spedisce il messaggio tramite posta elettronica, i *Butler* consultano in maniera indipendente la propria cartella per controllare eventuali messaggi, solo il *Butler* destinatario del messaggio in questione leggerà il messaggio.

Anche in questa modalità le osservazioni sono simili a quelle precedenti:

- L'accesso al protocollo SMTP si ha solo quando il *Master* ha necessità di inviare un messaggio. L'accesso al protocollo IMAP da parte dei *Butler* si ha indipendentemente dal fatto che si abbia o meno un messaggio.
- Per il *Master*, la cartella di posta elettronica delle e-mail inviate contiene le e-mail inviate da tutti i *Butler*
- La posta elettronica delle e-mail da leggere dei *Butler* è suddivisa in cartelle ognuna con le e-mail destinate al *Butler* di appartenenza.

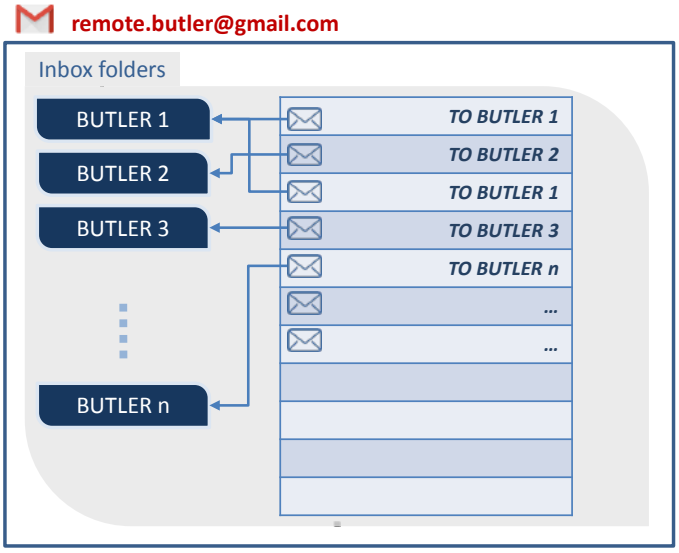

Figura 6 – Funzione dello smistatore del Butler

Lo smistatore della posta dei *Butler* è rappresentato in figura 6, in base al destinatario, che è indicato nel corpo dell'e-mail, lo smistatore sposta il messaggio nella cartella di appartenenza.

Analogamente lo smistatore del *Master* opera nella casella di posta elettronica del *Master*. In base al mittente, sempre indicato nel corpo dell'e-mail, il messaggio viene spostato nella relativa cartella.

Le funzionalità del protocollo e le procedure impiegate per l'implementazione del sistema sono riassunte di seguito:

- Il *Master* usa un indirizzo di posta elettronica di *Gmail*
- Per la lettura delle e-mail è utilizzato il protocollo IMAP; ciò rende possibile l'accesso alla casella di posta elettronica da più dispositivi o utenti, anche contemporaneamente, e nello specifico i client al massimo possono essere 15 come indicato nelle specifiche *Gmail* [GuiG]

- Per l'invio di messaggi è utilizzato il protocollo SMTP
- I *Butler* dispongono di un unico indirizzo di posta elettronica di *Gmail* in maniera condivisa
- Ogni *Butler* per leggere i messaggi a lui diretti, consulta la propria cartella contenente le *e-mail* con i comandi da eseguire
- Lo smistatore dei *Butler* si occupa di spostare le e-mail della posta "in arrivo" nelle cartelle relative ai *Butler* destinatari
- Le e-mail lette dai *Butler* contengono i comandi da eseguire
- Le e-mail spedite dai *Butler* contengono i risultati dei comandi eseguiti
- Lo smistatore del *Master* si occupa di spostare le e-mail dalla posta "in arrivo" alle cartelle relative ai *Butler* mittenti
- Le e-mail lette dal *Master* contengono i risultati dei comandi eseguiti dai *Butler*
- Le e-mail spedite dai *Master* contengono i comandi da far eseguire ai *Butler*
- Il server di posta utilizzato sia per il *Master* che per i *Butler* è *Gmail*
- Infine, nota che merita di essere puntualizzata, il protocollo si basa sulla schedulazione dell'azione da compiere indicata nel messaggio e non sulla sincronia tra messaggio spedito e messaggio ricevuto. La spedizione e ricezione da entrambe le parti può arrivare in tempi variabili.

Per la gestione delle e-mail è stato utilizzato il *package JavaMail API* [JavM], nato proprio per le applicazioni di messaggistica e e-mail. JavaMail API è disponibile come libreria opzionale da installare nella piattaforma Java.

2.2.2 Verifica codice dispositivo

Master and Butler ha una verifica dell'autenticità del messaggio basata su verifica codice dispositivo simile a quella usata dalle banche per verificare la correttezza dell'operatore che effettua un movimento bancario su conto on-line. Il sistema bancario chiede all'utente cosa c'è scritto in una determinata casella del codice dispositivo a lui in possesso, l'utente consulta la casella, scrive il contenuto e l'operazione è validata.

In *Master and Butler* la verifica dell'autenticità del messaggio è fatta, oltre che su verifica del mittente, anche su verifica di un codice alfanumerico a dieci cifre. Entrambe le parti hanno lo stesso listato di codici segreti; chi scrive sceglie casualmente uno di quei codici e lo spedice con il messaggio e-mail. Chi riceve verifica che il codice rice-

vuto appartenga alla lista dei codici dispositivi in suo possesso. Ciò vale sia per il *Master* sia per il *Butler*.

Il codice è spedito come oggetto della e-mail.

Il codice dispositivo andrebbe sostituito regolarmente.

2.3 Schemi UML

Il linguaggio di modellazione *UML* (*Unified Modeling Language*) [UML] nasce nel 1996, dalle mani di Grady Booch, Jim Rumbaugh e Ivar Jacobson, su richiesta dell'OMG (*Object Management Group*) con lo scopo di produrre uno standard unico per la modellazione progettazione e programmazione di sistemi ad oggetti.

2.3.1 Casi d'uso

Nella figura seguente si riportano i casi d'uso relativi agli elementi coinvolti, in cui sono dettagliate le funzionalità ed il flusso dei messaggi recepiti dal *Master* e dai *Butler*. I flussi vanno letti da sinistra a destra.

Il *Master* riceve dai *Butler* informazioni sul proprio stato: ciclicamente, questi inviano un messaggio di stato attivo per far sapere al *Master* se siano attivi o meno.

Indipendentemente da questo, il *Master* può inviare un comando. I comandi che si possono inviare di tipo *Print Screen* o Comando da console. In ogni caso il *Master* riceve il risultato del comando inviato. Nel caso in cui l'ordine di comando sia stato letto fuori tempo massimo, il Butler risponde con una dicitura standard: "Comando letto fuori tempo massimo. Non è stato eseguito", recapitato sempre in forma di allegato (.txt)

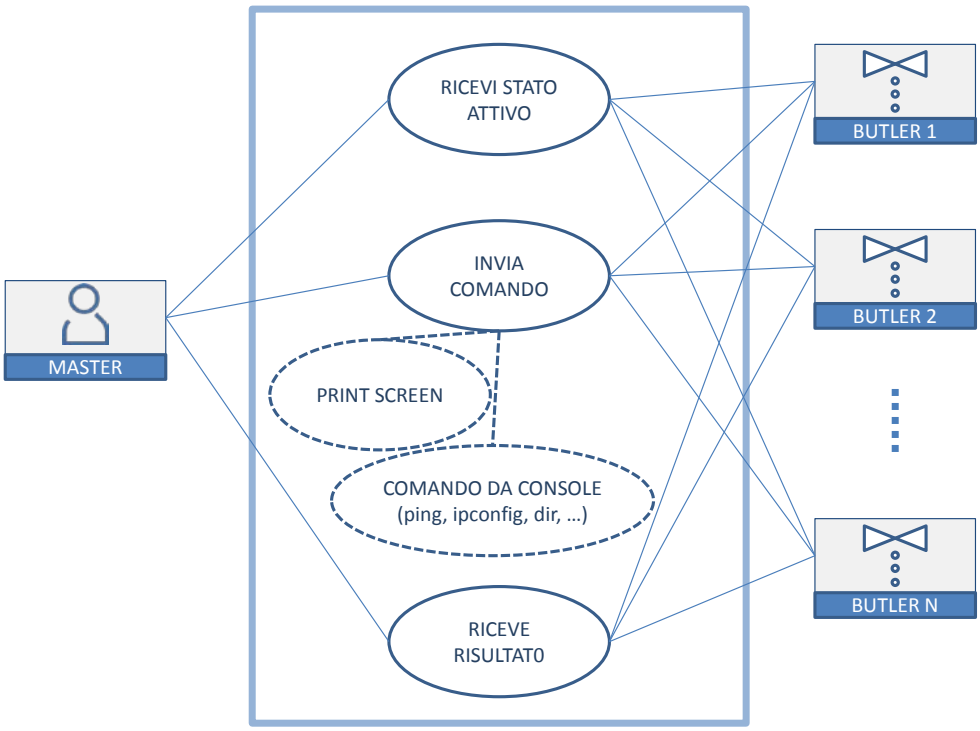

Figura 6 Casi d'uso

2.3.2 Corsie

La figura seguente mostra il diagramma di sequenza, evidenziando il dettaglio dei messaggi che gli oggetti si scambiano. Il *Butler* ciclicamente manda un messaggio di stato attivo; se il *Master* invia un comando di *Print Screeen* il *Butler* risponde inviando la schermata fotografata in formato .jpg; se il *Master* invia un comando di linea, il *Butler* risponde con il risultato del comando eseguito in formato .txt. In entrambi i casi

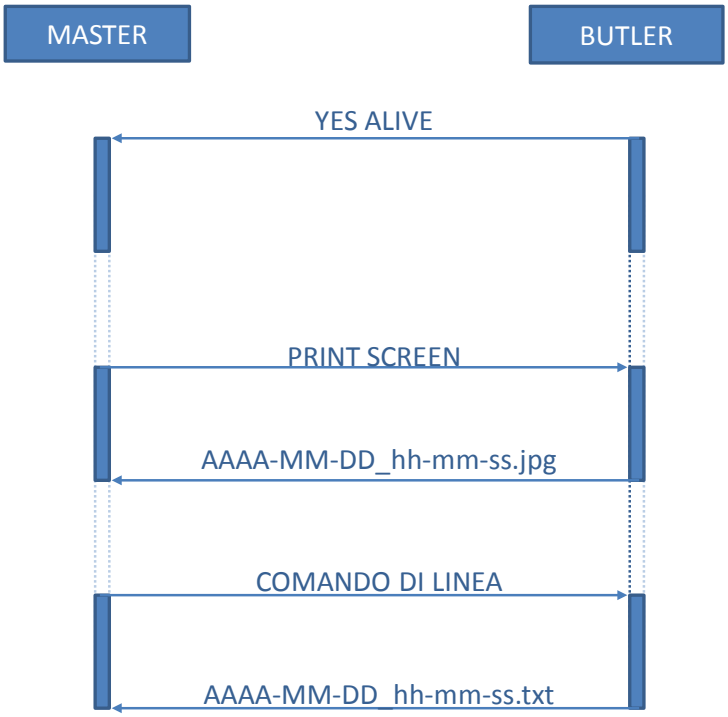

Figura 7 Corsie

2.3.3 Schema delle classi

La figura 8 mostra le classi implementate nel progetto e le loro relazioni.

Il fulcro dell'applicazione è il pannello di controllo del sistema, contenente i pannelli di comando e di ricezione comandi dei *Butler*. In ogni pannello è indicato se il relativo *Butler* è attivo o meno. Il sistema di pannelli di controllo è organizzato a schedari.

Con l'avvio della parte grafica si avviano anche altri processi: lo smistamento della posta del *Master* e lo smistamento della posta dei *Butler* istanziate dalle classi rispettivamente dalle classi *SortMail* e *SortMailB.*

TabbedPane è la classe istanziata all'avvio del programma che visualizza un pannello organizzato a schede, ognuna appartenente ad un *Butler*. Lo stato di configurazione indicante le credenziali per l'accesso a *Gmail* viene letto dal file *master.ini*. In ogni

scheda vi sono i comandi da inviare ai *Butler* e anche il pulsante per la visualizzazione dei risultati gestiti dalla classe *UpDate*.

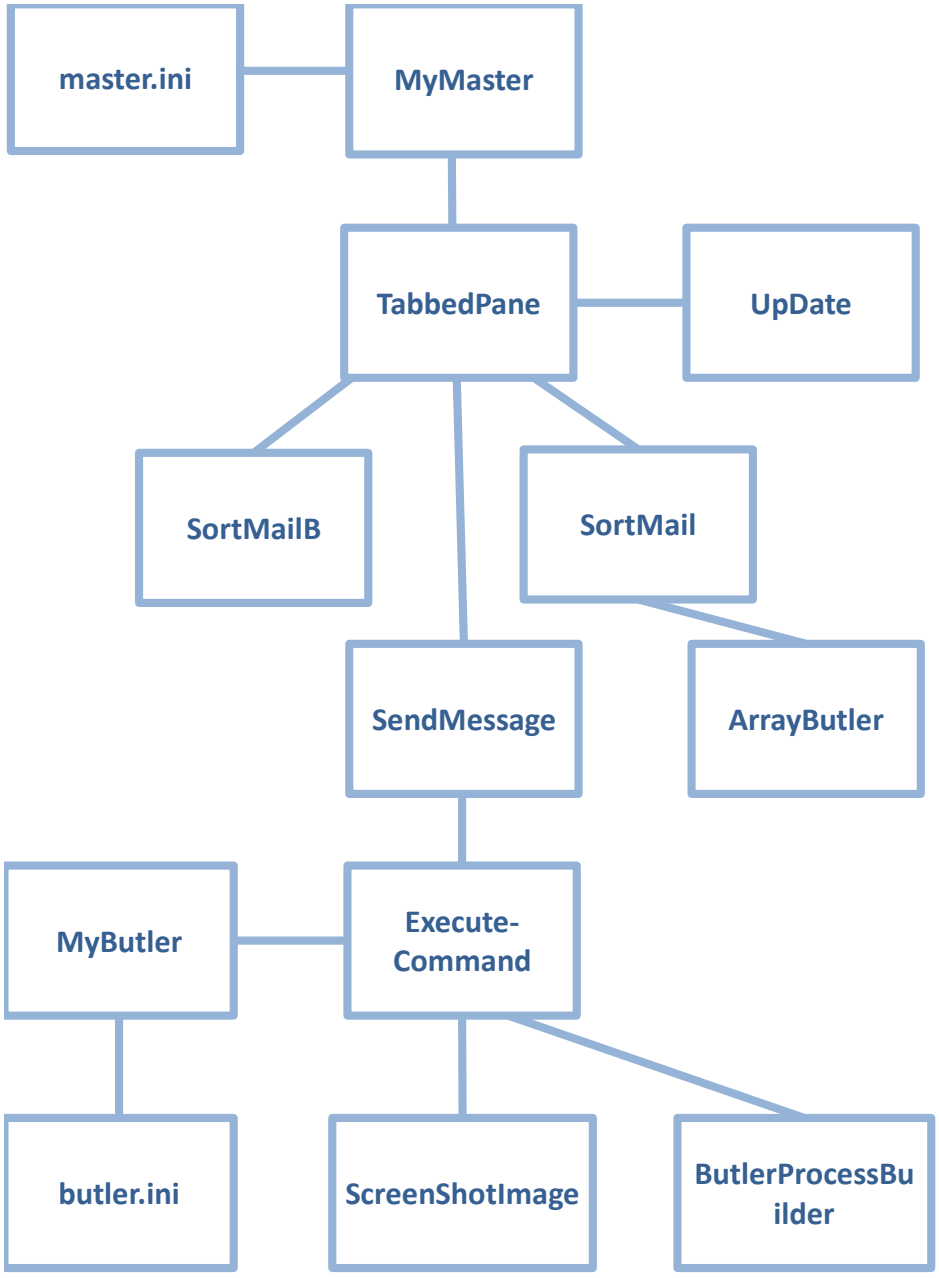

Figura 8 Diagramma delle classi

Per i *Butler*, le classi coinvolte sono: *MyButler*, *ExecuteCommand, ScreenShotImage, ButlerProcessBuilder* e *SendMessage*. La prima classe, dopo aver prelevato le credenziali dal file *butler.ini*, contente il nome del *Butler* (*Butler1*, *Butler2*, ecc..), nome utente e password per l'accesso al server di posta *Gmail*, ciclicamente invia al *Master* lo

stato di attivazione, per fargli capire che è attivo e può eseguire i suoi comandi e successivamente controlla se ci sono e-mail nella cartella di posta elettronica a lui destinata, attività che viene svolta istanziando la classe ExecuteCommand; in tal caso legge l'e-mail, decifra il commando, lo esegue richiamando le classi ScreenShotImage o ButlerProcessBuilder e rispedisce al *Master* il risultato. *SendMessage* è utilizzata per spedire un messaggio al *Master* sotto forma di e-mail.

Oltre la classe *ExecuteCommand*, *SendMessage* è usata da *TabbedPane* per mandare i comandi ai *Butler*.

Il *Master* è composto dalle seguenti classi: *MyMaster*, *TabbedPane*, *UpDate*, *SortMail*, *SortMailB*, *ArrayButler* e *SendMessage*. La classe *MyMaster*, dopo aver letto il file di inizializzazione istanzia la classe *TabbedPane*, il quale si occupa dell'interfaccia grafica, che a sua volta usa la classe *SortMailB* per smistare la posta del *Butler*, *SortMail* per smistare la posta del *Master*, *UpDate* per processare i risultati del *Butler*.

2.3.4 Dettaglio delle classi

Seguono i diagrammi delle classi, indicanti gli attribute istanziati, i metodi delle classi. Sono rappresentati in rettangoli e ogni rettangolo è diviso in tre parti: il primo indica il nome della classe, il secondo gli attributi e il loro tipo, infine ci sono i metodi con la loro visibilità: + pubblica, - private, # protetta, ~package.

MyButler
ec: ExecuteCommand time: int
+ execute()

ExecuteCommand
inbox: Folder other: Folder pw: String user: String server: String whoIAm: String time: int

+ mailReader() + printEnvelope()

ScreenShotImage
when: String whoIAm: String
+run()

ButlerProcessBuilder
when: String whoIAm: String nfile: String
+run()

SendMessage
user: String password: String
+codeAssign()

MyMaster
ar: ArrayButler jfrm: JFrame s: Thread sb: Thread

+ execute ()

TabbedPane
jtp: JTabbedPane lnum: Jlabel ta: JTextArea buttonsc: Jbutton pj: JPanel taupdate: JTextArea buttonupdate: JButton labelimm: Jlabel buttonupdate: JButton imicon: ImageIcon up: UpDate tup: Thread
+addTabb() +actionPerformed()

SortMail
b: String[] inbox: Folder store: Store ar: ArrayButler jfrm: JFrame tp: TabbedPane
+goMail() +sort() +resetMail() +contains() +codeVerify() +run()

SortMailB
b: String[] inbox Folder store Store
+goMail() +resetMail() +sort() +contains() +codeVerify()

UpDate
inbox: Folder store: Store ta: JTextArea labelimm: JLabel numButler: int who: String
+inizialization() +run()

ArrayButler
arrayB: int[]
+setValue() +getValue() +getLength()

2.4 La programmazione concorrente in Java

La soluzione più diffusa al problema della programmazione di più attività concorrenti sta nella scomposizione delle operazioni in linee di comando che rappresentano singole attività sequenziali.

Nell'ambito di ciascuna linea di esecuzione, modellabile attraverso un algoritmo sequenziale, l'ordinamento delle operazioni è fissato in modo univoco dall'algoritmo stesso; ma non è fissato l'ordinamento tra le operazioni appartenenti a linee di esecuzione diverse, ma ciò è irrilevante al fine della corretta esecuzione delle singole attività.

Nel linguaggio di programmazione Java, il concetto di linea di esecuzione è realizzato dalla classe *Thread* [JavTh]. Un oggetto appartenente a una sottoclasse di Thread è in grado di evolvere autonomamente dagli altri thread.

2.4.1 Le risorse condivise

Quando si parla di programmazione concorrente bisogna introdurre il problema delle risorse condivise e quando si parla di risorse condivise non si può non parlare del problema dei cinque filosofi, adatto a introdurre il problema della sincronizzazione delle risorse condivise. Fu formulato proprio a tale scopo da Edsger Dijkstra nel 1965 [DIJS].

Cinque filosofi si raccolgono intorno ad una tavola rotonda, hanno a disposizione una ciotola di riso ciascuno (potenzialmente infinita nella quantità, quindi il riso non termina mai) e cinque bacchette ciascuna posizionata tra due piatti.

Non per mera pignoleria, ma per una ricerca di chiarezza, si puntualizza che Dijkstra formula il problema parlando di spaghetti e di forchette, oggi, a questa versione, è affiancato il problema in cui gli spaghetti e le forchette sono sostituite dal riso e dalle bacchette.

I filosofi hanno un piatto davanti a loro, una bacchetta alla loro destra e una bacchetta alla loro sinistra. Essi pensano in modo solitario e quando hanno fame prendono una alla volta le bacchette ai lati del piatto e mangiano, finito di mangiare ripongono le bacchette e continuano a pensare e così via. Se il filosofo prende una bacchetta e si accorge che l'altra è impegnata, aspetta che si liberi, dopodiché la prende e può iniziare a mangiare. Se tutti hanno una bacchetta in mano aspettano invano che si liberi l'altra bacchetta, in questo caso di parla di *deadlock*, ossia si ha lo stallo del sistema,

mentre si parla di *starvation* quando un processo non riesce ad ottenere la risorsa necessaria per continuare la sua attività; il filosofo aspetterebbe invano di mangiare poiché non riesce a prendere le bacchette.

Inoltre ad una risorsa bisogna accedere in maniera mutuamente esclusiva per mantenere la coerenza del sistema.

La risorsa condivisa in *Master and Butler* è il gestore dei *Butler* attivi, un *array* che indica quali *butler* sono funzionanti e implementato dalla classe *ArrayButler. SortMail* accede a questa risorsa per conoscere se un dato *butler* sia attivo o meno; l'accesso deve essere mutuamente esclusivo per evitare incoerenze; a tale scopo Java utilizza il costrutto *synchronized*. Di seguito sono mostrati i metodi di lettura *setValue* e scrittura *getValue* dell'array

```
public synchronized void setValue(int val, int pos){
    arrayB[pos]=val;
}

public synchronized int getValue(int pos){
    System.out.println(arrayB[pos]);
    return arrayB[pos];
}
```

2.4.2 I listener

I *listener*, o meglio gli ascoltatori, sono elementi di particolare importanza nella programmazione concorrente e in particolare nell'implementazione delle interfacce grafiche, poiché intercettano l'interazione con l'utente. Essi vanno in esecuzione quando l'evento per il quale è in ascolto si è verificato.

In *Master and Butler* i listener sono i bottoni che permettono di interagire con i *Butler* inviando loro i comandi e visualizzando i risultati.

Solo il *Master* è dotato dell'interfaccia grafica, in quanto l'operatore interagisce solo sul *Master* attraverso l'interfaccia grafica.

I *Butler* lavorano autonomamente, senza l'intervento dell'operatore, quindi non hanno i *listener* operanti nell'interfaccia grafica.

2.5 Programmazione distribuita

La caratteristica di *Master and Butler*, oltre a essere basata sulla creazione di un protocollo basato su *Gmail*, è quella di essere un sistema distribuito, basato quindi sulla programmazione concorrente.

In un sistema distribuito due o più componenti cooperano tra loro, distanzianti anche tra loro, che si coordinano con lo scambio di messaggi.

Anche in *Master and Butler* ogni componente, pur mantenendo il proprio ruolo, opera su attività interconnesse tra loro:

- In ogni scheda dei *Butler* ci sono listener, di cui abbiamo parlato nel paragrafo precedente, fanno avviare la procedura per l'invio dei comandi, e la visualizzazione dei risultati.
- Il *thread* smistatore del *Butler* sposta le e-mail in arrivo nelle cartelle relative di ogni *Butler* destinatario
- Ogni *Butler* ciclicamente invia il segnale di stato attivo
- Ogni *Butler* legge il comando, lo esegue e restituisce il risultato
- Il *thread* smistatore del *Master* smista la posta in arrivo del *Master*

Capitolo 3
I Butler

"Rey: - Adesso tu mi libererai e lascerai la cella con la porta aperta.
Assaltatore: - Adesso io ti libererò e lascerò la cella con la porta aperta.
Rey: - E getterai la tua arma.
Assaltatore: - E getterò la mia arma."[3]

L'obbedienza è fondamentale in certe situazioni. Per fare bene il proprio lavoro a volte occorre semplicemente eseguire gli ordini. In *Master and Butler*, ogni *Butler*, come un bravo maggiordomo, riceve il comando, lo esegue e restituisce al *Master* il risultato prodotto.

In questo capitolo si parlerà di come è strutturato un *Butler*, la sua implementazione e l'interazione con il *Master*.

3.1 Funzionamento dei Butler

I *Butler*, ognuno in modo indipendete dall'altro, compiono semplici operazioni cicliche. Per essere inizializzati, leggono le proprie credenziali dal file *butler.ini* contenente il proprio nome (es. *Butler1*) e le credenziali per accedere all'account di *Google Mail*: username e password.

I *Butler* hanno un unico account di posta elettronica *Gmail* e lo condividono tra di essi, senza intralciarsi, accedendo alla propria cartella dove possono trovare le e-mail a loro destinate. Questo processo è eseguito dallo smistatore dei *Butler.*

Dopodiché, mandano ciclicamente un messaggio, al *Master,* di stato attivo, e controllano, sempre ciclicamente, accedendo a Gmail, nella cartella a loro destinata, se hanno dei nuovi messaggi inviati dal *Master*.

In tal caso, viene decifrato il comando da eseguire e la data dell'esecuzione indicate nel corpo dell'e-mail.

[3]Dialogo tra Rey e un assaltatore nel film "*Star Wars: The Force Awakens*" USA 2015.

```
Butler1
2017-04-06_18-00-00
Print Screen
```

Questo è un esempio di un messaggio inviato a *Butler1*, che, alle ore 18 del 6 aprile 2017, deve effetturare una stampa dello stato del monitor.

Dopo aver isolato le informazioni e salvato su un file il comando da eseguire, quest'ultima operazione solo per le operazioni da svolgere dal *promt* del DOS, il *Butler* lancerà il *thread* relativo al comando desiderato e ritornerà alla routine di operazioni da compiere: messaggio stato attivo/check nuovi comandi da eseguire.

Il nome del file dove sono contenuti i comandi da eseguire è dato dalla data di salvataggio dello stesso. Il fatto che nessun file è salvato nello stesso istante, poichè viene salvato solo alla fine della lettura dell'e-mail e questo è sicuramente maggiore di un second, rende univocal l'associazione nome file – commando da eseguire.

Il controllo sul mittente e sull'oggetto delle e-mail non viene effuetto dai *Butler* in quanto già precedentemente eseguito dallo smistatore delle e-mail del *Butler* al momento di spostare le e-mail nelle cartelle giuste.

3.2 I comandi

I comandi eseguiti sono operazioni indipendenti dal lavoro che compie il Butler e indipendeti con gli altri comandi eseguiti dal Butler, per questo sono stati implementati dai *Thread* di Java, in modo da poter essere eseguiti parallelamenti e non in modo sequenziale.

Ogni *Thread* di esecuzione del comando ha come parametro di ingresso la data di esecuzione dello stesso, il nome del *Butler* che lo esegue e, per il solo commando di esecuzione istruzioni da DOS, il nome del file dei comandi.

Il *Thread* di esecuzione si mette in pausa finchè non è trascorso il tempo necessario ad arrivare alla data programmata; quindi esegue l'istruzione ricevuta e rimanda al Master il risultato allegandolo per e-mail in formato testo o immagine, a seconda dell'istruzione.

Il *Thread* si mette in pausa con l'istruzione

```
try  {
     Thread.sleep(time);
}catch (InterruptedException exc){}
```

Dove time è il tempo espresso in millisecondi.

3.2.1 Print Screen

Per fare il printscreen del monitor, viene utilizzata la classe *java.awt.Robot*. Istanziando *Robot()* si crea un oggetto che usufruisce del dispositivo di schermo principale, di default, per compiere i suoi metodi.

Un oggetto *Robot*, al quale è applicato il metodo *createScreenCapture*, crea, o meglio mette in un buffer, un'immagine contenente i pixel letti dallo schermo; il parametro passato nel metodo citato è un oggeto *Rectangle* che gestisce i pixel da leggere.

```
SimpleDateFormat sdfDate = new SimpleDateFormat("yyyy-MM-
dd_HH-mm-ss");
Date now = new Date();
String strDate = sdfDate.format(now)+".jpg";
Robot robot = new Robot();
BufferedImage screenShot = robot.createScreenCapture(new
Rectangle(Toolkit.getDefaultToolkit().getScreenSize()));
ImageIO.write(screenShot, "JPG", new File(strDate));
```

3.2.2 ProcessBuider

Gli altri comandi presenti: *ping*, *ipconfig* e *dir*: si servono della classe *ProcessBuilder*.

Questa classe ha un utilizzo molto semplice; oltre al costruttore, ha bisogno del metodo start() per farla avviare, poi si precisa il file di ingresso dei comandi, il file degli eventuali errori e il file di uscita dove vengono scritti i risultati del comando eseguito.

```
Date now2 = new Date();
String strDate = sdfDate.format(now2)+".txt";

try {
     File commands = new File(nfile);
     File output = new File(strDate);
     File errors = new File("ErrorLog.txt");
     ProcessBuilder pb = new ProcessBuilder("cmd");
```

```
        System.out.println(pb.redirectInput());
        System.out.println(pb.redirectOutput());
        System.out.println(pb.redirectError());

        pb.redirectInput(commands);
        pb.redirectError(errors);
        pb.redirectOutput(output);

        System.out.println(pb.redirectInput());
        System.out.println(pb.redirectOutput());
        System.out.println(pb.redirectError());

        pb.start();

    }catch(IOException io){}
```

3.2 Interazione con il Master

I *Butler* interagiscono con il *Master* in maniera asincrona, sono ignari di quando il Master invia il comando. Controllano nella cartella di ricezione della posta elettronica, verificano quando devono eseguire il comando, decifrano il tipo di commando, lo eseguono, scrivono il risultato in un file con estensione: *.jpg*, se il comando è un *Print Screen*, *.txt* per gli altri comandi; per tutti i casi, se il comando è ricevuto in ritardo rispetto alla data di richiesta esecuzione, viene scritto un file *.txt* con il messaggio

> *"ButlerN* nome_*Comando* non eseguito in data *aaa-mm-gg_hh-mm-ss"*

Per tutti i comandi, il nome del file prodotto è la data di esecuzione effettiva del comando o la data di trattazione del comando se poi non verrà eseguito. Ad esempio, per i file testo il nome file è calcolato nel seguente modo:

```
Date date= new Date();
SimpleDateFormat sdfDate = new SimpleDateFormat("yyyy-MM-
dd_HH-mm-ss");
String strDate1 = sdfDate.format(now)+".txt";
```

Al termine dell'esecuzione il file viene spedito al *Master*, allegandolo alla e-mail.

3.3 Classi Java utilizzate

Ogni *Butler* utilizza le classi *MyButler* per essere avviato, *ExecuteCommand,* per mandare un messaggio di stato attivo al Master, per leggere se ci sono nuovi messaggi del *Master*, eventualmente leggerli uno ad uno, capire di che comando si tratta e lanciare il Thread del comando relativo. Ciò viene ripetuto ogni dieci secondi.

I *Thread ButlerProcessBuilder* e *ScreenShotImage* vengono istanziate per eseguire un comando singolo. Ogni *Thread* istanziato è autonomo dagli altri e dalla classe che li ha generati, perchè il comando potrebbe essere eseguito non immediatamente, quindi il Thread si pone in stato di pausa finchè non è processato.

Per tutti i comandi viene elaborato un file di con il risultato dei comandi prodotti. Nel caso di *Screen Shot* il file è di tipo immagine, nel caso di comando dal Promt di DOS il tipo di file è un testo; anche nel caso in cui il comando è stato letto fuori tempo di esecuzione e quindi non può essere eseguito, viene prodotto un file di testo contenente la motivazione.

I file prodotti vengono spediti al *Master* in allegato alla e-mail contenente come oggetto il codice dispositivo e nel corpo il nome del Butler che ha inviato il messaggio.

Capitolo 4
Il Master

"Con il sostegno della Repubblica,
il generale Leia Organa guida
una coraggiosa Resistenza." [4]

Per far si che il sistema sia ben funzionante ci deve essere un buon leader che sappia gestire e raggiungere l'obiettivo con successo. Ciò è valido sia nella società umana, sia nel mondo animale, sia nei sistemi informatici e di comunicazione.

Nel presente sistema il Master coordina le operazioni: avvia la comunicazione, la ferma, resetta il sistema, invia i comandi ad ogni Butler tramite pannello di controllo, raccoglie i risultati ottenuti e li visualizza sempre nel pannello di controllo.

Nel presente capitolo verrà analizzato il funzionamento del Master, come è fatta l'interfaccia operativa, come interagisce con i Butler, ed infine saranno trattati i Thread implicati nella programmazione concorrente, indipendenti tra loro ed operano per inviare i comandi e raccogliere le risposte dei Butler.

4.1 Funzionamento del Master

Il *Master* comanda e coordina l'attività dei Butler. Invia il comando da far eseguire ai *Butler* e ne visualizza l'esecuzione.

Dopo aver ricevuto le credenziali di accesso alla casella di posta elettronica Gmail, tramite il file di inizializzazione *master.ini*, il *Master* avvia l'interfaccia grafica di comando/controllo e gli smistatori di posta: quello del *Master* e quello del *Butler*, che riordinano ed organizzano i messaggi spostandoli nelle cartelle mittenti o destinatari presenti nelle e-mail. All'interno dell'interfaccia grafica è anche possibile visualizzare il risultato dell'esecuzione. Ogni *Butler* ha una scheda indipendente.

Si può inviare dei comandi anche se i *Butler* non sono attivi, questo sistema è stato studiato apposta per schedulare i processi in modo asincrono. Quando il *Butler* riceve il comando, se deve eseguirlo subito, lo esegue immediatamente, se deve eseguirlo dopo un certo periodo aspetta quel periodo, se il momento di esecuzione è già passato, non lo esegue più.

[4] Dal Film "*Star Wars: The Force Awakens*" USA 2015.

Poi, il *Master* può avviare i *thread* di verifica comandi, uno per ogni *Butler*, che visualizza il risultato dei comandi prodotti dai *Butler*.

4.2 L'interfaccia utilizzata

Per la realizzazione del pannello di controllo è stata utilizzata con le librerie *Swing* di Java [JavSw].

Java Swing fa parte della *Java Foundation Classes* (*JFG*) e utilizzate nello sviluppo di interface grafiche. Gli oggetti che fanno parte del *framework Swing* sono l'implementazione di oggetti grafici come pulsanti, caselle di testo, pannelli, e altro.

A differenza delle classi *awt* (*Abstract Window Toolkit*), anch'essa impiegata nelle interface grafiche, *Swing* è scritto completamente con codice Java e ne è una sua estensione in quanto tutti i componenti top-level in *Swing* estendono i contenitori *top-level-AWT*.

Java Swing è una libreria indipendente dalla piattaforma, poichè utilizza Java e anche perchè la renderizzazione dei widget è universale e non nativa.

Java Swing è estendibile le sue interfacce permettono l'estensione di altre interfacce da queste.

Java Swing è orientato ai componenti. Un componente è un oggetto con caratteristiche note ed attivano eventi in modo asincrono. Le caratteristiche degli oggetti sono legate agli eventi e rispondono in un determinato modo al comando inviato.

Java Swing è modificabile. I componenti sono personabilizzabile come il colore, sfondo, stile ed altro.

Java Swing è configurabile. I componenti cambiano la visualizzazione a seconda del Sistema operativo.

Java Swing è leggera. Un componente Swing può non avere il rispettivo nell'insieme dei componenti del sistema operativo nel quale è eseguito.

Java Swing usa l'architettura *Model-View-Controller* (*MVC*) [ECKS] all'interno di ogni suo componente. Consiste nello scomporre il componente grafico in tre elementi, ciascuno dei quali svolge un ruolo fondamentale nelle caratteristiche del componente.

- *Model*: rappresenta i dati e le regole che regolano l'accesso e l'aggiornamento dei componenti
- *View*: la vista dà il contenuto di un modello. Essa specifica esattamente come i dati del modello dovrebbero essere presentati. Se vi sono modifiche nei dati del modello, la vista deve aggiornare la sua presentazione in base alle esigenze. Ciò può essere ottenuto utilizzando un modello *push*, in cui la vista si registra con il modello per notifiche di modifica, o un modello *pull*, in cui la vista è responsabile della chiamata modello quando è necessario recuperare i dati più attuali.
- *Controller*: Il controller traduce le interazioni dell'utente con la vista in azioni che il modello si esibiranno. In un client GUI stand-alone, le interazioni degli utenti potrebbero essere clic pulsante o selezioni di menu, mentre in un'applicazione web enterprise, essi appaiono come GET e richieste HTTP POST. A seconda del contesto, un controller può anche selezionare una nuova vista - per esempio, una pagina web dei risultati - presentare indietro per l'utente.

4.2.3 Rappresentazione a schedario dei Butler

Il pannello di controllo dei *Butler* è suddiviso a schedario. Ogni *Butler* ha il suo schedario e ogni schedario contiene la visualizzazione dello stato del *Butler*, un menù a tendina dove scegliere il comando da inviare, la *textarea* per inserire la data di esecuzione, un *button* per avviare la visualizzazione dei risultati dei comandi, una *textarea* dove visualizzare i file mandato dal *Butler* e una area per visualizzare le immagini risultato del lavoro del *Butler*.

La classe *TabbedPane* eredita tutti i metodi e attibuti della classe padre: *JFrame*, dichiarata tale con in costrutto *extends*, mentre implementa i metodi della classe *ActionListener*, poiché quest'ultima è un'interfaccia per *TabbedPane*, dichiarata tale dal costrutto implements.

```
public class TabbedPane extends JFrame implements
ActionListener
```

All'avvio del *Master* vengono istanziati tanti panneli, quanti sono i *Butler* previsti nel sistema, ad esempio quattro.

```
this.jfrm=jfrm;
tp = new TabbedPane();
tp.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
tp.setVisible(true);
tp.addTab(1,"Butler1");
tp.addTab(2,"Butler2");
tp.addTab(3,"Butler3");
tp.addTab(4,"Butler4");
```

Ogni pannello è indipendente in quanto istaziato per ogni Butler.

4.3 Interazione con i Butler

Il *Master* interagisce in modo asincrono con i *Butler*. Al suo avvio istanzia lo smistatore dei *Butler*, lo smistatore del *Master* e il pannello di controllo, indipendentemente dallo stato dei *Butler*.

Per controllare se i *Butler* siano attivi o meno, il *Master* controlla la data di invio della e-mail inerente lo stato attivo del *Butler*. Se è precedente ai cinque minuti rispetto l'orario attuale allora è considerato inattivo, altrimenti è marcato come attivo e segnalato nel pannello di controllo relativo al *Butler* in questione.

Occorre apporre una nota molto importante: la data di invio delle e-mail è impostata dal server di posta elettronica, in questo caso Gmail, mentre la data attuale è data dall'impostazione del sistema operativo. Quindi è necessario controllare che l'orologio di sistema in cui è in esecuzione il *Master*, abbia orario esatto.

Il protocollo non pone il focus sulla sincronia dello scambio dei messaggi, ma sulla corretta schedulazione dei comandi. L'assenza di sincronizzazione tra le parti non comporta un caos tra le azioni: nell'invio dei comandi è sempre presente l'ora di esecuzione del comando, nella ricezione delle risposte è sempre presente la data di esecuzione del comando.

4.5 I Thread Java nel Master

I *Thread* in Java sono oggetti particolari ai quali si richiede l'esecuzione di un'attività e si avviano con *start(),* inoltre non si aspetta che l'attività sia terminata: è per questa

caratteristica quindi che si dicono concorrenti: il *Thread* procede affiancando chi ha richiesto il servizio ed affiancando anche gltri eventuali *Thread*.

Nel *Master* i Thread sono lo smistatore del *Butler*, lo smistatore del *Master* e il *Thread* delle interfacce grafiche di aggiornamento dei pannelli a schedario dei *Butler*, chiamati *UpDate*, attivi su comando dell'operatore tramite l'interfaccia grafica, uno per ogni *Butler*.

I Thread implementano tutti l'interfaccia Runnable e non ereditano direttamente dalla classe Thread. Java non permette l'ereditarietà multipla e le interfacce permettono di ovviare a questa limitazione.

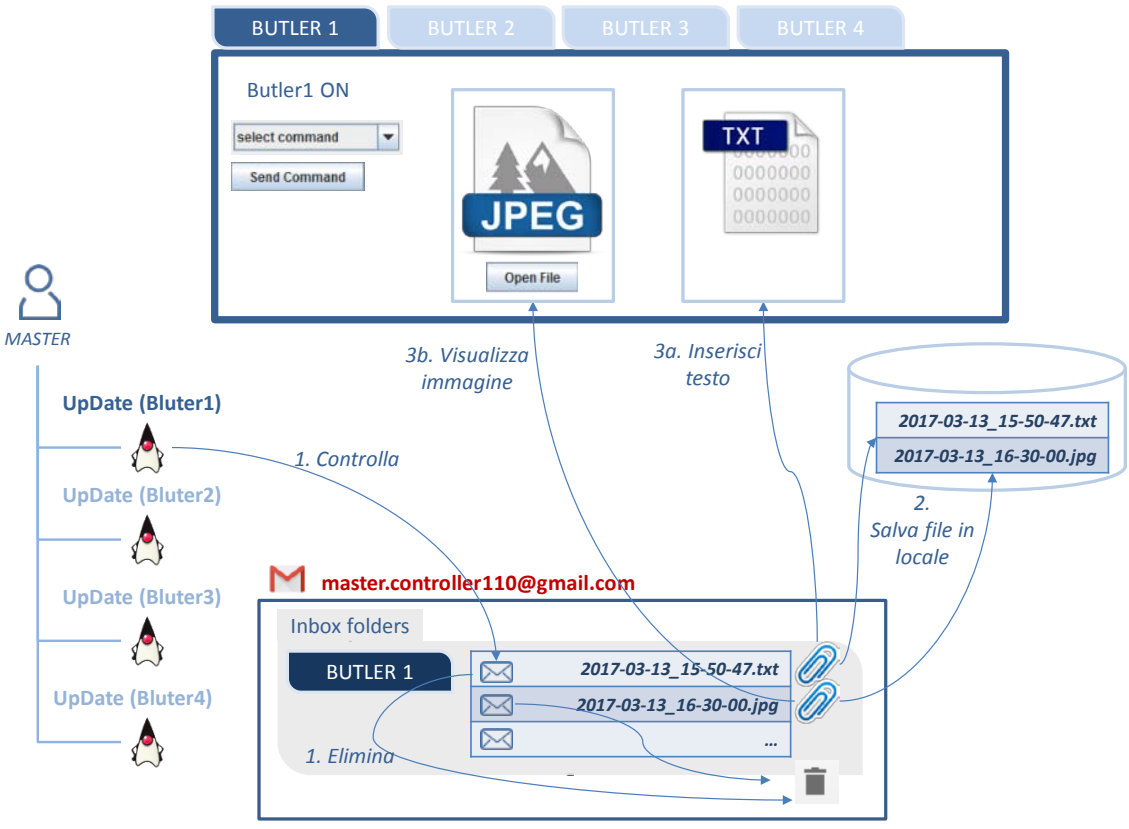

Figura 9 Funzionamento Thread UpDate

In figura 9 è illustrato il funzionamento di *Thread UpDate*, in questo caso il *Thread* lavora su *Butler1*.

I *Thread* si attivano con un pulsante nel pannello di controllo del *Butler*. Il suo funzionamento è il seguente:

1. Controlla se ci sono messaggi nella casella di posta elettronica del *Butler*, nella cartella *Butler1*
2. Se si, prende l'allegato della e-mail e lo salva in locale in uno spazio dedicato

3.a Se è in formato testo lo accoda al testo già presente nella textarea del pannello di controllo di Butler1

3.b Se è un'immagine la visualizza nella sezione immagine

4. Infine il messaggio viene eliminato.

I file di testo, che contengono i risultati delle istruzioni in linea dei *Butler*, vengono stampati sotto gli altri comandi, nell'interfaccia grafica. Le immagini, che contengono il *Print Screen*, sostituiscono le immagini presenti.

Se si volesse rivedere un'immagine passata o analizzare il testo di un messaggio, quando questi non sono più presenti nell'interfaccia grafica, si va a rivederli senza difficoltà in quanto tutto è salvato in locale e addirittura si può sempre consultare la posta inviata dal *Butler*.

4.5.1 Le classi utilizzate

Riepilogando, le classi che usa il *Master* per la sua attività, sono:

- *MyMaster*, per avviare la sua attività
- *TabbedPane*, per avviare l'interfaccia grafica di controllo dei *Butler*
- *SortMail,* per avviare lo smistatore della casella di posta elettronica del *Master*
- *SortMailB,* per avviare lo smistatore della casella di posta elettronica del *Butler*
- *UpDate,* per visualizzare i risultati dei comandi eseguiti dai *Butler*, nell'interfaccia grafica
- *ArrayButler,* per tenere aggiornata la lista dei Butler attivi
- *SendMessage,* per mandare i messaggi ai Butler (sotto forma di e-mail) .

A completamente di queste classi, i file master.ini e file.txt permettono il corretto funzionamento: il primo è necessario per l'inizializzazione del sistema con i dati del *Master*, il secondo contiene i codici dispositivi.

Capitolo 5
Conclusioni

"No! Provare no. Fare! O non fare. Non c'è provare!"[5]

Master and Butler è sistema di controllo distribuito, il cui protocollo di comunicazione è basato su scambio di messaggi di tipo e-mail, servendosi del servizio di posta Gmail.

Master and Butler è un sistema affidabile, poichè la risposta ad un comando avviene in ogni caso, magari è una risposta di mancata esecuzione, ma sempre una risposta. L'inaffidabilità è legata al *Butler* qualora sia inattivo.

Master and Butler è anche un sistema pensato alla schedulazione dei comandi da far eseguire in remoto, quindi il tempo impiegato alla consegna del messaggio è ininfluente e si adatta perfettamente ai tempi legati alla consegna di un'e-mail come messaggio di posta elettronica.

L'idea di sfruttare *Gmail* come base del protocollo di comunicazione proviene da un virus informatico: *GCat*. Le porte IMAP e POP3 sono facilmente accessibili in reti con politiche di restrizione, inoltre, le specifiche con cui si compone un messaggio di tipo elettronico, offrono la base su cui sviluppare il protocollo.

Sebbene *GCat* abbia fatto una scelta astuta nell'utilizzare messaggi di posta elettronica nel protocollo, non è il solo: *Cisco*, nel suo *Cisco IOS* utilizza dei moduli configurabili, tra i quali *Call Home* [CIS1] e *Embedded Event Manager EEM* [CIS2], i quali provvedono ad inviare e-mail contenente infomazioni sulla diagnostica, condizioni ambientali, inventario e eventi del *syslog*.

Il lavoro svolto si è basato sullo studio per lo sviluppo di un protocollo adeguato e la realizzazione del sistema distribuito.

Master and Butler è stato sviluppato con il linguaggio di programmazione Java 1.8, ed è composto da un'interfaccia grafica che coordina i Butler, fa eseguire loro i comandi e visualizza i risultati da loro prodotti.

Gli organi principali del sistema sono il *Master* e il *Butler*, due entità distinte ognuna con i suoi compiti e le sue funzioni. Il *Master* invia comandi, gestisce i messaggi sia del *Butler* sia del *Master*, gestisce lo stato dei *Butler* e visualizza le risposte dei *Butler*. I

[5]Yoda a Luke nel film "*Star Wars: The Empire Strikes Back*". USA 1980.

Butler si occupano di inviare il proprio stato di attività, esegue i comandi e restituisce il risultato al *Master*.

I componenti del *Master* sono: lo smistatore di posta del *Master*, che si occupa di organizzare le e-mail in base al mittente; lo smistatore di posta del *Butler*, che si occupa di organizzare le e-mail in base al destinatario; il gestore dell'interfaccia grafica, che si occupa di inviare i comandi ai Butler ed infine il visulizzatore dei risultati dei comandi, che si occupa di inviare all'interfaccia grafica i risultati dei comandi elaborate dai *Butler*.

I *Butler* sono composti dal gestore delle attività dei *Butler* stessi, che ciclicamente invia lo stato di attività, legge i comandi, lancia il gestore del comando che esegue il comando nel tempo previsto e restituisce il risultato al *Master*.

5.1 Suggerimenti per sviluppi futuri

L'applicativo realizzato è incentrato sul protocollo e sulla programmazione concorrente e distribuita. Non essendo stato prodotto per particolari scopi, il tipo di interazione tra *Master* e i *Butler* ha carattere dimostrativo.

Per eventuali sviluppi future si possono dare suggerimenti sul campo di applicazione e quindi sul tipo di operazioni che i *Butler* possono eseguire:

- *Master and Butler* si presta nel controllo di *Cluster* come ad esempio operazioni di monitoraggio degli stessi, controllo dello stato, accensione e spegnimento delle stazioni ecc… Analogamente può essere anche sviluppato per controllo dei laboratori informatici, come ad esempio controllo dello stato, aggiornamento e monitoraggio delle postazioni.
- Un altro campo di applicazione, all'interno dei laboratori informatici, può essere quello di supervisione del lavoro degli sudenti, per controllo del lavoro in sede di esame e svolgimento esecitazioni.
- Si può usare *Master and Butler* nella domotica: avviamento lavatrici, riscaldamento ed altri elettrodomesici e il loro monitoraggio.
- Infine un'altra applicazione di *Master and Butler* suggerita è la sorveglianza di abitazioni, edifici ed animali domestici.
- *ProcessBuilder* è una classe che supporta l'esecuzione di comandi di linea sia per Windows sia per altri sistemi operativi: Macintosh e Linux. I comandi quindi possono essere impostati per le varie esigenze.

Un’altra funzione che si potrebbe aggiungere al sistema è aumentare il livello si sicurezza introducendo dei codici dispositivi che si rigenerano in modo automatico, o/e intruducendo altre variabili per la validità, ad esempio possono esserci più file contenente i codici dispositivi, ma quello da prendere in considerazione è scelto in base al giorno, o un’altra variabile.

Bibliografia

[CIS1] CiscoSystem, *Call Home Configuration Guide*, http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst2k3k_smart_call_home/release/12-2_58_se/Call_Home_Config_Guide.html

[CIS2] F. Semperboni, *Send email from your router using EEM* http://www.ciscozine.com/send-email-from-your-router-using-eem/

[DIJS] H. Per Brinch, Operating System Principles, ed. PRENTICE-HALL, INC., Englewood Cliffs, New Jersey, 1973

[ECKS] R. Eckstein, Java SE Application Design With MVC http://www.oracle.com/technetwork/articles/javase/index-142890.html, marzo 2007

[GCa] byt3bl33d3r, *GCat*, https://github.com/byt3bl33d3r/gcat

[GuiG] Google Inc., *Guida di Gmail*, https://support.google.com/mail/answer/7126229?hl=it

[Java] Oracle Corporation, *Java SE Development Kit 8*, http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

[JavM] Oracle Corporation, *JavaMail* http://www.oracle.com/technetwork/java/javamail/index.html

[JavSw]M. Loy, R. Eckstein, D. Wood, J. Elliott, D. Cole, *Java Swing*, O' Reilly, 2002

[JavTh]S. Oaks, H. Wong, *Java Threads, 3rd Edition*, O' Reilly, 2004

[KuKe] J. F. Kurose, W. Keith, Computer Networking: A Top-Down Approach Featuring the Internet, 4th edition Pearson Education Inc, publishing as Addison-Wesley, 2008

[UML] L. Vetti Tagliati, *UML e ingegneria de software: dalla teoria alla pratica*,Ed. Tecniche Nuove, 931 pagine, 2003