

ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Seconda Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

P4P: INFRASTRUTTURE SPACE-BASED
PER IL P2P

Elaborata nel corso di: Sistemi Distribuiti

Tesi di Laurea di:
GIORGIO AMADEI

Relatore:
Prof. ANDREA OMICINI

Correlatori:
Prof. ALESSANDRO RICCI

ANNO ACCADEMICO 2005–2006
SESSIONE III

PAROLE CHIAVE

Cooperazione

p2p

Alle nuove generazioni, con l'augurio di un futuro
illuminato: libero dall'egoismo e pieno di cooperazione.

Indice

Introduzione	ix
1 Architettura	1
1.1 Gli strati dell'infrastruttura	1
1.1.1 TuCSoN e il layer di trasporto	1
1.1.2 Core	3
1.1.3 System	3
1.1.4 Service	3
1.2 TuCSoN	3
1.2.1 Tuple	4
1.2.2 Centri di tuple	4
1.2.3 Agenti	6
2 Core	7
2.1 Architettura del Core	7
2.2 SystemIO	8
2.2.1 Stato interno	10
2.2.2 Input	10
2.2.3 Output	12
2.2.4 Richieste pendenti	13
2.2.5 Comportamento	13
2.3 Agente di LookUp	15
2.3.1 Interazione con TuCSoN	15
2.3.2 Comportamento	18
2.4 RouterIO	19
2.4.1 Stato interno	19
2.4.2 Input	20

2.4.3	Output	22
2.4.4	Comportamento	23
2.5	AgentIO	28
2.5.1	Stato interno	28
2.5.2	Input	30
2.5.3	Output	34
2.5.4	Richieste pendenti	36
2.5.5	Comportamento	36
2.6	Agent	46
2.6.1	CoreAgent	46
2.6.2	AbstractAgent	46
2.6.3	ICoreApi	47
2.6.4	Comportamento	47
2.7	Meccaniche interne	47
2.7.1	End Point	48
2.7.2	Debug	48
2.7.3	Try e Catch	49
2.7.4	Indici e Next	51
2.7.5	Waiting	52
2.7.6	Invito, Protocollo di mediazione	53
2.8	Api	56
2.8.1	Precisazioni	58
3	System	59
3.1	Architettura dello strato	59
3.2	Agente Generico	60
3.3	Postman (postino)	62
3.3.1	Interazioni	62
3.4	Hostess	64
3.4.1	Interazioni	64
3.5	Detective	67
3.5.1	Interazioni	67
3.6	Operator (centralinista)	70
3.6.1	Start-Up	71
3.6.2	Interazioni	73
3.6.3	Gestione dei Gruppi	76
3.7	Helmsman (timoniere)	76

3.7.1	Interazioni	77
3.8	Worker (lavoratore)	79
3.8.1	Interazioni	80
3.8.2	Api	82
4	Service	87
4.1	Messenger	87
4.1.1	Comunicazione privata	88
4.1.2	Comunicazione di gruppo	89
4.2	TicTacToe (Tris)	90
4.2.1	Facciamo una partita	90
5	Guida all'Uso	93
5.1	Installazione	94
5.2	Log-In	94
5.3	Interfaccia principale	95
5.4	Gestione utente	95
5.5	Richiesta di conferma	96
5.6	Selezione di un contatto	96
5.7	Installare e disinstallare plugin	97
6	Guida allo Sviluppo	99
6.1	Analisi	99
6.2	Plugin di sistema	101
6.3	Plugin di servizio	103
6.4	Manifest.plugin	105
6.5	Interfacce grafiche	105
6.6	Prospettive future	106
A	Pattern decorator-delegation	109
B	Pattern decorator-polimorfico	111
	Ringraziamenti	117

Introduzione

La seguente tesi verte sull'analisi e lo sviluppo di un infrastruttura distribuita, basata su TuCSoN (vedi Sezione 1.2) e caratterizzata dall'assenza di server, dalla facilità di estensione e dalla completa portabilità. Ne verranno descritte la struttura generale e, nel dettaglio, i vari strati che la compongono; si concluderà con una breve guida allo sviluppo di plug-in.

TuCSoN è una infrastruttura open source per la coordinazione di agenti, sviluppata in Javae basata sulla rete internet. Utilizza oggetti chiamati “tuple” che sono composti da un nome seguito da una lista di parametri (`nomeTupla(par1, par2, par3(par31, par32))`), questi messaggi vengono contenuti e gestiti in artefatti di coordinazione chiamati “centri di tuple”. Tali centri possono essere programmati in modo da reagire autonomamente e possono anche essere interrogati dagli agenti che sono interessati ai messaggi che contengono.

L'infrastruttura **p4p** implementa una piattaforma distribuita per la creazione di sistemi software di cooperazione, si basa sul concetto “If you need something you must ask to who know about that” (Se ti serve qualcosa devi chiedere a chi conosce ciò che stai cercando). Un approccio simile è alla base dei newsgroup che raggruppano gli utenti in base ad argomenti di discussione essi, attraverso una bacheca, possono scambiarsi informazioni e pareri; anche i forum spesso riuniscono persone con interessi o conoscenze in comune, ma in tutti questi casi manca la possibilità di interagire in tempo reale, comunicare e cooperare attraverso strumenti più complessi della semplice corrispondenza testuale. In quest'ottica, **p4p**, si propone di aumentare le possibilità di interazione fra i componenti di un gruppo; infatti è facilmente estendibile attraverso plug-in, tali da rendere possibile “qualunque” tipo di interazione.

Capitolo 1

Architettura

L'architettura è in accordo con il modello di riferimento ISO OSI (Figura 1.1) che è suddiviso in layer (strati); alcuni di essi sono realizzati attraverso l'infrastruttura di rete, altri sono gestiti da TuCSoN[3]; i restanti verranno trattati all'interno di questo documento. Si tratta più precisamente degli strati superiori (5,6,7) lasciati a carico degli sviluppatori delle applicazioni; essi vengono implementati dall'infrastruttura **p4p**.

1.1 Gli strati dell'infrastruttura

L'infrastruttura **p4p** è suddivisa in vari sotto strati disgiunti dal modello che sono raffigurati nello schema in Figura 1.2 e sono stati progettati per essere tra loro il più possibile indipendenti. Essi espongono il minor numero di metodi nascondendo i tecnicismi degli strati inferiori, per ottenere questo effetto è stato usato un pattern particolare (Appendice A) che permette di nascondere i metodi e renderli disponibili all'interno della classe attraverso uno specifico puntatore.

1.1.1 TuCSoN e il layer di trasporto

Il layer 4 viene gestito dall'infrastruttura TuCSoN che permette la realizzazione di un sistema distribuito ad agenti basato sui protocolli di comunicazione TCP-IP e UDP-IP. Una trattazione più approfondita di questa infrastruttura verrà affrontata nella Sezione 1.2.

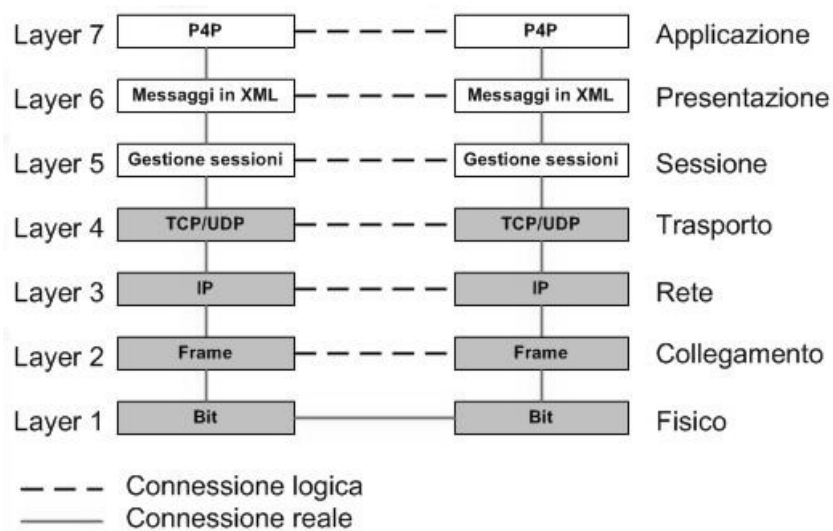
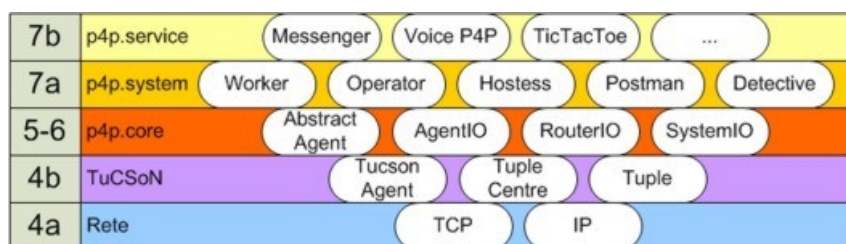


Figura 1.1: Il modello di riferimento ISO OSI

Figura 1.2: Gli strati dell'infrastruttura **p4p**

1.1.2 Core

Il Core rappresenta il cuore dell'infrastruttura **p4p** e realizza l'interazione con TuCSoN, la gestione delle sessioni, la presentazione delle informazioni e le meccaniche di indirizzamento dei messaggi. Esso utilizza massivamente i centri di tuple ed il linguaggio ReSpecT per realizzare le sessioni e l'indirizzamento dei messaggi, invece per la presentazione si appoggia sulle librerie: XStream[6] per la serializzazione in xml, JDOM[7] per la manipolazione e Xerces[8] per la validazione. (Per maggiori informazioni consultare il Capitolo 2)

1.1.3 System

Il layer di sistema contiene esclusivamente agenti preposti allo svolgimento di determinati compiti, essi non interagiscono direttamente con gli utenti ma vengono utilizzati dagli altri agenti; eccezione fa l'*Operator* che è l'unico agente di questo strato ad interagire con l'utente: coordina le procedure di login, di peer discovery e ip lookup ed infine gestisce l'interfaccia principale. (Per maggiori informazioni consultare il Capitolo 3)

1.1.4 Service

Gli agenti che appartengono a questo layer sono gli strumenti di cooperazione, possono “costruire” delle connessioni di varie topologie fra gli utenti e sono attivabili attraverso opportuni menù dell'interfaccia principale. Essi subappaltano gran parte dei loro task agli agenti del layer di sistema, sia locali che remoti e coordinano l'interazione fra gli utenti. (Per maggiori informazioni consultare il Capitolo 4)

1.2 TuCSoN

L'infrastruttura TuCSoN è un sistema distribuito ad agenti ed è realizzata utilizzando il linguaggio Java, quindi, indipendente dal sistema operativo. Permette la creazione di una rete di nodi remoti raggiungibili attraverso l'indirizzo ip della macchina su cui vengono creati, nei

quali risiedono agenti e centri di tuple che comunicano attraverso lo scambio di messaggi.

1.2.1 Tuple

Rappresentano “messaggi” composti da un nome e da un numero qualunque di argomenti: `nomeTupla(arg0, arg1, ... argN)`. Vengono utilizzate per la comunicazione fra gli agenti e all’interno dei centri di tuple.

1.2.2 Centri di tuple

Si tratta di contenitori di messaggi (“tuple”) che possono inoltre reagire a determinati stimoli; ad esempio possono: generare, spostare o eliminare un messaggio sia quando viene richiesto sia quando viene depositato. Sono programmabili attraverso il linguaggio logico ReSpecT[4] basato sul Prolog[5]. Un assioma ReSpecT viene chiamato regola e in questo volume verrà rappresentato con uno schema a blocchi come da Figura 1.3 e Figura 1.4.

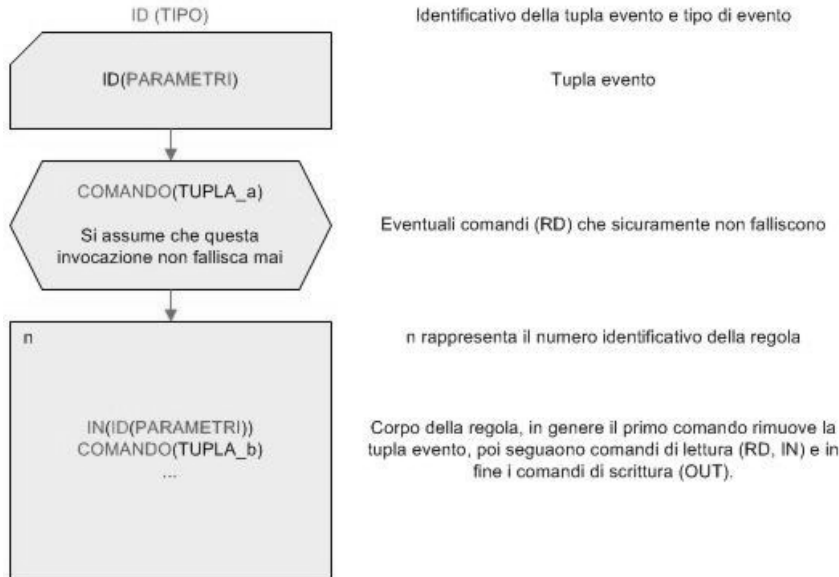


Figura 1.3: Una regola base ReSpecT

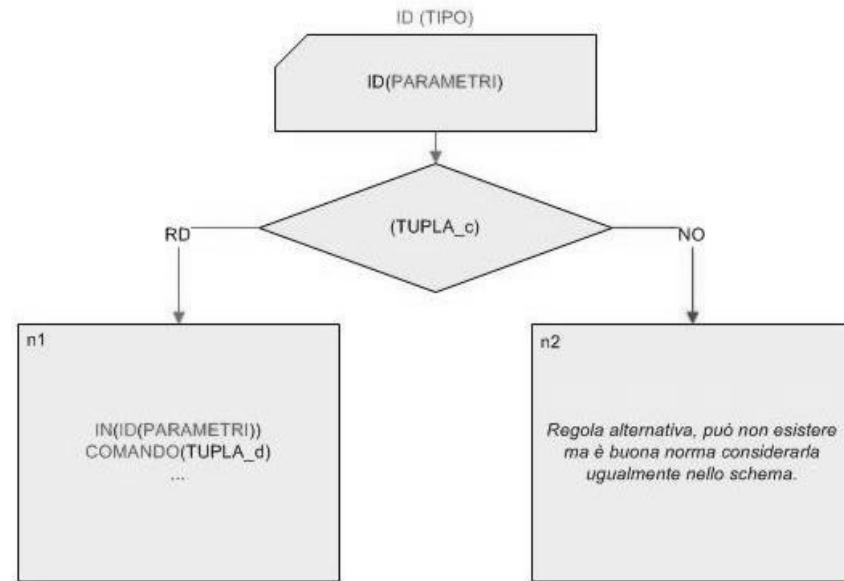


Figura 1.4: Una regola condizionale ReSpecT

Lo schema presentato nella Figura 1.5 è un esempio pratico: corrisponde a due regole in linguaggio ReSpecT che gestiscono una meccanica importante affrontata più avanti nel Capitolo 2. Le due regole si traducono nel seguente codice e sono mutuamente esclusive poichè la prima si attiva se la tupla `waiting(TYPE, TUPLE)` esiste e l'altra, invece, se non esiste.

```

reaction(out_r(wake_up(TYPE)), (
    rd_r(waiting(TYPE, TUPLE)),
    in_r(wake_up(TYPE)),
    in_r(waiting(TYPE, TUPLE)),
    out_r(TUPLE),
    out_r(wake_up(TYPE))
)
).
reaction(out_r(wake_up(TYPE)), (
    no_r(waiting(TYPE, TUPLE)),
    in_r(wake_up(TYPE))
)
).

```

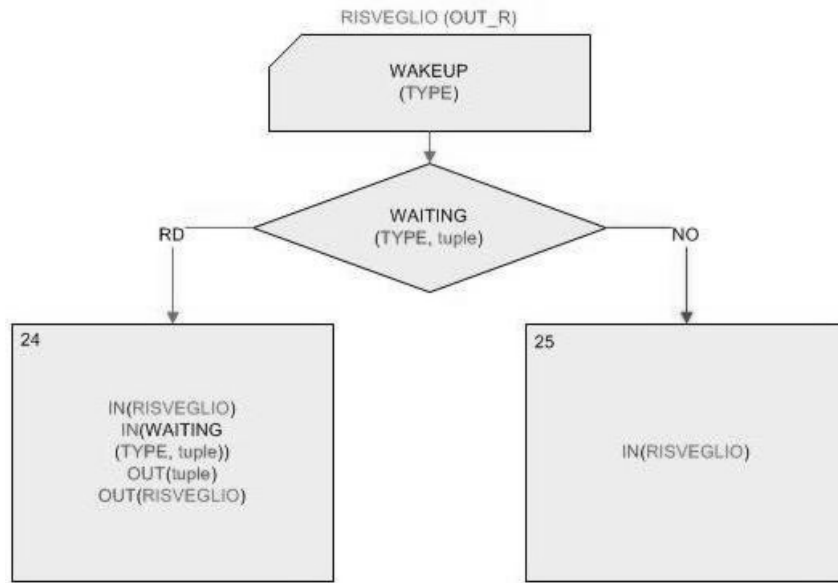


Figura 1.5: Un esempio di due regole ReSpecT

1.2.3 Agenti

Gli agenti sono le entità attive che operano in questo ambiente e interagiscono fra loro attraverso i centri di tuple.

Capitolo 2

Core

Il Core è lo strato più basso dell'infrastruttura **p4p** permette l'interazione con TuCSoN ed implementa la base per gli strati superiori. Essendo realizzato utilizzando: Centri di tuple programmati in ReSpecT, agenti e tuple; presenta le caratteristiche e le problematiche di un sistema distribuito concorrente.

La dinamicità della rete e la concorrenza di un sistema ad agenti sono punti di forza, ma si portano dentro possibili debolezze. Infatti se la possibilità di aggiungere nodi a runtime è di certo una cosa gradita, il fatto che un nodo possa diventare irraggiungibile può comportare diverse problematiche, allo stesso modo la possibilità di richiedere un servizio che non è ancora stato attivato (o che non sarà mai attivato) non deve pregiudicare il normale svolgimento dell'applicazione, pertanto comporta un'opportuna gestione.

2.1 Architettura del Core

Ogni nodo **p4p** è costituito da un RouterIO, un SystemIO con il relativo agente di look up e da un numero imprecisato di agenti generici, corredati dai relativi AgentIO. RouterIO, SystemIO e AgentIO sono realizzati con dei centri di tuple, mentre gli agenti generici e quello di look-up sono agenti che interagiscono ognuno con un solo centro di tuple. Tale scelta permette di rendere l'infrastruttura adattabile il più possibile alle versioni future di TuCSoN, in quanto basta modificare

opportunamente le regole relative ai centri di tuple per allinearsi con le nuove versioni.

Nella Figura 2.1 è possibile identificare questi elementi, ognuno dei quali ha caratteristiche e funzioni ben definite:

- SystemIO (Sezione 2.2)
- Agente di LookUp (Sezione 2.3)
- RouterIO (Sezione 2.4)
- AgentIO (Sezione 2.5)
- Agent (Sezione 2.6)

Tutti questi elementi verranno trattati nelle sezioni successive di questo capitolo che faranno riferimento allo schema UML di Figura 2.2.

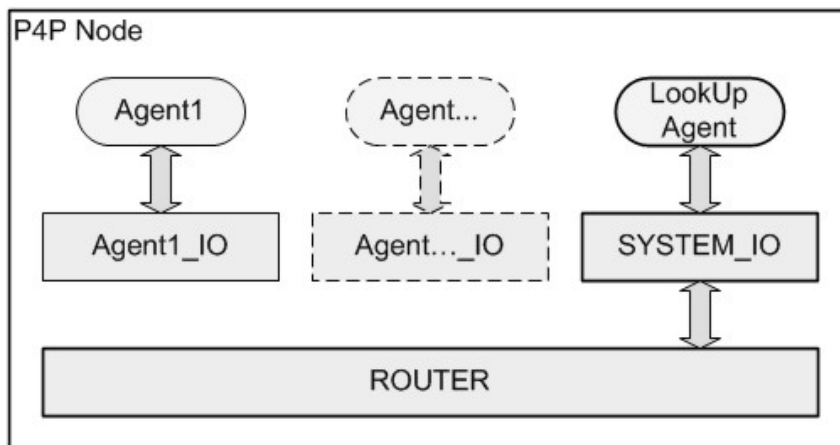


Figura 2.1: Rappresentazione logica di un nodo **p4p**

2.2 SystemIO

SystemIO è l'interfaccia del nodo verso l'esterno, permette il reinstradamento delle richieste remote ed implementa il sistema di look up

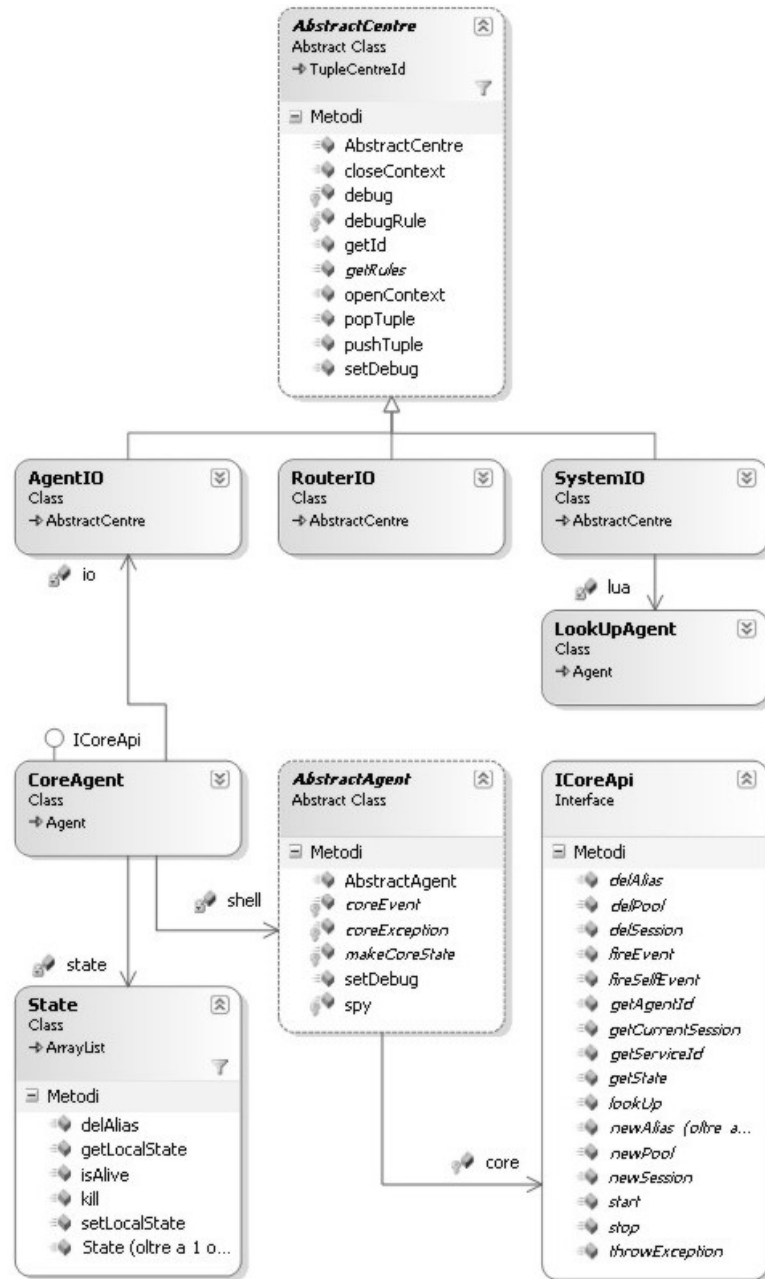


Figura 2.2: Schema UML delle classi principali presenti nel Core

(utilizzato per l'ip look-up e nel peer discovery). Per poter essere raggiungibile il suo ID è well known, più precisamente è uguale alla stringa `system_io`.

2.2.1 Stato interno

Le tuple seguenti rappresentano lo stato interno dell'artefatto, possono essere presenti sin dalla creazione oppure comparire successivamente ma tutte hanno un tempo di vita apprezzabile. A fianco di ciascuna tupla ne verrà specificata la cardinalità.

<code>ip(IP)</code> # ¹ IP: ip del nodo Descrizione: L'ip del nodo corrente. Viene allegato alle richieste remote così da permettere l'indirizzamento delle risposte
<code>router(ID)</code> # ¹ ID: identificatore del RouterIO Descrizione: Identificatore del RouterIO. Viene usato per reinstradare le richieste

2.2.2 Input

Le altre entità comunicano con il SystemIO depositando particolari tuple al suo interno, esse sono in breve tempo consumate e quindi non sono propriamente tuple di stato, ma tuple di comunicazione.

ack(*ID*)

Mittente: SystemIO (remoto)

ID: id della trappola

Descrizione:

Questa tupla notifica l'effettivo raggiungimento di un nodo remoto. Se questa tupla non arriva prima dello scadere del timeout viene generata una tupla di eccezione

ip(*AGENT,SESSION,IO,ID,IP*)

Mittente: Look-Up Agent

AGENT: agente che ha richiesto il look-up

SESSION: sessione dell'agente

IO: identificativo del IO dell'agente

ID: utilizzato per correlare la richiesta con la risposta

IP: indirizzo IP del nodo locale

Descrizione:

Questa tupla è la risposta dell'agente di look-up (Sezione 2.3).

lookup(*FROM,SESSION,IO,ID,IP*)

Mittente: RouterIO

vedi Sezione 2.2.4

Descrizione:

Questa tupla rappresenta una richiesta per l'agente di look-up.

make_try(*IP,LOCAL_IP,ID_TRAP,MSG*)

Mittente: SystemIO (remoto)

IP: Indirizzo del nodo che ha inviato questa tupla

LOCAL_IP: Indirizzo del nodo corrente

ID_TRAP: id della trappola creata dal mittente

MSG(tuple): messaggio ricevuto

Descrizione:

Questa tupla rappresenta un messaggio ricevuto da un nodo remoto.

try(*AGENT,SESSION,IO,IP,MSG*)

Mittente: RouterIO

AGENT: agente che ha generato la richiesta

SESSION: sessione dell'agente

IO: identificativo del IO dell'agente

IP: indirizzo remoto da contattare

MSG(tuple): tupla che si vuole inoltrare

Descrizione:

Rappresenta la richiesta di inoltro della tupla MSG verso il SystemIO di un nodo remoto

2.2.3 Output

Questa entità comunica con le altre depositando particolari tuple, le quali verranno trattate nella sezione Input dei relativi destinatari.

ack(*ID*)

Destinatario: SystemIO (remoto)

exception(*broken,AGENT,SESSION,IO,MSG*)

Destinatario: RouterIO

incoming(*IP,LOCAL_IP,MSG*)

Destinatario: RouterIO

ip(*AGENT,SESSION,IO,ID,IP*)

Destinatario: RouterIO

make_try(*IP,LOCAL_IP,ID-TRAP,MSG*)

Destinatario: SystemIO (remoto)

2.2.4 Richieste pendenti

L'agente di look-up che risiede in questo nodo rimane in attesa di ricevere le tuple di richiesta che lo riguardano.

lookup(*FROM,SESSION,IO,ID,IP*)

Mittente: RouterIO

FROM: agente che ha richiesto il look-up

SESSION: sessione dell'agente

IO: identificativo del IO dell'agente

ID: utilizzato per correlare la richiesta con la risposta

IP: indirizzo remoto da contattare

Descrizione:

Richiesta per l'agente di look-up (Sezione 2.3).

2.2.5 Comportamento

Le seguenti regole ReSpecT sono alla base del comportamento di questo artefatto.

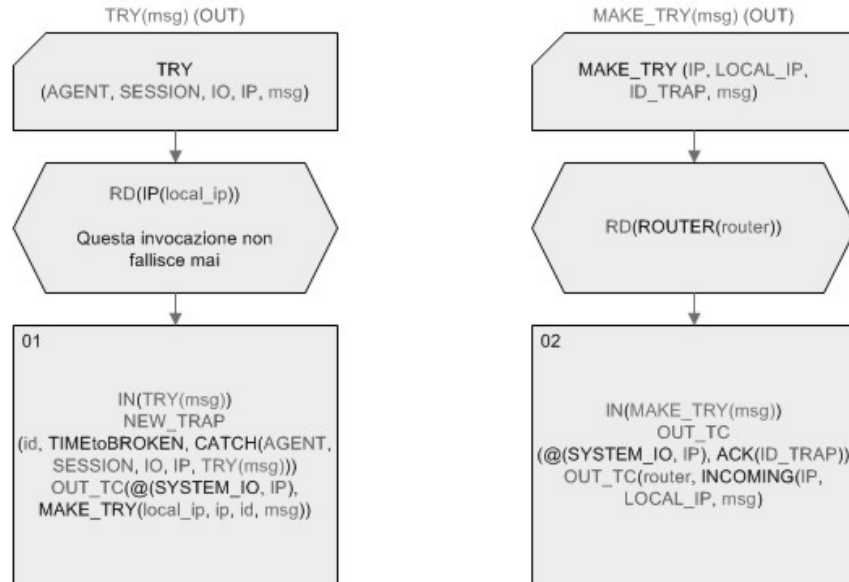


Figura 2.3: Invio e ricezione di un invito

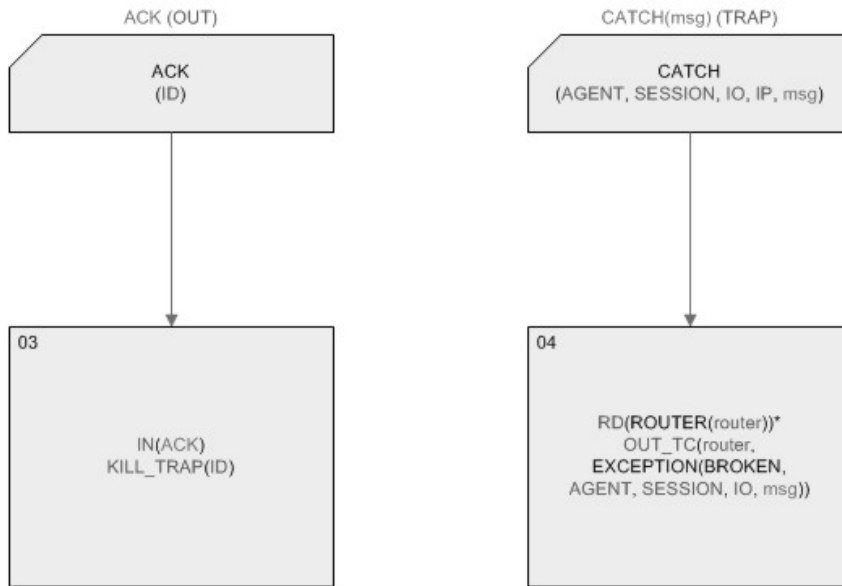


Figura 2.4: Conferma di ricezione e generazione di un'eccezione

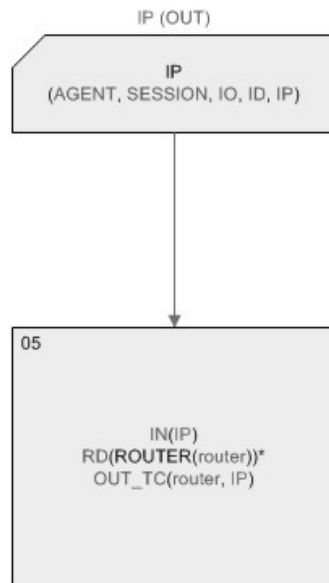


Figura 2.5: Reinstradamento della risposta dell'agente di look-up

2.3 Agente di LookUp

L'agente di LookUp è lo strumento che il SystemIO utilizza per interrogare i nodi e scoprire informazioni utili, ad esempio: capire se un nodo è attivo o scoprire il proprio ip.

2.3.1 Interazione con TuCSoN

TuCSoN mette a disposizione un accesso telnet che restituisce alcune informazioni sull'infrastruttura. Più precisamente ritorna il seguente testo:

```
TuCSoN Infrastructure (version 1.4.0)
Current date Mon Jan 29 20:44:01 CET 2007
Running since Mon Jan 29 20:39:10 CET 2007
```

Tali informazioni però non sono sufficienti alle nostre necessità, per questo motivo ho introdotto le seguenti modifiche:

Alla classe `alice.tucson.api.util.TucsonProtocolTCP` ho aggiunto il metodo:

```
/**
 * Return the IP Address of the remote end point
 */
public String getIp(){
    try{
        return socket.getInetAddress().toString().substring(1);
    }
    catch(Exception e){
        return "";
    }
}
```

Nel package `alice.tucson.runtime` ho aggiunto o modificato le seguenti classi: `InfrastructureExtensionInformer`, (aggiunta), rappresenta l'interfaccia di una estensione di TuCSoN ed è definita dal seguente codice:

```
package alice.tucson.runtime;
```

```

/**
 * Interface of a Extension Informer
 *     for TuCSoN infrastrutture
 * @author Giorgio Amadei
 */
public interface InfrastructureExtensionInformer {
    public String getInfrastructureExtensionInfo();
}

```

Alla classe `WelcomeAgent` ho modificato il costruttore ed aggiunto un campo privato:

```

private InfrastructureExtensionInformer _extension;

public WelcomeAgent(int port, Node node, ContextManager cm,
                    InfrastructureExtensionInformer extension){
    contextManager = cm;
    this.port=port;
    this.node=node;
    _extension = extension;
    start();
}

```

Il suo metodo `public void run()` è stato modificato per gestire l'estensione:

```

if (dialog.isTelnetRequest()){
    // Extension
    String extension = "none";
    if(_extension != null)
        extension = _extension.getInfrastructureExtensionInfo();
    // Extension
    // your ip
    String ip = "unknown";
    try {
        ip = "" + ((TucsonProtocolTCP)dialog).getIp();
    }
    catch(Exception e){}
    // your ip
}

```

```

String _return = "\n\r";
String message="TuCSoN Infrastructure (version "
+ Tucson.getVersion() + ")" + _return +
"Extension      " + extension + _return +
"Current date  " + new Date() + _return +
"Running since " + node.getInstallationDate() + _return+
"Your ip       " + ip + _return;
dialog.sendTelnetRequestAnswer(message);
dialog.end();
} ...

```

Alla classe Node ho aggiunto un campo privato e modificato il costruttore ed un metodo:

```

/** Extension - Add by Giorgio Amadei 2006 */
private InfrastructureExtensionInformer extension;

public Node(String configFile, int portNumber,
            TupleArgument persistTempl,
            InfrastructureExtensionInformer extension)
    throws TucsonException {
    this.configFile = configFile;
    tcp_port = portNumber;
    persistencyTemplate = persistTempl;
    idConfigTC = new TupleCentreId("'$ORG'");
    idObsTC = new TupleCentreId("'$OBS'");
    observed = false;
    agents = new ArrayList();
    this.extension = extension;
}

```

Nel metodo `private void bootManagementAgents()` ho modificato il costruttore del `WelcomeAgent`:

```

private void bootManagementAgents() {
    new NodeManagementAgent(
        configTupleCentreInfo.getVirtualMachine(),this);
    ContextManager ctxman = new ContextManager(this,
        configTupleCentreInfo.getVirtualMachine());
}

```

```
new WelcomeAgent(tcp_port,this,ctxman,extension);
}
```

Al termine delle modifiche TuCSoN risponde a telnet con le seguenti stringhe di testo, divise in due colonne: la prima di 14 caratteri contiene il tipo di informazione; la parte restante della stringa contiene il dato:

```
TuCSoN Infrastructure (version 1.4.0)
Extension      P4P Peer for Passion (version 1.0)
Current date   Mon Jan 29 20:44:01 CET 2007
Running since  Mon Jan 29 20:39:10 CET 2007
Your ip        XXX.XXX.XXX.XXX
```

Se però l'infrastruttura **p4p** non è ancora pronta, TuCSoN risponde anteponendo al nome dell'estensione la stringa "starting":

```
TuCSoN Infrastructure (version 1.4.0)
Extension      starting P4P Peer for Passion (version 1.0)
Current date   Mon Jan 29 20:44:01 CET 2007
Running since  Mon Jan 29 20:39:10 CET 2007
Your ip        XXX.XXX.XXX.XXX
```

2.3.2 Comportamento

Il comportamento di questo componente è relativamente semplice e può essere descritto nei seguenti passi:

1. Il RouterIO deposita la tupla `lookup(FROM,SESSION,IO,ID,IP)` nel SystemIO;
2. Il Look-up Agent la preleva ed interroga via telnet il nodo remoto;
3. Il nodo remoto risponde con le informazioni testuali;
4. L'agente le riceve e ne estrae l'ip: se è valido lo salva come tupla di stato `ip(IP)` e deposita la risposta `ip(FROM,SESSION,IO,ID,IP)` dove IP è uguale all'ip ottenuto se è valido altrimenti è una stringa vuota

5. SystemIO preleva la tupla di risposta e la invia al RouterIO (Figura 2.5)

2.4 RouterIO

Il RouterIO conserva le informazioni relative agli agenti e ai nodi raggiungibili, viene utilizzato quindi per reinstradare le richieste di attivazione degli agenti. Quando un agente locale viene caricato esso si “registra” nel RouterIO così da risultare raggiungibile.

2.4.1 Stato interno

Le tuple seguenti rappresentano lo stato interno dell’artefatto. Possono essere presenti sin dalla creazione oppure comparire successivamente, ma tutte hanno un tempo di vita apprezzabile. A fianco di ciascuna tupla ne verrà specificata la cardinalità.

resource (<i>SERVICE, AGENT, IO</i>) #0-N SERVICE: tipo di servizio offerto dalla risorsa AGENT: identificativo dell’agente che lo offre IO: il suo AgentIO Descrizione: Rappresenta una risorsa locale
resource_pool (<i>POOL, IP, PORT</i>) #0-N POOL: identificativo del contenitore di risorse IP: indirizzo del nodo remoto PORT: porta logica su cui il nodo è in ascolto Descrizione: Rappresenta un contenitore di risorse remote

waiting(*ID, TYPE, MSG*) #0-N

ID: identificativo univoco
 TYPE(tuple): evento atteso
 MSG(Tuple): messaggio
Descrizione:
 Messaggio in attesa

2.4.2 Input

Le altre entità comunicano con il SystemIO depositando particolari tuple al suo interno, esse sono consumate in breve tempo e quindi non sono propriamente tuple di stato, ma tuple di comunicazione.

cmd(*resource, TYPE, parameter(SERVICE, AGENT, IO)*)

Mittente: AgentIO
 TYPE: può assumere i valori **new** e **del**
 SERVICE: servizio offerto dall'agente
 AGENT: agente offre il servizio
 IO: identificativo del IO dell'agente
Descrizione:
 Comando per la creazione e distruzione di un descrittore di risorsa locale

cmd(*resource_pool, TYPE, parameter(POOL, IP, PORT)*)

Mittente: AgentIO
 TYPE: può assumere i valori **new** e **del**
 POOL: identificativo del contenitore di risorse remote
 IP: indirizzo del nodo remoto
 PORT: porta logica su cui ascolta il nodo
Descrizione:
 Comando per la creazione e distruzione di un descrittore di contenitori di risorse remote

exception(*MSG,AGENT,SESSION,IO,DATI*)

Mittente: SystemIO

MSG: tipo di eccezione

AGENT: agente che ha generato la richiesta

SESSION: sessione dell'agente

IO: identificativo del IO dell'agente

DATI(tuple): dati aggiuntivi riguardanti l'eccezione

Descrizione:

Notifica il verificarsi di una eccezione

incoming(*IP,LOCAL_IP, INVITE*)

Mittente: SystemIO

IP: indirizzo del destinatario

LOCAL_IP: ip del nodo corrente

INVITE(tuple): **invite**(IP,POINT,SERVICE,ID)

Descrizione:

Richiesta remota di un servizio locale

invite(*POINT,SERVICE,ID*)

Mittente: AgentIO

POINT(tuple): End point (Sezione 2.7.1) del mittente

SERVICE: servizio richiesto

ID: identificativo per far corrispondere richiesta e risposta

Descrizione:

Richiesta di un servizio locale

invite(*POINT,POOL,SERVICE,ID*)

Mittente: AgentIO

POINT(tuple): End point (Sezione 2.7.1) del mittente

POOL: identificativo del contenitore di risorse remote **local**

SERVICE: servizio richiesto

ID: identificativo per far corrispondere richiesta e risposta

Descrizione:

Richiesta di un servizio remoto

ip(*AGENT,SESSION,IO,ID,IP*)

Mittente: SystemIO

AGENT: agente che ha richiesto il look-up

SESSION: sessione dell'agente

IO: identificativo del IO dell'agente

ID: utilizzato per correlare la richiesta con la risposta

IP: indirizzo IP del nodo locale

Descrizione:

Questa tupla è la risposta dell'agente di look-up (Sezione 2.3).

lookup(*FROM,SESSION,IO,ID,IP*)

Mittente: AgentIO

FROM: agente che ha richiesto il look-up

SESSION: sessione dell'agente

IO: identificativo del IO dell'agente

ID: utilizzato per correlare la richiesta con la risposta

IP: indirizzo remoto da contattare

Descrizione:

Richiesta per l'agente di look-up (Sezione 2.3).

2.4.3 Output

Questa entità comunica con le altre depositando particolari tuple, le quali verranno trattate nella sezione Input dei relativi destinatari.

exception(*TYPE,AGENT,SESSION,IO,MSG*)

Destinatario: AgentIO

invite(*POINT,AGENT,ID,IO*)

Destinatario: AgentIO

ip(*AGENT,SESSION,IO,ID,IP*)

Destinatario: AgentIO

lookup (<i>AGENT,SESSION,IO,ID,IP</i>)
Destinatario: SystemIO

try (<i>AGENT,SESSION,IO,IP,MSG</i>)
Destinatario: SystemIO

2.4.4 Comportamento

Le seguenti regole ReSpecT sono alla base del comportamento di questo artefatto.

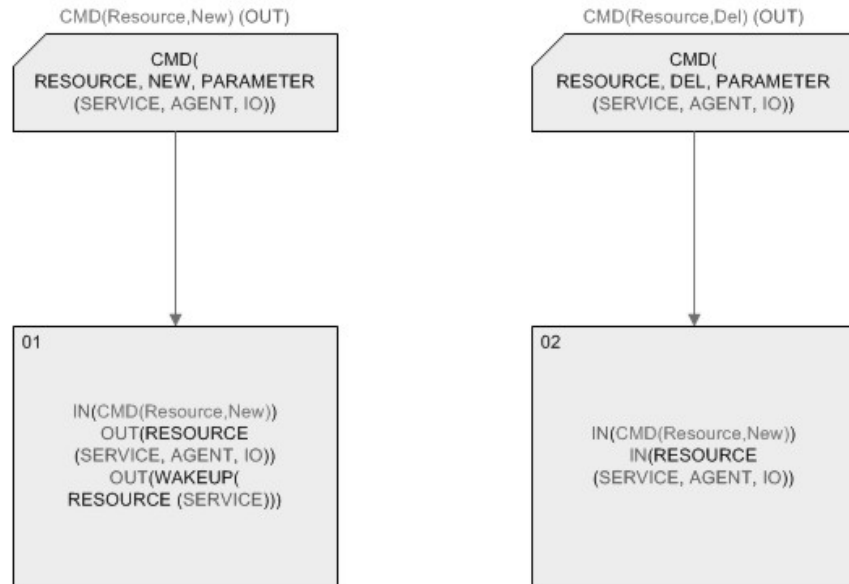


Figura 2.6: Comando di creazione e distruzione del descrittore di una risorsa

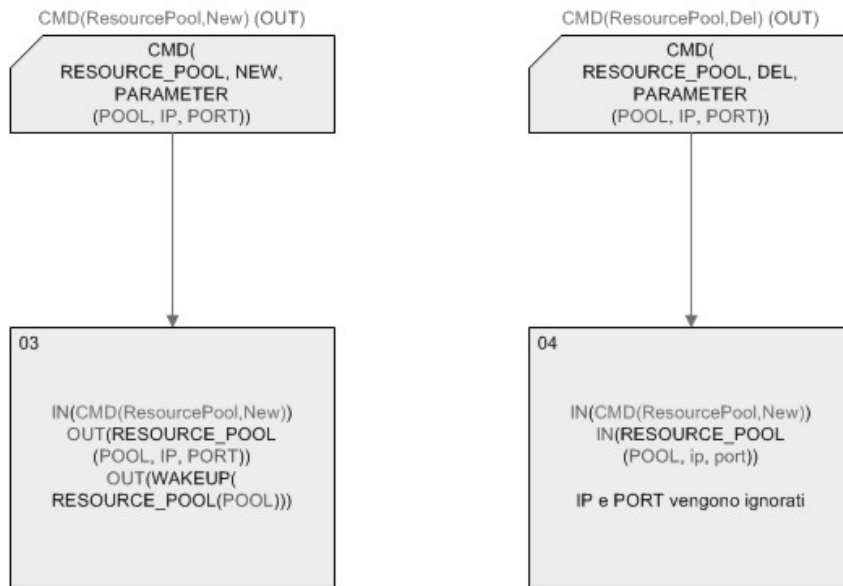


Figura 2.7: Comando di creazione e distruzione del descrittore di un contenitore di risorse

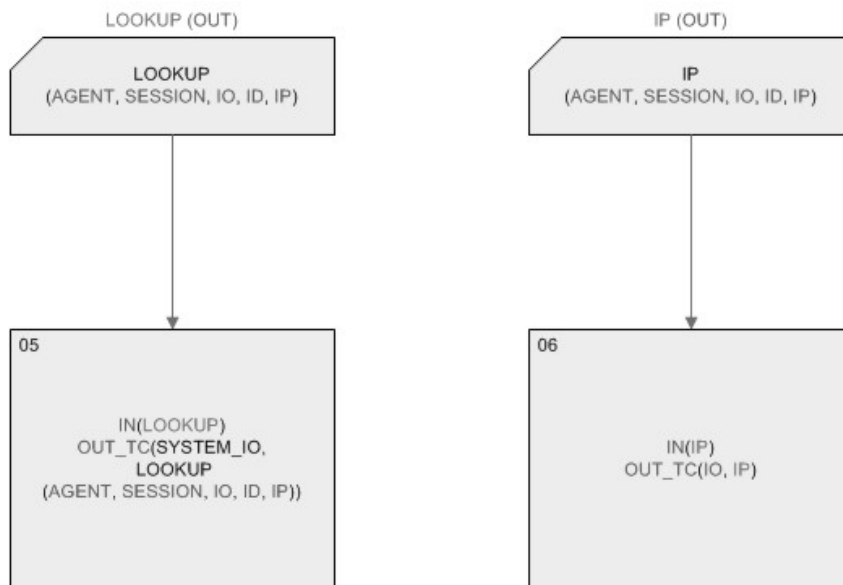


Figura 2.8: Reinstradamento richiesta e risposta di look-up

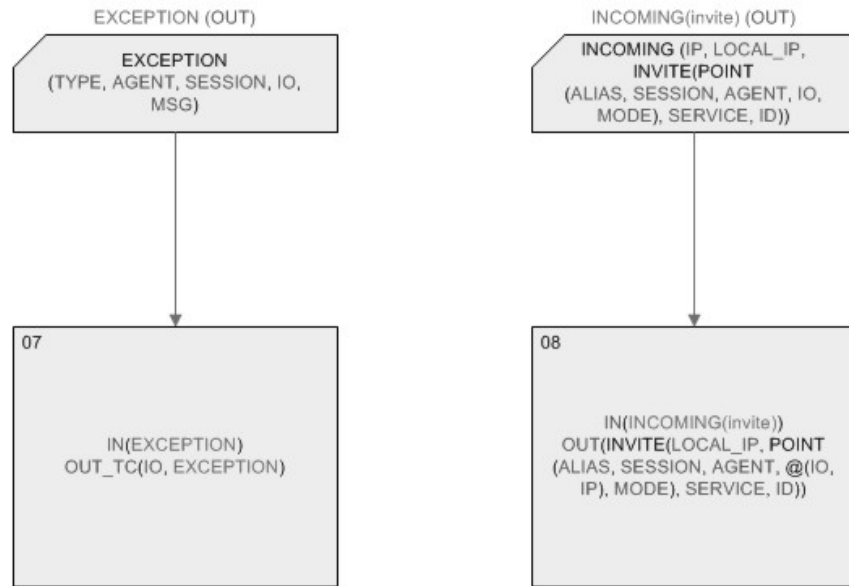


Figura 2.9: Reinstradamento eccezione e conversione di un invito remoto

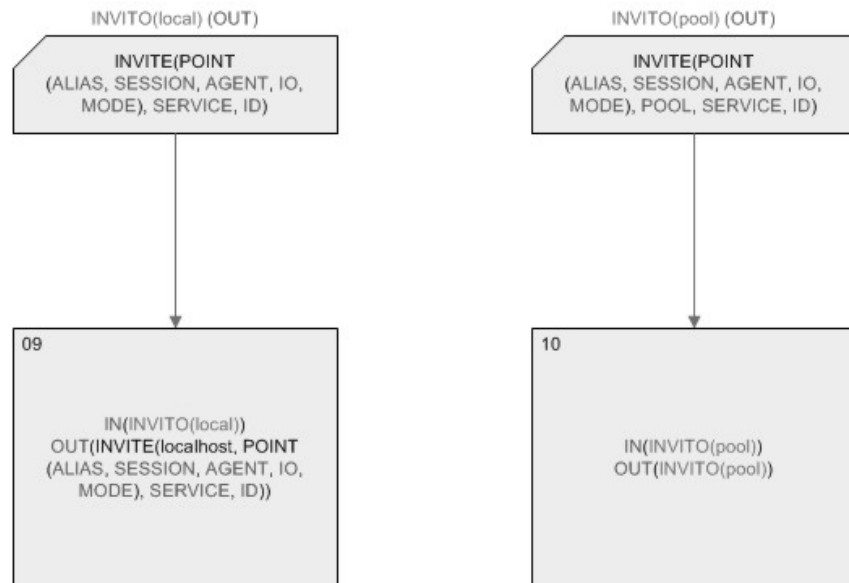


Figura 2.10: Regole di supporto per convertire l'evento esterno **out** in evento interno **out_r**

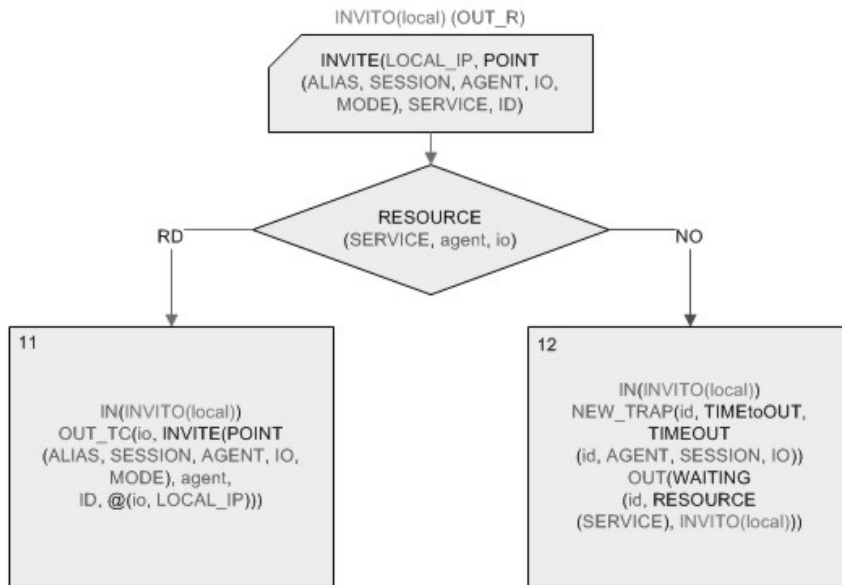


Figura 2.11: Gestione richiesta di risorsa locale

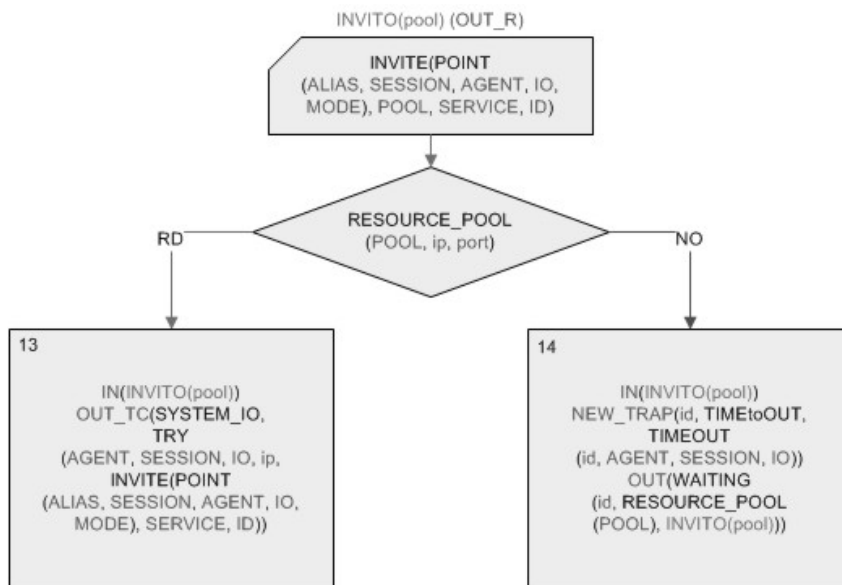


Figura 2.12: Gestione richiesta di risorsa remota

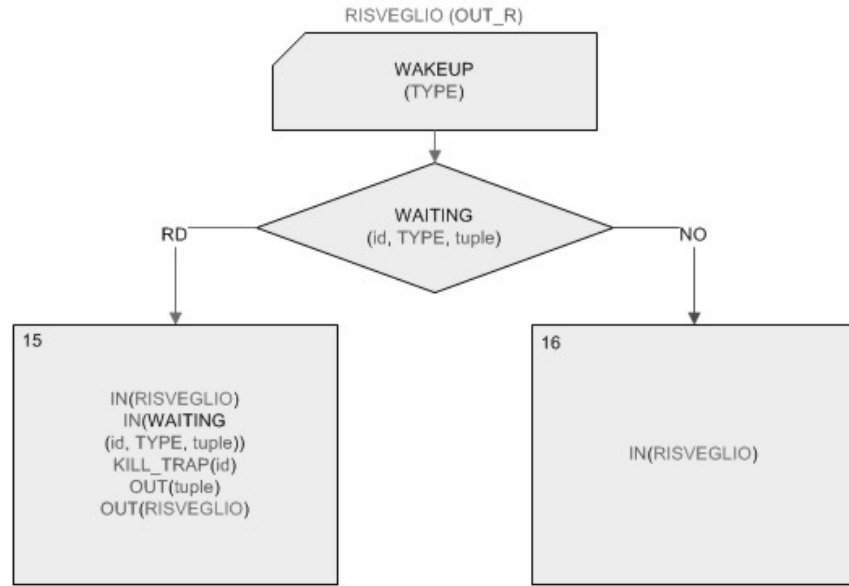


Figura 2.13: Risveglio messaggi in attesa

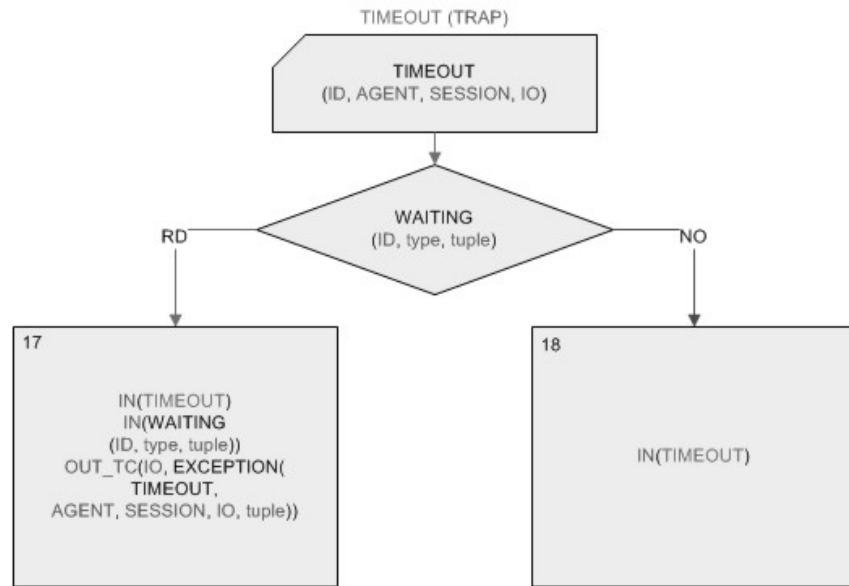


Figura 2.14: Generazione eccezioni da timeout

2.5 AgentIO

L'AgentIO è l'interfaccia dell'agente con l'infrastruttura, reindirizza i messaggi di comunicazione e gestisce gli alias.

2.5.1 Stato interno

Le tuple seguenti rappresentano lo stato interno dell'artefatto. Possono essere presenti sin dalla creazione oppure comparire successivamente ma tutte hanno un tempo di vita apprezzabile. A fianco di ciascuna ne verrà specificata la cardinalità.

alias (<i>ALIAS,SESSION,incomplete</i>) #0-N ALIAS: identificativo dell'alias SESSION: identificativo della relativa sessione Descrizione: Un alias incompleto
alias (<i>ALIAS,SESSION,POINT</i>) #0-N ALIAS: identificativo dell'alias SESSION: identificativo della relativa sessione POINT(tuple): End point (Sezione 2.7.1) del destinatario Descrizione: Un alias completo
local_pool (<i>LOCAL_IP</i>) #1 LOCAL_IP: la costante local Descrizione: Viene utilizzata in alcune regole
max (<i>MAX</i>) #0-1 MAX: la costante 2147483645 Descrizione: Identifica il massimo numero che può essere utilizzato come identificatore, se non è presente non c'è nessun limite

next (<i>alias,I</i>) # ¹ I: id valido Descrizione: Il prossimo identificatore valido per gli alias
next (<i>session,I</i>) # ¹ I: id valido Descrizione: Il prossimo identificatore valido per le sessioni
router (<i>ID</i>) # ¹ ID: identificatore del RouterIO Descrizione: Identificatore del RouterIO viene usato per reinstradare le richieste
session (<i>SESSION,AGENT</i>) # ^{0-N} SESSION: identificativo della sezione AGENT: identificativo dell'agente Descrizione: Una sessione aperta
ticket (<i>AGENT,IO,POINT</i>) # ^{0-N} AGENT: identificativo dell'agente IO: AgentIO dell'agente POINT(tuple): End point (Sezione 2.7.1) remoto Descrizione: Un invito non ancora servito

waiting(*TYPE,MSG*) #0-N

TYPE(tuple): evento atteso

MSG(Tuple): messaggio

Descrizione:

Messaggio in attesa

2.5.2 Input

Le altre entità comunicano con il SystemIO depositando particolari tuple al suo interno, esse sono consumate in breve tempo e quindi non sono propriamente tuple di stato, ma tuple di comunicazione.

ack(*ID*)

Mittente: AgentIO (remoto)

ID: id della trappola

Descrizione:

Questa tupla notifica l'effettivo raggiungimento di un nodo remoto. Se questa tupla non arriva prima dello scadere del timeout viene generata una tupla di eccezione

cmd(*AGENT,SESSION,agent,CMD,parameter(SERVICE)*)

Mittente: Agent

AGENT: id dell'agente

SESSION: sessione dell'agente

CMD: può assumere i valori **new** o **del**

SERVICE: servizio svolto

Descrizione:

Comando per la registrazione di un agente nel router

cmd(*AGENT,SESSION,alias,del,parameter(ALIAS)*)

Mittente: Agent
 AGENT: id dell'agente
 SESSION: sessione dell'agente
 ALIAS: id di un alias
Descrizione:
 Comando per la distruzione di un alias

cmd(*AG,SES,alias,new,parameter(PPOOL,SERVICE,ID,MODE)*)

Mittente: Agent
 AG: id dell'agente
 SES: sessione dell'agente
 PPOOL: contenitore di risorse remote o la costante **local**
 SERVICE: servizio richiesto
 ID: utilizzato per correlare la richiesta con la risposta
 MODE: modalità di utilizzo
Descrizione:
 Comando per la distruzione di un alias

cmd(*AGENT,SESSION,alias,kill,parameter(ALIAS)*)

Mittente: Agent
 AGENT: id dell'agente
 SESSION: sessione dell'agente
 ALIAS: id di un alias
Descrizione:
 Comando per la distruzione di un alias incompleto e di tutte le relative richieste pendenti

cmd(*alias,del,parameter(ALIAS,SESSION)*)

Mittente: AgentIO (remoto)
 ALIAS: id di un alias
 SESSION: sessione dell'agente
Descrizione:
 Comando remoto per la distruzione di un alias

cmd(*AGENT,SESSION,lookup,new,parameter(ID,IP)*)

Mittente: Agent

AGENT: id dell'agente

SESSION: sessione dell'agente

ID: utilizzato per correlare la richiesta con la risposta

IP: indirizzo del nodo remoto

Descrizione:

Comando per il servizio di look up

cmd(*AG,SES,resource_pool,CMD,parameter(Pool,IP,PORT)*)

Mittente: Agent

AG: id dell'agente

SES: sessione dell'agente

CMD: può assumere i valori **new** o **del**

POOL: id di un contenitore di risorse remote

IP: indirizzo del nodo remoto

PORT: porta logica su cui il nodo è in ascolto

Descrizione:

Comando di creazione o distruzione del descrittore di un contenitore di risorse remote

cmd(*AGENT,SESSION,session,TYPE,parameter(null)*)

Mittente: Agent

AGENT: id dell'agente

SESSION: sessione dell'agente

TYPE: può assumere i valori **new** o **del**

Descrizione:

Comando per la creazione o distruzione di una sessione

exception(*MSG, AGENT, SESSION, IO, DATI*)

Mittente: RouterIO, RouterIO (remoto)

MSG: tipo di eccezione

AGENT: agente che ha generato la richiesta

SESSION: sessione dell'agente

IO: identificativo dell'IO dell'agente

DATI(tuple): dati aggiuntivi riguardanti l'eccezione

Descrizione:

Notifica il verificarsi di una eccezione

input(*AGENT, ALIAS, SESSION, TYPE, DATA*)

Mittente: Agent

AGENT: id del mittente

ALIAS: destinatario

SESSION: sessione dell'agente

TYPE: tipo di messaggio

DATA: messaggio

Descrizione:

Richiesta di inoltro di un messaggio

ip(*AGENT, SESSION, IO, ID, IP*)

Mittente: RouterIO

AGENT: agente che ha richiesto il look-up

SESSION: sessione dell'agente

IO: identificativo del IO dell'agente

ID: utilizzato per correlare la richiesta con la risposta

IP: indirizzo IP del nodo locale

Descrizione:

Questa tupla è la risposta dell'agente di look-up (Sezione 2.3).

invite(*POINT, AGENT_L, ID, IO_L*)

Mittente: RouterIO

POINT(tuple): End point (Sezione 2.7.1) del mittente

AGENT_L: id dell'agente locale

ID: utilizzato per correlare la richiesta con la risposta

IO_L: AgentIO locale

Descrizione:

Richiesta del servizio svolto

make_alias(*AGENT, SESSION, POINT*)

Mittente: AgentIO (remoto)

AGENT: agente che ha generato la richiesta

SESSION: sessione dell'agente

POINT(tuple): End point (Sezione 2.7.1) remoto

Descrizione:

Richiesta di completamento alias

make_try(*ALIAS, SESSION, ID_TRAP, MSG*)

Mittente: AgentIO (remoto)

ALIAS: alias locale del mittente

SESSION: Relativa sessione locale

ID_TRAP: id della trappola creata dal mittente

MSG(tuple): messaggio ricevuto

Descrizione:

Questa tupla rappresenta un messaggio ricevuto da un nodo remoto.

2.5.3 Output

Questa entità comunica con le altre depositando particolari tuple, esse verranno trattate nella sezione Input dei relativi destinatari.

ack(*ID*)

Destinatario: AgentIO (remoto)

cmd (<i>alias,del,parameter(ALIAS,SESSION)</i>)
Destinatario: AgentIO (remoto)
cmd (<i>resource,TYPE,parameter(SERVICE,AGENT,IO)</i>)
Destinatario: RouterIO
cmd (<i>resource_pool,TYPE,parameter(Pool,IP,PORT)</i>)
Destinatario: RouterIO
cmd_return (<i>AG,SES,TYPE,CMD,PARAMETER,RETURN</i>)
Destinatario: Agent
invite (<i>IP,POINT,SERVICE,ID</i>)
Destinatario: RouterIO
invite (<i>POINT,POOL,SERVICE,ID</i>)
Destinatario: RouterIO
lookup (<i>FROM,SESSION,IO,ID,IP</i>)
Destinatario: RouterIO
make_alias (<i>AGENT,SESSION,POINT</i>)
Destinatario: AgentIO (remoto)
make_try (<i>ALIAS,SESSION,ID-TRAP,MSG</i>)
Destinatario: AgentIO (remoto)
output (<i>FROM,AGENT,SESSION,TYPE,DATA</i>)
Destinatario: AgentIO (remoto)

2.5.4 Richieste pendenti

L'agente generico che si interfaccia con questo artefatto rimane in attesa di ricevere le tuple di richiesta che lo riguardano.

cmd_return(*AG, SES, TYPE, CMD, PARAMETER, RETURN*)

Mittente: AgentIO

AG: agente destinatario

SES: sessione dell'agente

TYPE: tipo della funzione richiesta (**alias**)

CMD: il comando (**new**)

PARAMETER(tuple): parameter(PPOOL, SERVICE, ID)

RETURN(tuple): return(ALIAS)

ALIAS: id di ritorno

Descrizione:

Tupla di ritorno, all'occorrenza viene richiesta dall'agente

output(*FROM, AGENT, SESSION, TYPE, DATA*)

Mittente: AgentIO

FROM: alias mittente

AGENT: destinatario

SESSION: sessione dell'agente

TYPE: tipo di messaggio

DATA: messaggio

Descrizione:

Messaggio per l'agente

2.5.5 Comportamento

Le regole ReSpecT sono alla base del comportamento di questo artefatto, ma, pur essendo esaustive più avanti verranno affrontate le meccaniche di interazione nel dominio dei tempi.

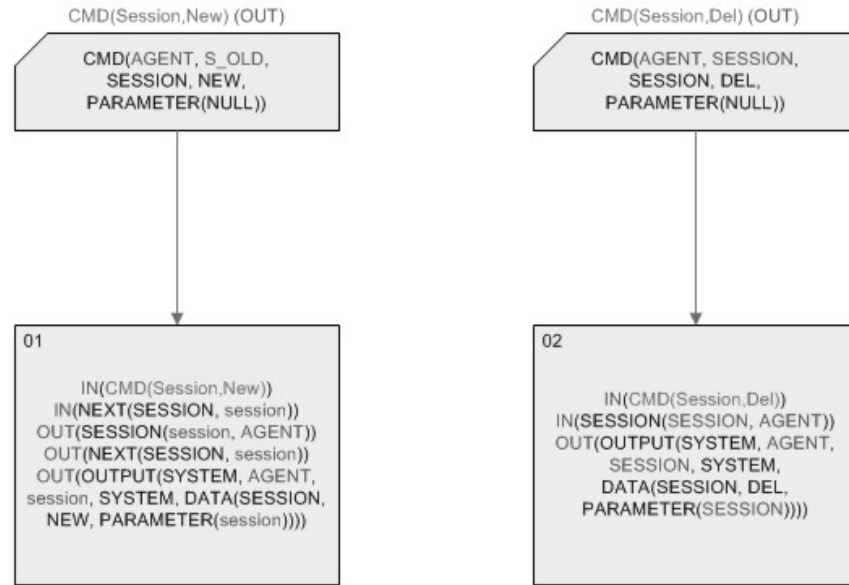


Figura 2.15: Comando di creazione e distruzione di una sessione

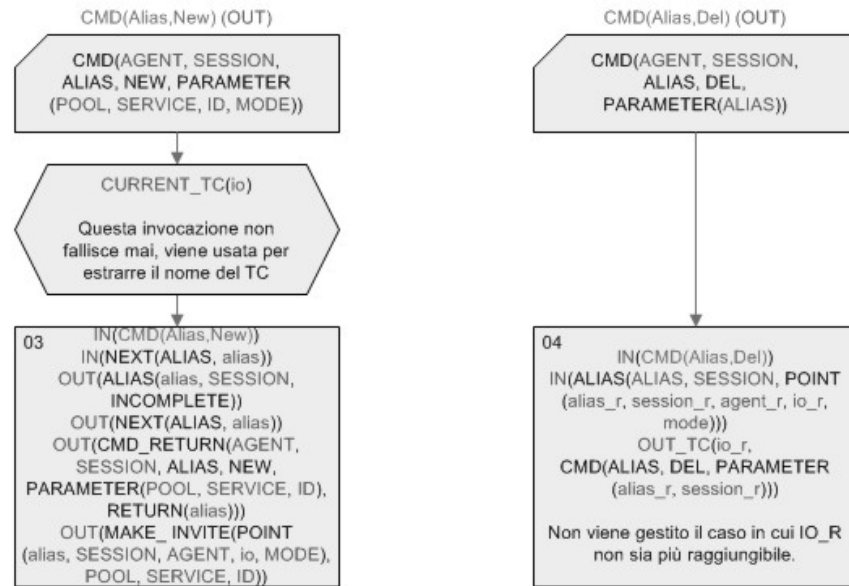


Figura 2.16: Comando di creazione e distruzione di un alias

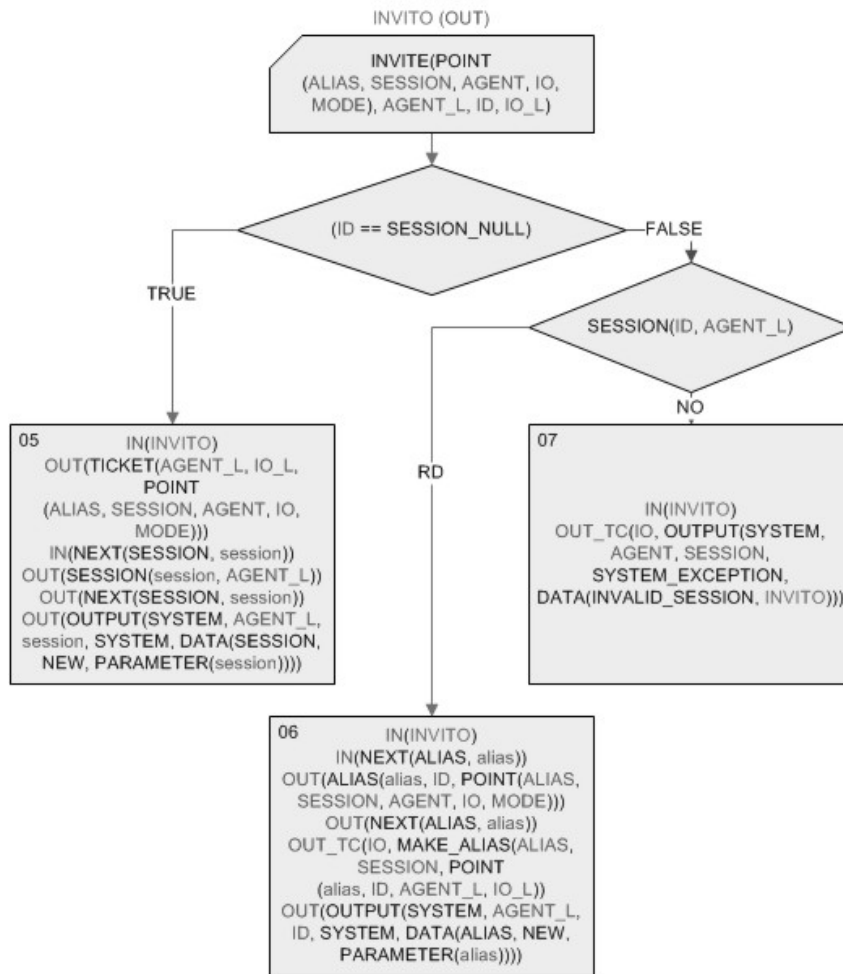


Figura 2.17: Gestione della ricezione di un invito

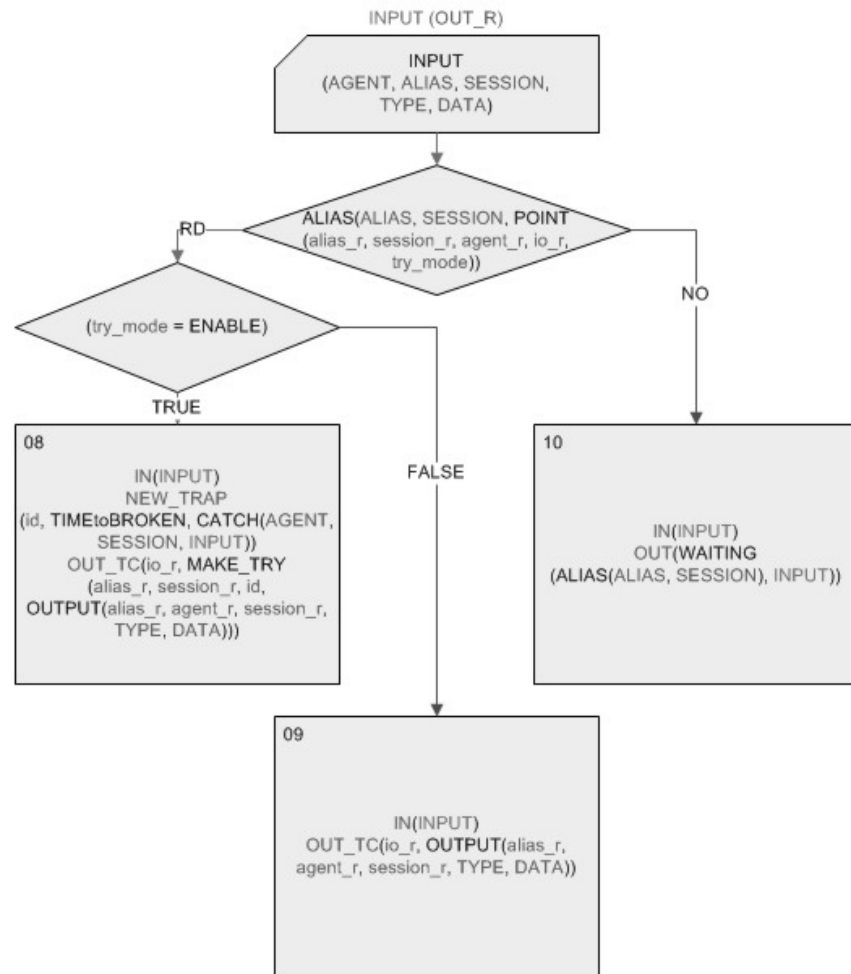


Figura 2.18: Reinstradamento messaggi dell'agente

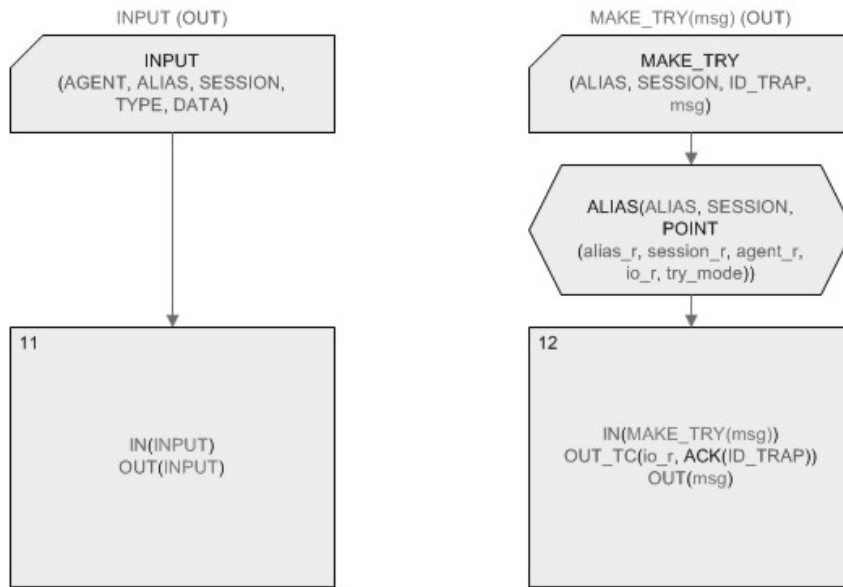


Figura 2.19: Regola di supporto e tentativo di invio di un messaggio

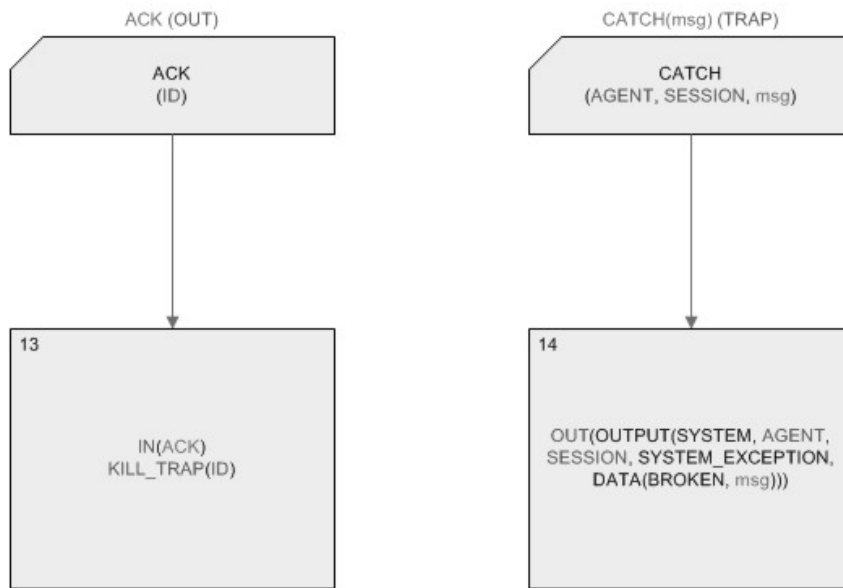


Figura 2.20: Messaggio di conferma e generazione eccezioni

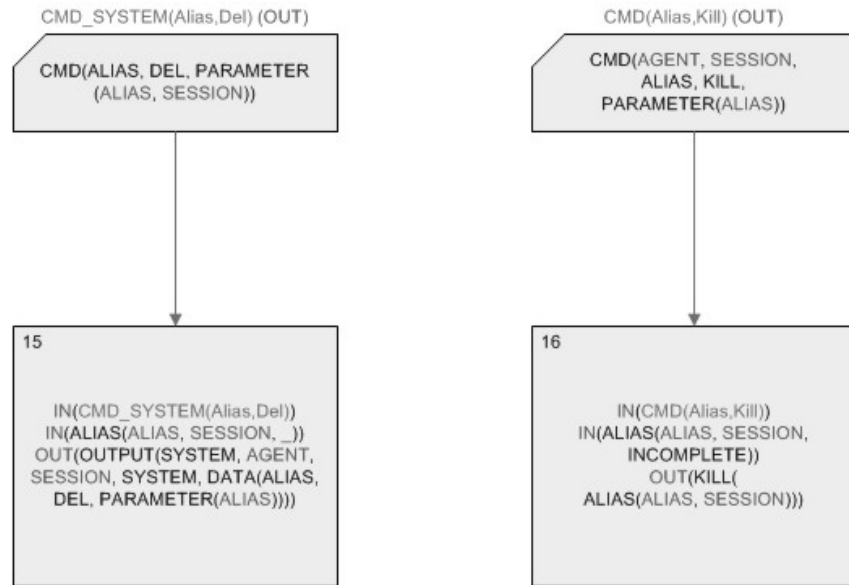


Figura 2.21: Comando remoto di distruzione di un alias e rimozione alias incompleto

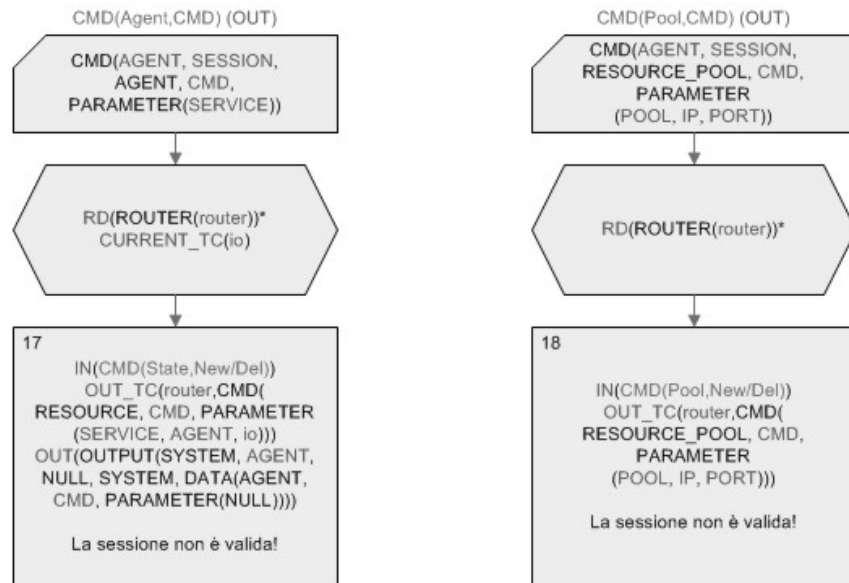


Figura 2.22: Notifica stato dell'agente e dei descrittori di risorse

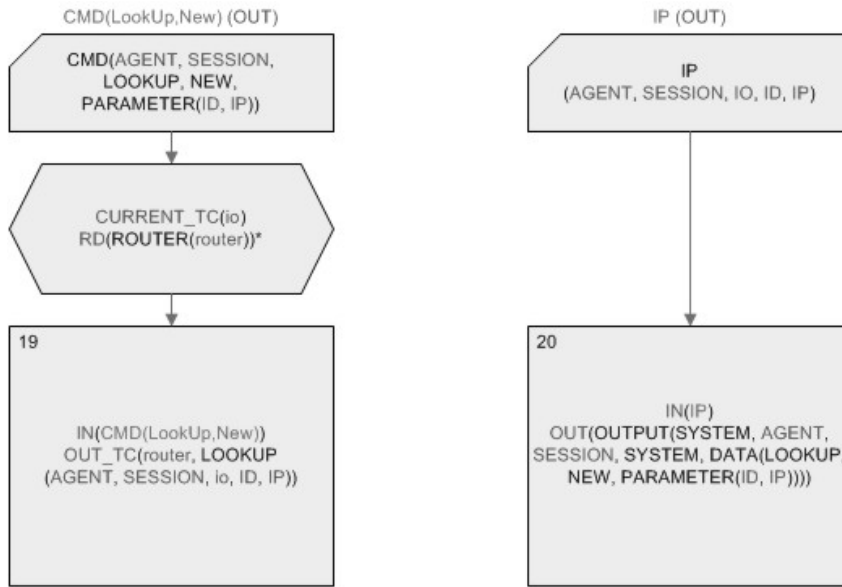


Figura 2.23: Richiesta di LookUp, reinstradamento, risposta al LookUp

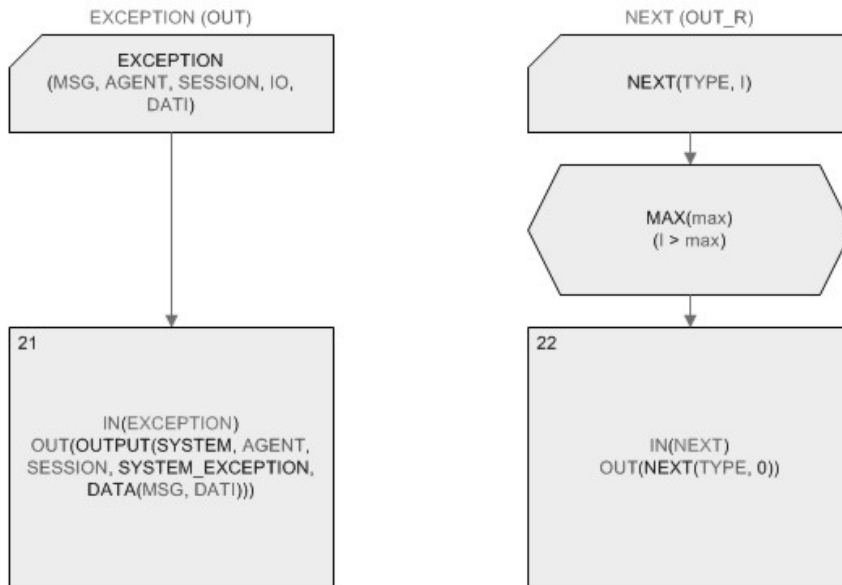


Figura 2.24: Reinstradamento di eccezioni e gestione massimo per la tupla **next**

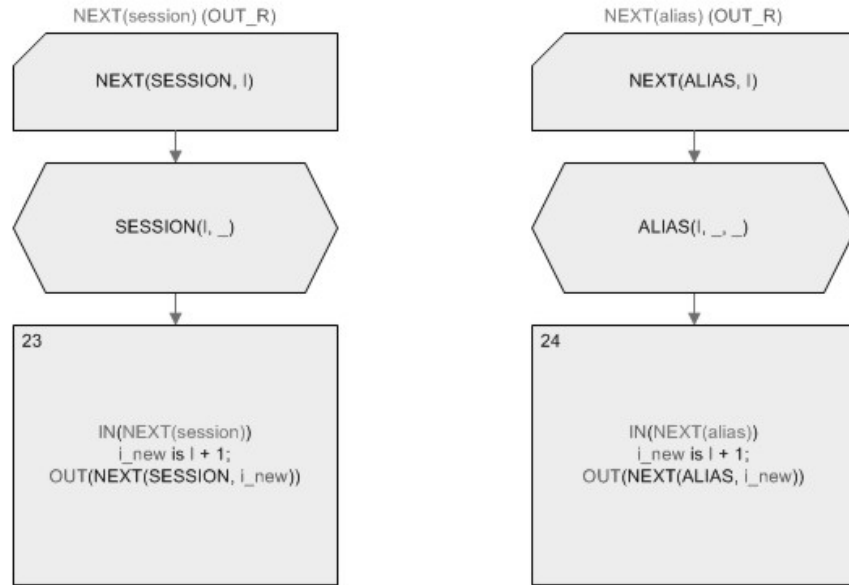
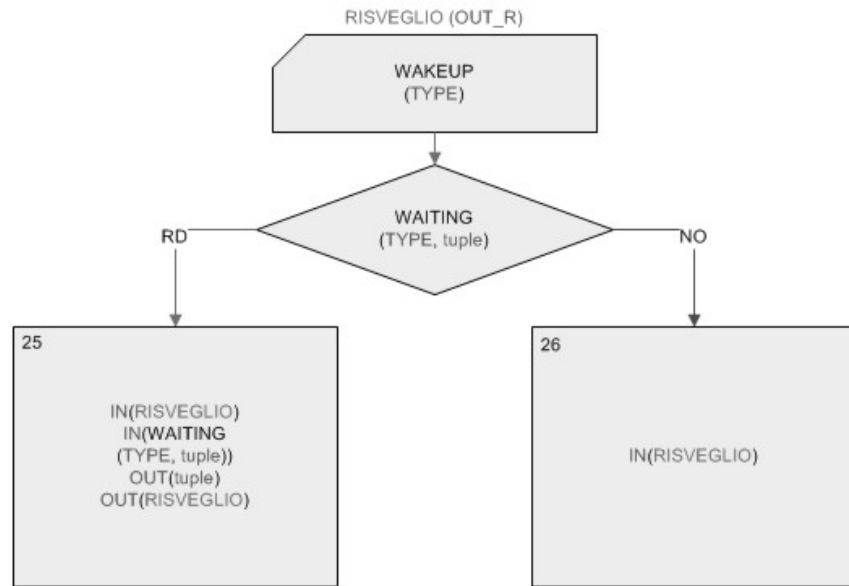
Figura 2.25: Gestione validità della tupla **next**

Figura 2.26: Risveglio tuple in attesa

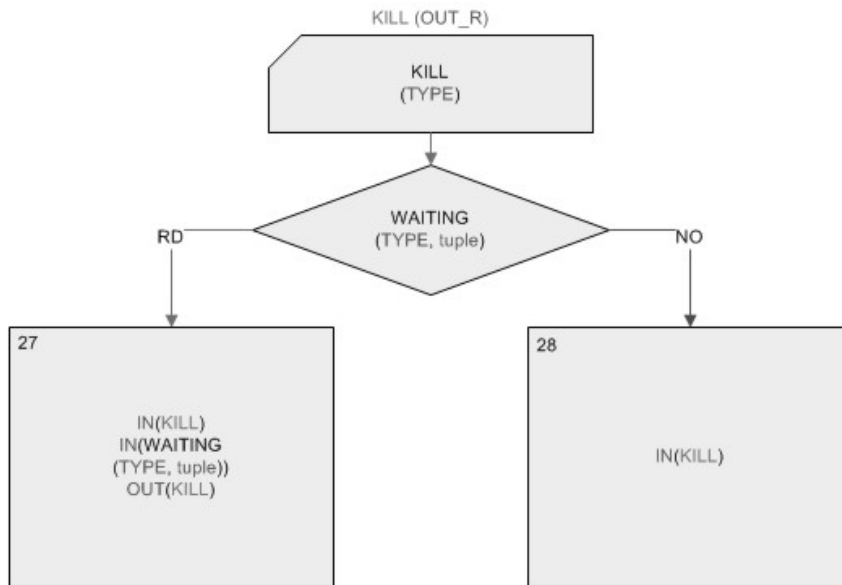


Figura 2.27: Eliminazione tuple in attesa

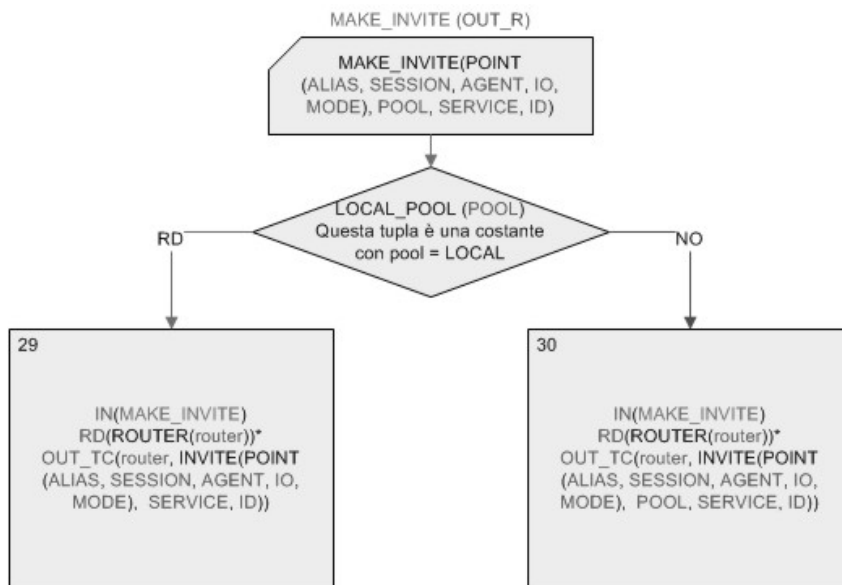


Figura 2.28: Creazione di un invito

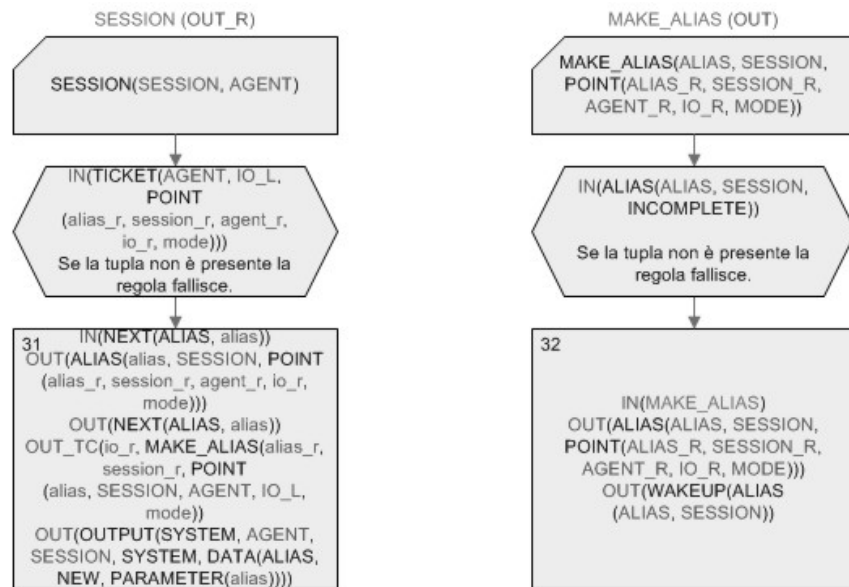


Figura 2.29: Completamento connessione su invito

2.6 Agent

L'Agent è la base di ogni agente dello strato superiore ed espone opportunamente le api disponibili (Sezione 2.8). Esso rimane in attesa delle tuple di tipo **output** presenti nel suo AgentIO, le legge e le traduce in eventi. Quando un agente vuole mandare un messaggio, deposita una tupla di tipo **input** nel suo AgentIO che provvederà al suo reinstradamento. Utilizzando un pattern opportuno (Appendice A), come si vede anche in Figura 2.2, sono state create due classi ed una interfaccia più precisamente: **CoreAgent**, **AbstractAgent**, **ICoreApi**.

2.6.1 CoreAgent

Estende la classe **Agent** di TuCSoN, implementa l'interfaccia **ICoreApi** e contiene un campo privato che punta ad un oggetto **AbstractAgent**.

```
public class CoreAgent<T> extends Agent
    implements ICoreApi<T> {
    private AbstractAgent<T> shell;
    ...
}
```

Tutte le classi utilizzate sono generiche perchè lo stato interno dell'agente è rappresentato da una classe definita successivamente, in questo modo lo sviluppatore può creare un agente con lo stato che preferisce. Quindi per utilizzare i metodi della classe **AbstractAgent** è sufficiente utilizzare tale puntatore, ad esempio per sollevare un evento basta scrivere: `shell.coreEvent(...)`;

2.6.2 AbstractAgent

Una classe astratta molto semplice, rappresenta la base di partenza per la creazione di nuovi agenti. Possiede alcuni metodi protetti astratti che dovranno essere implementati e un campo, anch'esso protetto, che rappresenta il puntatore alle api:

```
public abstract class AbstractAgent<T> {
    protected ICoreApi<T> core;
    ...
}
```

```
protected abstract T makeCoreState();

protected abstract void coreEvent(CoreEvent event);

protected abstract void coreException(CoreException ex);
}
```

Similmente alla classe `CoreAgent`, per accedere alle api basta usare questo puntatore, ad esempio: `core.newSession()`. I metodi astratti sono gli unici che lo sviluppatore deve obbligatoriamente implementare. Sono semplici: `makeCoreState()` non è altro che un factory per lo stato interno dell'agente; `coreEvent(CoreEvent event)` è il gestore degli eventi ed in fine `coreException(CoreException ex)` è il gestore delle eccezioni.

2.6.3 ICoreApi

Questa interfaccia contiene i metodi che sono stati messi a disposizione agli sviluppatori per lo strato superiore, verranno analizzati nella Sezione 2.8.

2.6.4 Comportamento

Il comportamento di questo componente in fase di input è il seguente:

1. Nell'AgentIO viene deposita una tupla
`output(FROM, AGENT, SESSION, TYPE, DATA)`
2. Il CoreAgent la preleva e solleva un evento o una eccezione
3. L'AbstractAgent gestisce l'evento e può generare comandi o risposte verso l'AgentIO attraverso le api (Sezione 2.8)

2.7 Meccaniche interne

All'interno degli artefatti avvengono trasformazioni che non sono state prese in esame nelle sezioni precedenti, ma verranno ora affrontate per completezza.

2.7.1 End Point

L'end point è una tupla che viene utilizzata in diverse regole, identifica il capo di una connessione logica fra due agenti:

point(*ALIAS,SESSION,AGENT,IO,MODE*)

ALIAS: alias dell'agente che ha generato la richiesta

SESSION: sessione dell'agente

AGENT: id dell'agente che ha generato la richiesta

IO: id del IO dell'agente

MODE: modalità di utilizzo

Descrizione:

Questa tupla rappresenta un end point

2.7.2 Debug

Per facilitare le operazioni di debug è stato creato un sistema di regole e di agenti di supporto che permettono la generazione del log relativo alla sequenza di attivazione delle regole. Questo sistema consiste nell'aggiunta di una postilla in coda ad ogni regola attraverso il metodo `debugRule(int, string)` della classe `AbstractCentre` essa infatti aggiunge il seguente codice ReSpecT:

```
...
current_time(DEBUG_TIME),
out_r(debug(DEBUG_TIME,id, num, tuple))
).
```

che non fa altro che ottenere l'orario corrente e creare una tupla di debug contenente tale orario ed altri dati, seguendo il seguente schema:

debug(*TIME, ID, NUM, BODY*)
Mittente: AbstractCentre

TIME: orario di attivazione della regola

ID: centro di tuple

NUM: numero della regola

BODY: tupla che l'ha innescata

Descrizione:

Tupla di debug

Essa viene consumata da un agente **Debugger** che genera il log. Se la funzione statica **AbstractCentre.setDebug**(bool) è settata a true; vengono aggiunti sia questi agenti ausiliari sia le relative modifiche alle regole; in questo modo nel suo normale funzionamento, l'infrastruttura non viene appesantita da queste aggiunte.

2.7.3 Try e Catch

Il SystemIO è l'interfaccia fra i nodi, quindi mette a disposizione le funzionalità necessarie per tale interazione. Una di queste, è la Try e Catch similmente alla gestione delle eccezioni questa funzionalità permette l'invio di un messaggio dal nodo locale ad uno remoto e ne gestisce le eventuali eccezioni. Per utilizzare questa funzione si deve inserire una tupla di tipo:

try(*AGENT, SESSION, IO, IP, MSG*)

AGENT: agente che ha generato la richiesta

SESSION: sessione dell'agente

IO: identificativo del IO dell'agente

IP: indirizzo remoto da contattare

MSG(tuple): tupla che si vuole inoltrare

Descrizione:

Rappresenta la richiesta di inoltro della tupla MSG verso il SystemIO di un nodo remoto

L'inserimento di questa tupla innesca una serie di reazioni. Il funzionamento normale comporta il trasferimento del messaggio incapsulato in una tupla **incoming** al Router remoto.

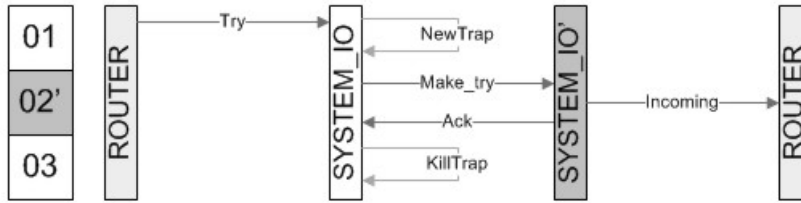


Figura 2.30: Funzionamento normale

incoming(*IP,LOCAL_IP,MSG*)

Destinatario: RouterIO

La sequenza delle regole attivate è la seguente:

- Regola 01 del SystemIO local (Figura 2.3)
- Regola 02 del SystemIO remoto (Figura 2.3)
- Regola 03 del SystemIO local (Figura 2.4)

Invece, nel caso si verifichi il mancato recapito del messaggio o della conferma, si ha la generazione di una eccezione.

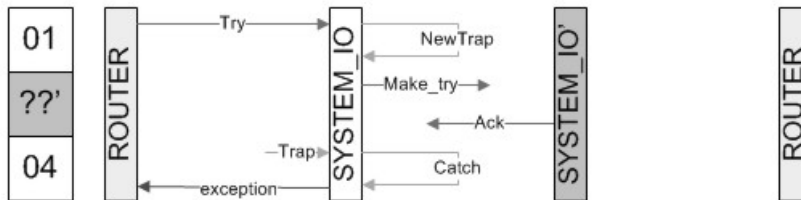


Figura 2.31: Funzionamento in caso di eccezione

exception(*broken,AGENT,SESSION,IO,MSG*)

Destinatario: RouterIO

Poichè la trap non viene disattivata, innesca la regola 04:

- Regola 01 del SystemIO local (Figura 2.3)
- Regola ?? qualcosa va storto

- Regola 04 del SystemIO local (Figura 2.4)

Questa funzionalità è presente anche nell'AgentIO in forma leggermente diversa; essa infatti viene utilizzata automaticamente nella comunicazione fra agenti remoti.

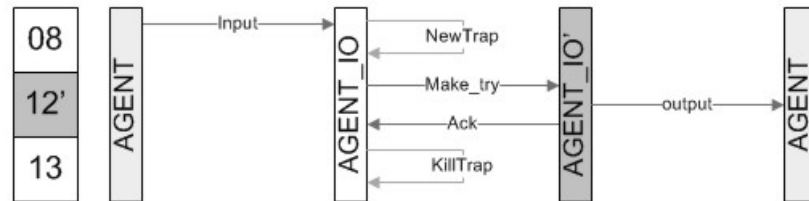


Figura 2.32: Trasferimento di un messaggio fra agenti remoti

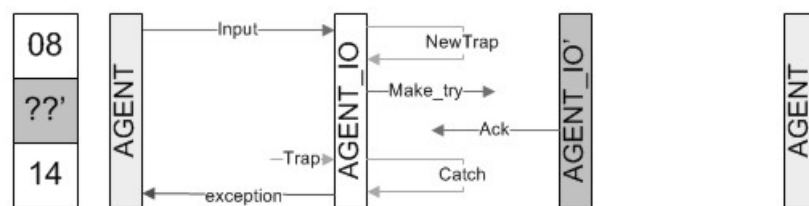


Figura 2.33: Trasferimento di un messaggio fra agenti remoti con eccezioni

2.7.4 Indici e Next

La gestione degli indici in ReSpecT risulta un pò complicata nel AgentIO essa necessita dell'introduzione di alcune regole e delle seguenti tuple di stato:

max(MAX) #0-1

MAX: la costante 2147483645

Descrizione:

Identifica il massimo numero che può essere usato come identificatore, se non è presente non c'è nessun limite

next (<i>alias, I</i>) # ¹ I: id valido Descrizione: Il prossimo identificatore valido per gli alias
next (<i>session, I</i>) # ¹ I: id valido Descrizione: Il prossimo identificatore valido per le sessioni

Queste tuple vengono supportate da alcune regole: la regola 22 (Figura 2.24) gestisce il massimo, mentre le regole 23 e 24 (Figura 2.25) controllano la validità dell'indice. Dopodichè ogni volta che si vuole utilizzare un indice, lo si recupera dalla tupla di stato che v'è eliminata e poi reinserita, permettendo così alle regole di aggiornarla opportunamente. Questa procedura viene eseguita nelle regole seguenti:

- Regola 01 (Figura 2.15)
- Regola 03 (Figura 2.16)
- Regola 05 (Figura 2.17)
- Regola 31 (Figura 2.29)

2.7.5 Waiting

Nelle tuple di **waiting** vengono incapsulate le tuple che, una volta inserite, non possono essere smistate in quanto richiedono una tupla di stato che non è ancora presente:

waiting (<i>ID, TYPE, MSG</i>) # ^{0-N} ID: identificativo univoco TYPE(tuple): evento atteso MSG(Tuple): messaggio Descrizione: Messaggio in attesa

Queste tuple rimangono in attesa che lo stato interno ne permetta lo smistamento; il modificarsi dello stato si notifica attraverso l'emissione di una tupla di **wakeup**, la quale, attivando una specifica regola (Figura 2.13 e Figura 2.26), ripropone tutte le tuple in attesa di tale stato.

wakeup(*TYPE*)

TYPE(tuple): evento atteso

Descrizione:

Evento: cambiamento di stato

Nello specifico, nella classe RouterIO, questo sistema viene utilizzato per gestire le richieste di risorse locali o remote ancora non registrate; mentre nell'AgentIO viene utilizzato per accodare i messaggi relativi agli alias incompleti. In quest'ultimo caso è stata aggiunta una coppia di regole (Figura 2.27) che permettono l'eliminazione delle tuple in attesa del completamento di un alias grazie alla tupla **kill**, utilizzata quando il completamento dell'alias non è possibile.

kill(*TYPE*)

TYPE(tuple): evento atteso

Descrizione:

Elimina tutte le tuple in attesa di un dato evento

2.7.6 Invito, Protocollo di mediazione

Per poter comunicare, due agenti devono scambiarsi alcune informazioni e sulla base di esse creare degli alias. Tale creazione non è un procedimento immediato ma necessita di alcuni passi. Primo, dobbiamo distinguere due casi:

1. I due agenti risiedono nello stesso nodo.
2. I due agenti NON risiedono nello stesso nodo.

Secondo, dividiamo il processo in varie fasi:

- Fase 1: Richiesta (Figura 2.34) Attraverso il metodo **newAlias**(String service): l'agente richiede la creazione di

un alias relativo ad un agente locale che svolge un determinato servizio (nel caso remoto viene utilizzato il metotodo:

`newAlias(String pool, String service)`, `pool` identifica il nodo/utente dove è localizzato il servizio richiesto). Questa funzione deposita nell'AgentIO una tupla del seguente tipo: `cmd` (vedi Sezione 2.5.2 tupla `cmd(...,alias,new,...)`) questa tupla innesca la regola 03 (Figura 2.16) che: crea un alias incompleto; crea un messaggio di ritorno per l'agente e crea una tupla `make_invite` che viene consumata dalla regola 29 (nel caso remoto dalla regola 30) (Figura 2.28), che a sua volta genera una tupla `invite`.



Figura 2.34: Richiesta

- Fase 2: Localizzazione risorsa (Figura 2.35) Questa fase è presente solo nel caso remoto, in quanto, in quello locale, il router conosce già il servizio e può reinstradare direttamente la richiesta. Nel caso remoto non è così banale: inizialmente la regola 10 (Figura 2.10) converte l'invito e lo passa alla regola 13 (Figura 2.12) che, utilizzando la funzionalità TryCatch (Sezione 2.7.3) del SystemIO, tenta una Try verso il nodo remoto che se va a buon fine recapita una tupla `incoming` nel RouterIO remoto che si occuperà di smistare la richiesta verso l'agente corretto.
- Fase 3: Completamento (Figura 2.36) Questa è la fase finale del processo: si innesca in modo leggermente differente se la richiesta è locale o remota, ma, a parte la regola iniziale (regola 08 Figura 2.9 per il caso remota e regola 09 Figura 2.10 per il caso locale), gran parte è comune ad entrambi i casi. La regola iniziale propone una tupla di invito locale, la regola 11 (Figura 2.11) la raccoglie e la reinstrada verso l'AgentIO di destinazione.

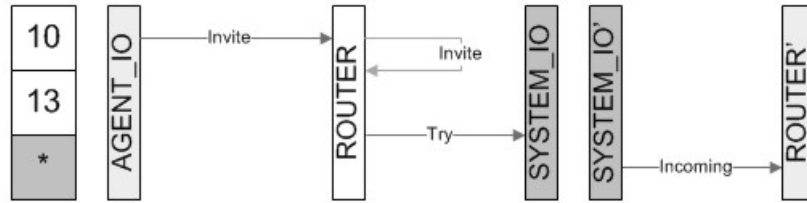


Figura 2.35: Localizzazione remota

Qui, grazie alla regola 05 (Figura 2.17) viene creato un ticket con le informazioni per la creazione dell'alias, successivamente viene creata una nuova sessione per l'agente a cui viene notificata tale creazione. La creazione di una nuova sessione e la presenza di una tupla **ticket**, fa scattare la regola 31 (Figura 2.29) che crea un alias completo. Dopo aver notificato all'agente la presenza del nuovo alias deposita nell'AgentIO remoto una tupla **Make_Alias** che a sua volta attiva la regola 32 (Figura 2.29). Quest'ultima sostituisce l'alias parziale con un alias completo.

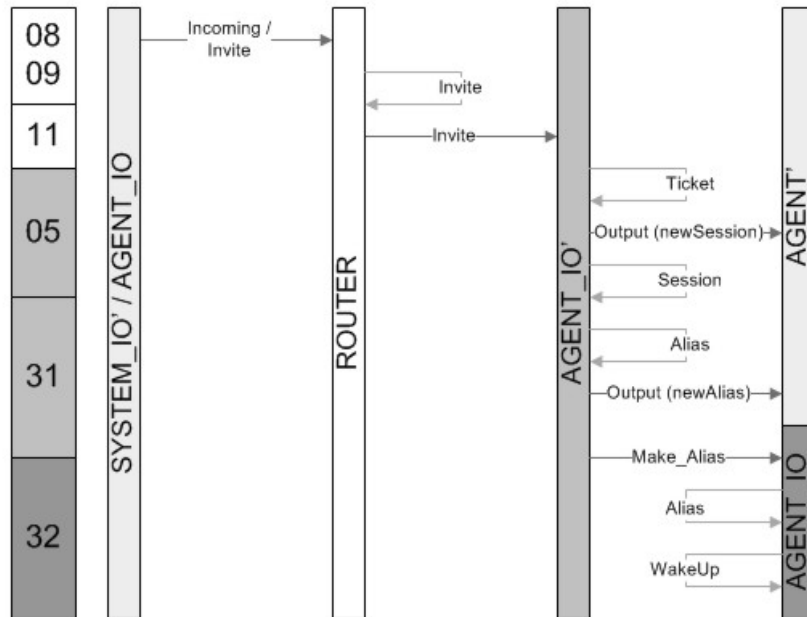


Figura 2.36: Completamento

Ora l'alias è completo da entrambe le parti ed è completamente funzionale quindi vengono risvegliate le eventuali tuple in attesa dell'alias. Questa trattazione, per motivi di chiarezza espositiva, non ha affrontato il caso della creazione di un alias verso una specifica sessione già esistente, che comunque presenta minime differenze.

2.8 Api

Questa sezione tratta i metodi resi disponibili dall'interfaccia `ICoreApi`, questi metodi permettono di interagire efficacemente con il core.

- **Alias (class):**
Gli Alias rappresentano gli interlocutori. Vengono creati attraverso in metodo `newAlias(...)` e utilizzati nella comunicazione con i metodi: `throwException(...)` e `fireEvent(...)`.
- **getAgentId:**
Ritorna l'id dell'agente.
Sintassi:
`String getAgentId()`
- **getServiceId:**
Ritorna il tipo di servizio offerto dall'agente.
Sintassi:
`String getServiceId()`
- **getCurrentSession:**
Restituisce l'identificativo della sessione corrente.
Sintassi:
`long getCurrentSession()`
- **getState:**
Restituisce lo stato interno dell'agente relativo alla sessione corrente.
Sintassi:
`T getState()`
- **start:**
Attiva l'agente e viene richiamato automaticamente all'atto della

creazione.

Sintassi:

```
void start()
```

- **stop:**

Distrugge l'agente

Sintassi:

```
void stop()
```

- **newSession:**

Crea una nuova sessione.

Sintassi:

```
void newSession()
```

- **delSession:**

Distrugge la sessione corrente.

Sintassi:

```
void delSession()
```

- **newAlias:**

Crea un nuovo alias di una risorsa locale

Sintassi:

```
Alias newAlias(String service)
```

```
Alias newAlias(String service, int id) (id: sessione)
```

Crea un nuovo alias di una risorsa remota.

Sintassi:

```
Alias newAlias(String pool, String service)
```

```
Alias newAlias(String pool, String service, int id) (id:  
sessione)
```

Questa primitiva fa riferimento al Protocollo di mediazione (Sezione 2.7.6).

- **delAlias:**

Distrugge un alias.

Sintassi:

```
void delAlias(Alias a)
```

- **newPool:**

Registra un nuovo contenitore di risorse.

Sintassi:

```
void newPool(String poolID, String ip, int port)
```

- **delPool:**

Distrugge un descrittore relativo ad un contenitore di risorse.

Sintassi:

```
void delPool(String poolID)
```

- **lookUp:**

Richiede il servizio di lookUp.

Sintassi:

```
void lookUp(String id, String ip)
```

- **fireEvent:**

Genera un evento verso un determinato alias.

Sintassi:

```
void fireEvent(Alias to, Object event)
```

- **fireSelfEvent:**

Genera un evento verso la sessione **id** di se stesso

Sintassi:

```
void fireSelfEvent(long id, Object event)
```

- **throwException:**

Lancia una eccezione.

Sintassi:

```
void throwException(Alias to, Object exception)
```

2.8.1 Precisazioni

Grazie alla libreria XStream, che viene utilizzata per serializzare gli oggetti in XML, si possono passare alle primitive di comunicazione (`fireEvent`, `fireSelfEvent` e `throwException`) oggetti qualsiasi, se però si vuole inviare un documento xml è stata creata un'apposita classe chiamata `JXMLDocument`, mentre per la validazione è fornita la classe `JXMLSchema`. Per una più approfondita trattazione sulla creazione di servizi rimandiamo il lettore ai capitoli successivi.

Capitolo 3

System

Lo strato di sistema raccoglie tutti i servizi offerti dall'infrastruttura, è composto da un insieme di agenti ognuno con un determinato compito. I servizi offerti da questi agenti sono identificati da un mestiere: ad esempio l'agente che gestisce la posta ha il compito di **Postman** (postino), questa scelta diverrà chiara nel Capitolo 4. Alcuni agenti hanno compiti particolari che li rendono necessari al corretto funzionamento dell'infrastruttura, in questo capitolo verranno descritti tali servizi ed alcuni servizi aggiuntivi.

3.1 Architettura dello strato

L'architettura risulta molto semplice poichè tutti gli agenti sono posti allo stesso piano ad eccezione dell'**Operator** che è l'unico agente che interagisce direttamente con l'utente, infatti gestisce le interazioni con l'interfaccia principale.



Figura 3.1: Architettura dello strato System

3.2 Agente Generico

Se ci poniamo all'esterno dell'infrastruttura, possiamo osservare gli agenti presenti nello strato di sistema, essi si possono suddividere in due tipologie: agenti interattivi ed agenti non interattivi. Un agente interattivo può ricevere degli eventi dall'esterno mentre un agente non interattivo può ricevere eventi solo dall'infrastruttura. I tipi di interazione con l'agente sono tre: Risposta immediata Figura 3.3; Risposta differita Figura 3.4; Evento esterno Figura 3.5. In una risposta immediata, l'agente riceve la richiesta e la serve immediatamente restituendone il risultato, un esempio pratico potrebbe essere una calcolatrice che riceve le operazioni da eseguire le esegue e ne ritorna il risultato. In una risposta differita, l'agente, ricevuta la richiesta, necessita dell'interazione con entità esterne. Quindi la risposta non viene presentata immediatamente, ma quando le entità esterne terminano la computazione viene inviato un auto evento che notifica all'agente il risultato, che a questo punto esso viene ritornato al richiedente. Un esempio di tale tipologia è il Postman che, dovendo interagire con un server di posta elettronica, deve affrontare dei ritardi imprevisti. Nell'evento esterno, l'agente riceve un auto evento generato dal gestore locale della risorsa esterna, tale evento viene gestito e può dare luogo all'emissione di un messaggio di output. Questo caso riguarda esclusivamente gli agenti interattivi, dei quali un esempio notevole è il gestore dell'interfaccia grafica (Hostess) che riceve gli eventi dal thread di swing. Tutti gli agenti estendono la classe astratta `AbstractAgent<OperatorState>` e mantengono un'istanza della relativa classe di stato per ogni sessione aperta; tale stato è perciò locale. Se un agente deve conservare uno stato globale, va gestito dallo sviluppatore come campo privato della classe agente; in Figura 3.2 ne è rappresentato un esempio.

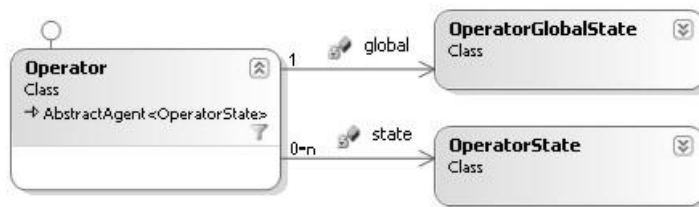


Figura 3.2: Schema UML dell'agente Operator

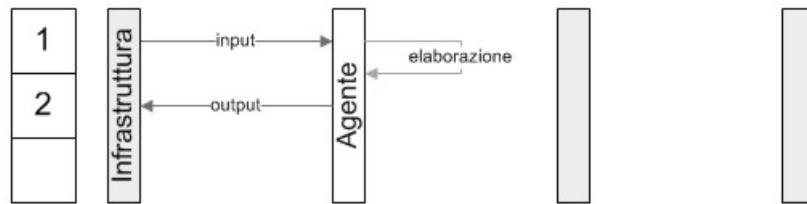


Figura 3.3: Risposta immediata

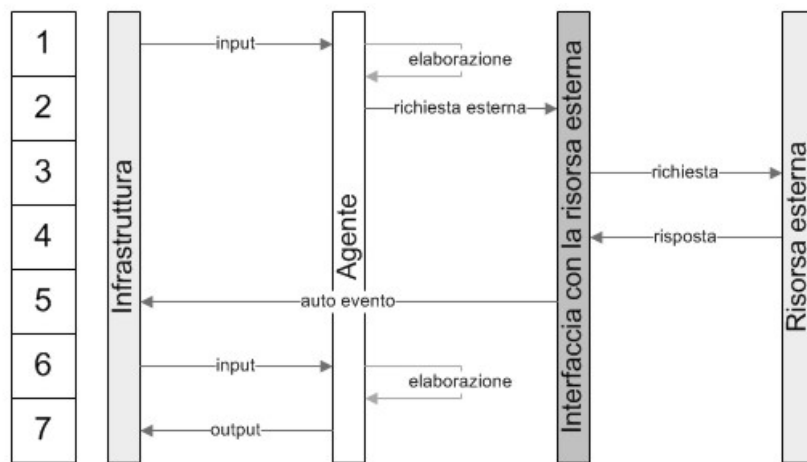


Figura 3.4: Risposta differita



Figura 3.5: Evento esterno

3.3 Postman (postino)

Questo agente non ha uno stato globale e si occupa della gestione del servizio di posta elettronica, esso infatti permette la ricezione e l'invio di messaggi e-mail grazie alla libreria JavaMail[2]. Dato che la posta viene utilizzata nel peer discovery dei gruppi privati, questo agente è necessario per il corretto funzionamento dell'infrastruttura.

3.3.1 Interazioni

Le interazioni avvengono attraverso lo scambio di messaggi XML, di seguito verranno mostrati i messaggi in ingresso e in uscita. Ricordiamo che è stata realizzata la classe **PostmanMsgFactory** per facilitare la creazione dei messaggi in ingresso, in ogni caso essi vengono validati con lo Schema Xml presente nel file **postman.xsd**.

- Invio di un messaggio
Invia il messaggio al destinatario, se incontra dei problemi con il server smtp ritorna un messaggio di eccezione.

```
<?xml version='1.0' encoding='UTF-8'?>
<postman>
  <send/>
  <cmd>send</cmd>
  <id>0</id>
  <email>mittente</email>
  <smtp>server smtp</smtp>
  <to>destinatario</to>
  <object>oggetto del messaggio</object>
  <body>corpo del messaggio</body>
</postman>
```

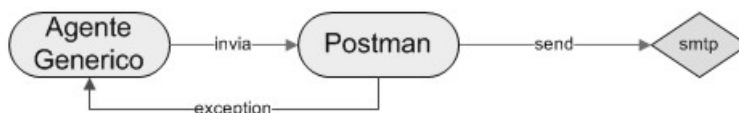


Figura 3.6: Invio di un messaggio di posta

- Ricezione messaggi
Richiede al server pop la lista dei messaggi relativi ad un account di posta.

```
<?xml version='1.0' encoding='UTF-8'?>
<postman>
  <receive/>
  <cmd>receive</cmd>
  <id>0</id>
  <email>indirizzo</email>
  <pop>server pop</pop>
  <password>password</password>
</postman>
```

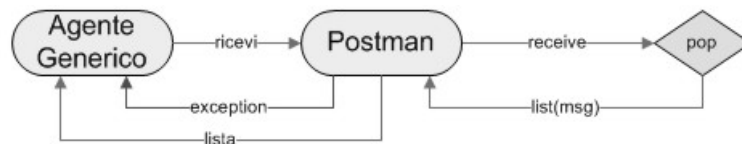


Figura 3.7: Ricezione dei messaggi di posta

La risposta dell'agente contiene la lista delle e-mail con i relativi corpi e informazioni aggiuntive:

```
<?xml version='1.0' encoding='UTF-8'?>
<postman>
  <to>destinatario</to>
  <id>0</id>
  <mail>
    <from>mittente</from>
    <object>oggetto del messaggio</object>
    <date>data</date>
    <body>corpo del messaggio</body>
    <mailId>identificatore</mailId>
  </mail>
  <mail>...</mail>
  <mail>...</mail>
</postman>
```

- Rimozione messaggi

Richiesta di eliminazione di uno o più messaggi:

```
<?xml version='1.0' encoding='UTF-8'?>
<postman>
  <remove/>
  <cmd>remove</cmd>
  <id>0</id>
  <email>indirizzo</email>
  <pop>server pop</pop>
  <password>password</password>
  <mailId>id dell'e-mail da eliminare</mailId>
  <mailId>...</mailId>
</postman>
```



Figura 3.8: Rimozione di messaggi di posta

3.4 Hostess

Questo agente non ha uno stato globale e si occupa della gestione delle interfacce grafiche, infatti si occupa della creazione, aggiornamento e interazione di tutte le GUI utilizzate dagli altri agenti. Data la sua particolare funzione questo agente è necessario per il corretto funzionamento dell'infrastruttura.

3.4.1 Interazioni

Le interazioni avvengono attraverso lo scambio di messaggi XML, di seguito verranno mostrati i messaggi in ingresso e in uscita. Ricordiamo che è stata realizzata la classe `HostessMsgFactory` per facilitare la creazione dei messaggi in ingresso, in ogni caso essi vengono validati con lo Schema Xml presente nel file `hostess.xsd`.

- Creazione di una interfaccia

Richiede la creazione di una interfaccia identificata dal nome della relativa classe; la quale deve estendere la classe base `AgentGui`.

```
<?xml version='1.0' encoding='UTF-8'?>
<hostess>
  <new/>
  <cmd>new</cmd>
  <gui>className</gui>
</hostess>
```



Figura 3.9: Richiesta di creazione di un'interfaccia

- Distruzione di un'interfaccia

Richiede la distruzione dell'interfaccia precedentemente creata.

```
<?xml version='1.0' encoding='UTF-8'?>
<hostess>
  <del/>
  <cmd>del</cmd>
</hostess>
```

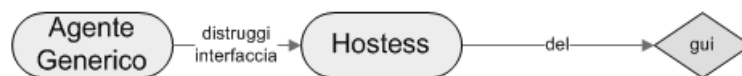


Figura 3.10: Richiesta di distruzione di un'interfaccia

- Modifica alla visibilità di una interfaccia

Richiede un cambio di visibilità dell'interfaccia; se `state=true` l'interfaccia diverrà visibile, se invece `state=false` l'interfaccia scomparirà.

```
<?xml version='1.0' encoding='UTF-8'?>
<hostess>
  <visibility/>
  <cmd>visibility</cmd>
  <state>state</state>
</hostess>
```

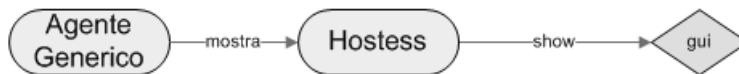


Figura 3.11: Richiesta di visualizzazione di un'interfaccia



Figura 3.12: Richiesta di occultamento di un'interfaccia

- Ridimensionamento di un'interfaccia

Richiede la modifica dello stato di un'interfaccia; `dim` può assumere i seguenti valori: `MINIMIZED = 0`, `MAXIMIZED = 1` e `NORMAL = 2`.

```
<?xml version='1.0' encoding='UTF-8'?>
<hostess>
  <state/>
  <cmd>state</cmd>
  <dimension>dim</dimension>
</hostess>
```

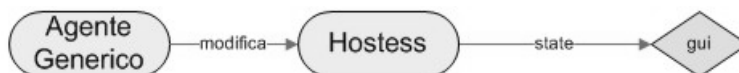


Figura 3.13: Richiesta di ridimensionamento di un'interfaccia

- Aggiornamento di un'interfaccia

Tutti i messaggi che non rientrano nei precedenti, vengono passati direttamente all'interfaccia, questo permette la massima libertà allo sviluppatore che può così comunicare come meglio crede con la propria interfaccia.



Figura 3.14: Richiesta di aggiornamento di un'interfaccia

- **Evento esterno**
Gli eventi generati dall'interfaccia vengono passati direttamente all'agente che ne ha fatto richiesta.

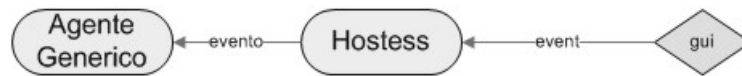


Figura 3.15: Evento dell'interfaccia grafica

3.5 Detective

Questo agente ha un semplicissimo stato globale che conserva l'indirizzo ip pubblico del nodo e si occupa di alcune funzionalità interne all'infrastruttura; per questo motivo questo agente è necessario per il corretto funzionamento dell'infrastruttura.

3.5.1 Interazioni

Le interazioni avvengono attraverso lo scambio di messaggi XML, di seguito verranno mostrati i messaggi in ingresso e in uscita. Ricordiamo che è stata realizzata la classe `DetectiveMsgFactory` per facilitare la creazione dei messaggi in ingresso, in ogni caso essi vengono validati con lo Schema Xml presente nel file **detective.xsd**.

- **Pivot Discovery**
Cerca in una lista di ip il primo indirizzo valido. Si intende valido l'indirizzo ip pubblico relativo ad un nodo **p4p** pronto. Questa procedura viene utilizzata per ottenere un peer valido dalla cache che verrà presentata nella Sezione 3.6.2.

```
<?xml version='1.0' encoding='UTF-8'?>
```

```

<detective>
  <pivotDiscovery/>
  <cmd>pivotDiscovery</cmd>
  <id>id</id>
  <name>name</name>
  <type>type</type>
  <ip>ip1#ip2#ip3</ip>
</detective>

```



Figura 3.16: Richiesta di Pivot Discovery

Il messaggio di risposta è il seguente:

```

<?xml version='1.0' encoding='UTF-8'?>
<detective>
  <id>id</id>
  <cmd>pivotDiscovery</cmd>
  <name>name</name>
  <type>type</type>
  <ip>ipPivot</ip>
</detective>

```

- Service Discovery

Richiesta di generazione della lista di servizi (add-on) installati. I servizi sono corredati da un manifesto (`manifest.plugin`) in xml che descrive l'add-on. Questo file verrà mostrato nella Sezione 6.4. Questa procedura cercherà le occorrenze del manifesto nelle sotto cartelle delle directory indicate nella lista.

```

<?xml version='1.0' encoding='UTF-8'?>
<detective>
  <addonDiscovery/>
  <cmd>addonDiscovery</cmd>

```

```

<id>id</id>
<directory>dir1</directory>
<directory>dir2</directory>
<directory>dir...</directory>
</detective>

```



Figura 3.17: Richiesta di Service Discovery

Il messaggio di risposta è il seguente e contiene la lista degli add-on identificati:

```

<?xml version='1.0' encoding='UTF-8'?>
<detective>
  <id>id</id>
  <cmd>addonDiscovery</cmd>
  <resource>
    <name>name</name>
    <type>type</type>
    <version>version</version>
    <className>className</className>
    <groupMenu>menu</groupMenu>
    <peerMenu>menu</peerMenu>
  </resource>
  <resource>
    ...
  </resource>
</detective>

```

- Ip Discovery
Richiesta dell'indirizzo pubblico del nodo corrente. Questa informazione è salvata nello stato globale dell'agente e risulta valida solo nel caso in cui il nodo abbia un indirizzo ip pubblico, oppure

in seguito alle operazioni di Pivot Discovery (se si identificano pivot esterni alla rete locale). Negli altri casi l'indirizzo ip salvato risulta essere quello locale, che quindi permette la comunicazione esclusivamente nella rete locale.

```
<?xml version='1.0' encoding='UTF-8'?>
<detective>
  <ipDiscovery/>
  <cmd>ipDiscovery</cmd>
  <id>id</id>
</detective>
```

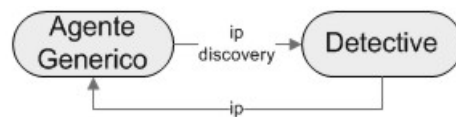


Figura 3.18: Richiesta di Ip Discovery

Il messaggio di risposta è il seguente:

```
<?xml version='1.0' encoding='UTF-8'?>
<detective>
  <id>id</id>
  <cmd>ipDiscovery</cmd>
  <ip>localIp</ip>
</detective>
```

3.6 Operator (centralinista)

Si tratta sicuramente dell'agente più complesso fra quelli di base, esso infatti coordina le operazioni di start-up dell'infrastruttura, gestisce gli utenti e il loro log-in, gestisce i gruppi e gli add-on ed infine gestisce l'interfaccia di controllo principale. La procedura maggiormente complicata è lo Start-Up (Sezione 3.6.1) essa richiede l'interazione con tutti gli agenti di base e solo alla sua terminazione viene visualizzata l'interfaccia di controllo. Questo agente utilizza uno stato globale implementato nella classe `OperatorGlobalState` che conserva le informazioni

relative ai gruppi e agli add-on; per il suo corretto funzionamento richiede i seguenti agenti: Postman, Detective e Hostess.

3.6.1 Start-Up

Nella Figura 3.19 sono state schematizzate le interazioni che vengono effettuate durante questa procedura, eccole presentate sottoforma di passi per rendere più agevole la loro comprensione:

1. Completato l'avviamento del core viene visualizzata la finestra di log-in;
2. L'utente inserisce user e password effettuando il log-in;
3. L'Operator recupera dal file: `p4pData/user/usrename.user` le informazioni relative ai gruppi a cui l'utente appartiene;
 - Richiede al Postman la cache dei gruppi privati;
 - Richiede al Detective il PivotDiscovery per la cache locale (gruppi pubblici e gruppo locale);
4. Quando il Postman ritorna la cache dei gruppi privati, l'operator ne richiede il PivotDiscovery al Detective;
5. Quando il Detective ha teminato il PivotDiscovery l'Operator interroga il Pivot richiedendogli la lista dei peer e aggiornando il suo stato interno;
6. Completate tali interrogazioni, l'Operator chiede al Detective di identificare gli add-on installati (Service Discovery) e aggiorna;
7. Poi chiede l'Ip al Detective (Ip Discovery);
8. Ottenuto il suo Ip notifica il suo stato a tutti i peer;
9. Infine visualizza l'interfaccia principale;

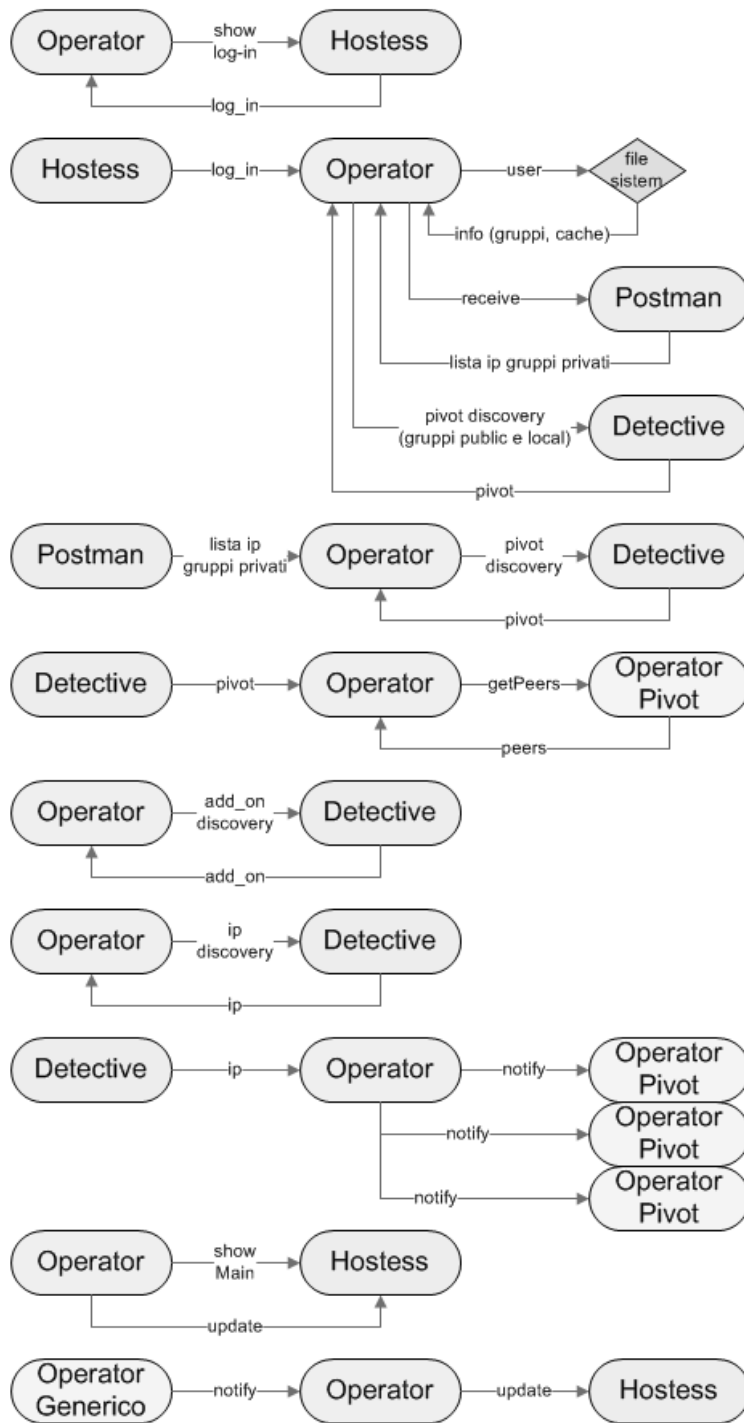


Figura 3.19: Start-Up

3.6.2 Interazioni

Oltre allo start-up e all'interazione con l'utente, effettuata tramite l'interfaccia grafica, questo agente offre alcuni servizi utilizzabili dagli altri agenti grazie allo scambio di messaggi XML; di seguito verranno mostrati i messaggi in ingresso e in uscita. Ricordiamo che è stata realizzata la classe `OperatorMsgFactory` per facilitare la creazione dei messaggi in ingresso, in ogni caso essi vengono validati con lo Schema Xml presente nel file `operator.xsd`.

- Informazioni dell'utente
Richiede i dati relativi all'utente correntemente loggato.

```
<?xml version='1.0' encoding='UTF-8'?>
<operator>
  <getUserInformation/>
  <cmd>getUserInformation</cmd>
</operator>
```



Figura 3.20: Richiesta delle informazioni dell'utente

Questa è la risposta che viene generata:

```
<?xml version='1.0' encoding='UTF-8'?>
<operator>
  <userInformation/>
  <cmd>userInformation</cmd>
  <userId>userId</userId>
  <userName>userName</userName>
</operator>
```

- Get Contacts
Richiede la lista dei peer relativi ad un gruppo.

```
<?xml version='1.0' encoding='UTF-8'?>
<operator>
  <getContacts/>
  <cmd>getContacts</cmd>
  <group>group</group>
</operator>
```

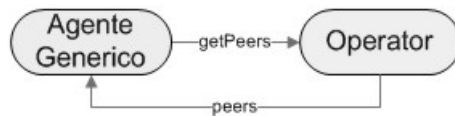


Figura 3.21: Richiesta della lista degli utenti di un gruppo

Questa è la risposta che viene generata:

```
<?xml version='1.0' encoding='UTF-8'?>
<operator>
  <contactsList/>
  <cmd>contactsList</cmd>
  <group>groupName</group>
  <type>groupType</type>
  <contact><name>name</name><id>id</id><ip>ip</ip></contact>
  <contact>...</contact>
</operator>
```

- Select Contact

Richiede la visualizzazione di una finestra per la selezione di un contatto.

```
<?xml version='1.0' encoding='UTF-8'?>
<operator>
  <selectContact/>
  <cmd>selectContact</cmd>
  <group>group</group>
</operator>
```

Nel messaggio di risposta viene comunicato l'id del contatto selezionato dall'utente

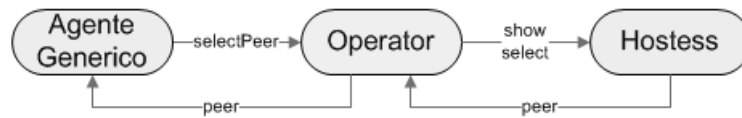


Figura 3.22: Richiesta di selezione di un contatto

```

<?xml version='1.0' encoding='UTF-8'?>
<operator>
  <pool>pool</pool>
</operator>

```

- Confirmation

Richiede la visualizzazione di una finestra di conferma che mostra un messaggio.

```

<?xml version='1.0' encoding='UTF-8'?>
<operator>
  <confirmation/>
  <cmd>confirmation</cmd>
  <msg>msg</msg>
</operator>

```

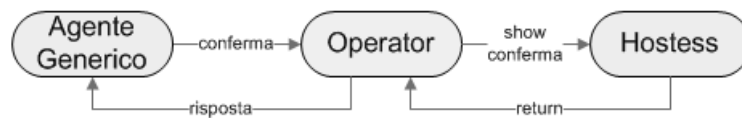


Figura 3.23: Richiesta di conferma

Nel messaggio di risposta viene comunicata la scelta effettuata

```

<?xml version='1.0' encoding='UTF-8'?>
<operator>
  <confirmation>cmd</confirmation>
</operator>

```

3.6.3 Gestione dei Gruppi

I gruppi rappresentano insiemi di utenti che potenzialmente vogliono interagire, tali utenti si possono raggruppare secondo varie caratteristiche per esempio: la cultura (lingua o nazionalità), la conoscenza personale (amicizia), il luogo nel quale si riuniscono, gli interessi (per es. astronomia, sport, ecc.) e i rapporti di lavoro. Per gestire tutti questi casi sono stati creati tre tipi di gruppo:

- Gruppo Pubblico: aggregazione per interessi comuni;
- Gruppo Locale (o Protetto): aggregazione per luogo;
- Gruppo Privato: aggregazione per rapporti interpersonali;

Ognuno di questi tipi di gruppo ha un sistema di peer discovery e modalità di accesso differenti: I Gruppi Pubblici sono di grandi dimensioni, necessitano di alcuni peer con un indirizzo ip pubblico e statico e, l'accesso non ha restrizioni. I Gruppi Locali sono di medie o piccole dimensioni, necessitano di una rete privata e per accedervi è necessario essere collegati a tale rete. I Gruppi Privati sono di piccole dimensioni, necessitano di un account di posta elettronica e l'accesso è ristretto a chi conosce account e password del gruppo. Le informazioni relative ai gruppi a cui un utente appartiene vengono salvate nel file user e vengono utilizzate al momento del log-in per accedere ai vari gruppi.

Ulteriori informazioni relative all'Operator verranno esposte nel Capitolo 5.

3.7 Helmsman (timoniere)

Questo agente aggiuntivo permette di interfacciare l'infrastruttura **p4p** alla tecnologia web, esso infatti crea un semplice server http locale che, interrogato da un navigatore, genera chiamate verso gli agenti. Gli agenti possono registrare una risorsa (indirizzo) che diverrà raggiungibile attraverso il navigatore. Questo permette la creazione di interfacce alternative per i servizi: ad esempio se si vuole creare un agente che gestisce un forum risulterà sicuramente più agevole realizzare l'interfaccia utilizzando l'html invece delle finestre swing. Lo stato globale conserva la lista degli indirizzi registrati.

3.7.1 Interazioni

Le interazioni avvengono attraverso lo scambio di messaggi XML, di seguito verranno mostrati i messaggi in ingresso e in uscita. Ricordiamo che è stata realizzata la classe `HelmsmanMsgFactory` per facilitare la creazione dei messaggi in ingresso, in ogni caso essi vengono validati con lo Schema Xml presente nel file **helmsman.xsd**.

- Add Url

Registra una nuova risorsa web raggiungibile attraverso il seguente URL:`http://localhost:porta/url/`.

```
<?xml version='1.0' encoding='UTF-8'?>
<helmsman>
  <addUrl/>
  <cmd>addUrl</cmd>
  <url>url</url>
  <name>name</name>
  <note>note</note>
</helmsman>
```

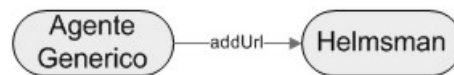


Figura 3.24: Registrazione di una nuova risorsa web

- Remove Url

Cancella una risorsa web registrata e raggiungibile attraverso il seguente URL:`http://localhost:porta/url/`.

```
<?xml version='1.0' encoding='UTF-8'?>
<helmsman>
  <removeUrl/>
  <cmd>removeUrl</cmd>
  <url>url</url>
</helmsman>
```



Figura 3.25: Rimozione di una nuova risorsa web

- Send Page
Invia una pagina al navigatore web.

```

<?xml version='1.0' encoding='UTF-8'?>
<helmsman>
  <sendPage/>
  <cmd>sendPage</cmd>
  <id>id</id>
  <page>text</page>
</helmsman>
  
```



Figura 3.26: Invio di una pagina web

- Send Error Page
Invia una pagina di errore al navigatore web.

```

<?xml version='1.0' encoding='UTF-8'?>
<helmsman>
  <sendErrorPage/>
  <cmd>sendErrorPage</cmd>
  <id>id</id>
  <code>code</code>
</helmsman>
  
```

Il codice di errore può assumere i seguenti valori relativi al protocollo HTTP[9]:

- 200 (OK) - La richiesta è stata servita.

- 400 Bad Request. - La richiesta non è comprensibile al server.
- 404 Not Found. - La risorsa richiesta non È stata trovata.
- 500 Internal Server Error. - Il server non è in grado di rispondere alla richiesta.



Figura 3.27: invio di una pagina web di errore

- Request Url
Richiesta proveniente dal navigatore web.

```

<?xml version='1.0' encoding='UTF-8'?>
<helmsman>
  <requestUrl/>
  <cmd>requestUrl</cmd>
  <id>id</id>
  <type>GET</type>
  <url>url</url>
  <parameter>parameter</parameter>
</helmsman>
  
```



Figura 3.28: Ricezione di una richiesta del navigatore

3.8 Worker (lavoratore)

Si tratta di un agente astratto che viene utilizzato come base per lo strato superiore, mette a disposizione nuove api e nuovi strumenti volti ad una comunicazione più naturale. Ora non si parla più di alias ma di nomi (semplici stringhe di testo) ed è stata automatizzata la gestione

delle topologie di rete basilari, dalle quali successivamente possono essere derivate tutte le altre.

Le topologie di rete gestite comprendono: le reti a maglia completamente connessa e quelle a stella (client server) entrambe realizzate attraverso due specifici costrutti le stanze (Room) e i teatri (Theater).

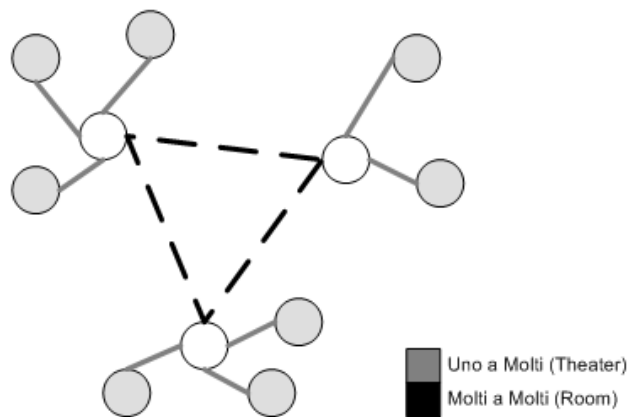


Figura 3.29: Rete a topologia mista

3.8.1 Interazioni

Le interazioni avvengono attraverso lo scambio di messaggi XML, di seguito verranno mostrati i messaggi in ingresso e in uscita. Ricordiamo che è stata realizzata la classe `WorkerMsgFactory` per facilitare la creazione dei messaggi in ingresso, in ogni caso essi vengono validati con lo Schema Xml presente nel file **worker.xsd**.

I messaggi che non vengono validati generano la chiamata del metodo astratto `reaction(...)`.

- **Make Start Up**
Inizializza l'agente. Questo messaggio viene inviato dall'operator quando istanzia un agente derivato dalla classe `AbstractWorker`.

```
<?xml version='1.0' encoding='UTF-8'?>
```

```

<worker>
  <makeStartUp/>
  <cmd>makeStartUp</cmd>
  <userId>userId</userId>
  <userName>userName</userName>
</worker>

```



Figura 3.30: Notifica di Start Up

- Make Invite Contact
Chiede al Worker di invitare un suo “simile” per iniziare una cooperazione.

```

<?xml version='1.0' encoding='UTF-8'?>
<worker>
  <makeInviteContact/>
  <cmd>makeInviteContact</cmd>
  <contactName>contactName</contactName>
  <contact>contactId</contact>
  <group>group</group>
</worker>

```

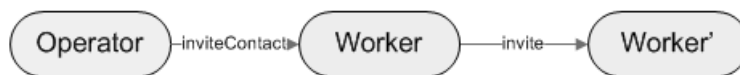


Figura 3.31: Richiesta di invito

Il Worker remoto riceverà il seguente messaggio, che completerà la procedura di invito.

```

<?xml version='1.0' encoding='UTF-8'?>
<worker>
  <invite/>

```

```

    <cmd>invite</cmd>
    <group>group</group>
  </worker>

```

- **Make Group Service**
Chiede al Worker di attivarsi per offrire il servizio locale relativo al gruppo.

```

<?xml version='1.0' encoding='UTF-8'?>
<worker>
  <makeGroupService/>
  <cmd>makeGroupService</cmd>
  <group>group</group>
</worker>

```



Figura 3.32: Richiesta di servizio locale

3.8.2 Api

Questa sezione tratta i metodi che vengono resi disponibili dall'interfaccia `IWorkerApi`, questi metodi permettono di interagire efficacemente con lo strato di sistema.

- **getId:**
Restituisce l'identificatore del worker corrente.
Sintassi:
`String getId()`
- **getType:**
Restituisce il tipo di worker.
Sintassi:
`String getType()`

- **getId:**
Restituisce l'identificatore dell'utente corrente.
Sintassi:
`String getId()`
- **getName:**
Restituisce il nome dell'utente corrente.
Sintassi:
`String getName()`
- **getState:**
Restituisce lo stato corrente.
Sintassi:
`T getState()`
- **iKnow:**
Ritorna VERO se il worker conosce il nome.
Sintassi:
`boolean iKnow(String name)`
- **rename:**
Rinomina un contatto.
Sintassi:
`String rename(String oldName, String newName)`
- **sayHello:**
Crea un contatto per un servizio locale oppure per un servizio remoto.
Sintassi:
`String sayHello(String service, String name)`
`String sayHello(String who, String service, String name)`
- **sayGoodbye:**
Distrugge la sessione corrente oppure distrugge il contatto who.
Sintassi:
`void sayGoodbye()`
`void sayGoodbye(String who)`
- **sayMsg:**
Dice il messaggio al contatto.

Sintassi:

```
void sayMsg(String who, Object msg)
```

- **openRoom:**

Apre una nuova stanza.

Sintassi:

```
void openRoom(String room)
```

- **enterInRoom:**

Entra in una stanza.

Sintassi:

```
void enterInRoom(String who, String room)
```

- **exitFromRoom:**

Esce da una stanza.

Sintassi:

```
void exitFromRoom(String room)
```

- **sayMsgInRoom:**

Dice il messaggio nella stanza.

Sintassi:

```
sayMsgInRoom(String room, Object msg)
```

- **openTheater:**

Apre un nuovo teatro.

Sintassi:

```
void openTheater(String theater)
```

- **enterInTheater:**

Entra in un teatro.

Sintassi:

```
void enterInTheater(String who, String theater)
```

- **exitFromTheater:**

Esce da un teatro.

Sintassi:

```
void exitFromTheater(String theater)
```

- **putOutFromTheater:**

Butta fuori un utente da un teatro. Solo l'utente creatore puo'

usare questo comando.

Sintassi:

```
void putOutFromTheater(String who, String theater)
```

- **sayMsgInTheater:**

Dice il messaggio nel teatro.

Sintassi:

```
void sayMsgInTheater(String theater, Object msg)
```


Capitolo 4

Service

Questo strato contiene gli agenti che permettono l'interazione con l'utente, essi infatti sono attivabili attraverso opportuni menù dell'interfaccia principale gestita dall'Operator. Come ogni agente aggiunto, essi, hanno un relativo file **manifest.plugin** che però a differenza degli agenti dello strato di sistema presenta almeno una stringa non vuota nel tag **groupMenu** o nel tag **peerMenu**. Queste stringhe vengono utilizzate per creare i menù contestuali; più precisamente selezionando un gruppo il menù visualizzato sarà composto da tutte le stringhe non nulle presenti nei tag **groupMenu**, mentre selezionando un contatto il menù sarà composto dalle stringhe non nulle presenti nei tag **peerMenu**. La sostanziale differenza fra queste due modalità è il tipo di interazione ad esempio se si vuole comunicare con un utente, grazie ad esempio al plugin di messaggiera istantanea (Messenger), si seleziona l'utente e il menù "chiacchera" mentre se si vuole creare una stanza per parlare (Chat Room) si seleziona il gruppo e il menù "chat room". In questo capitolo presenteremo i servizi **messenger** e **tictactoe** che sono due semplici esempi di agenti di cooperazione.

4.1 Messenger

La comunicazione è molto importante per la cooperazione e svariate applicazioni si sono proposte di facilitare questo tipo di interazione; inizialmente con comunicazioni differite come Mailing List, News Group e Forum; poi con le Chat Room, la Messaggiera Istantanea e

il Voice Over Ip. Questo agente svolge una duplice funzione, permette la comunicazione diretta fra due utenti (Messaggeria Istantanea) e permette la creazione di “stanze” (Chat Room) pubbliche per la comunicazione di gruppo.

4.1.1 Comunicazione privata

Clicchiamo con il tasto destro su un contatto, facendo apparire il menù contestuale (Figura 4.1) relativo ai servizi attivabili.

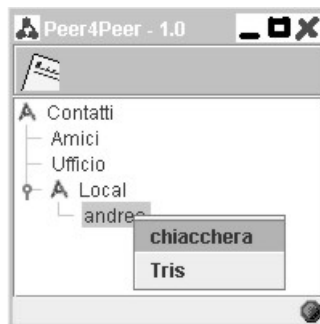


Figura 4.1: Menù contestuale relativo ai contatti

Selezioniamo da esso il menù “chiacchera” in questo modo attiviamo il plugin di messaggeria istantanea, la cui interfaccia è presentata in Figura 4.2.

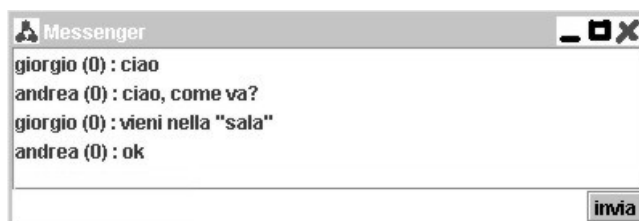


Figura 4.2: Messaggeria istantanea

Attraveso questa finestra possiamo comunicare in privato con il nostro interlocutore.

4.1.2 Comunicazione di gruppo

Clicchiamo con il tasto destro su un gruppo, facendo apparire il menù contestuale (Figura 4.3) relativo ai servizi attivabili.



Figura 4.3: Menù contestuale relativo ai gruppi

Selezioniamo da esso il menù “chat room” in questo modo attiviamo il plugin di chat. Appare, ora, l’interfaccia di Figura 4.4 e Figura 4.5 che gestisce la creazione e l’accesso delle stanze.



Figura 4.4: Gestione Chat Room (creazione)



Figura 4.5: Gestione Chat Room (accesso)

Più precisamente Giorgio crea la stanza “sala” (Figura 4.4) ed invita Andrea (Figura 4.2), il quale entra nella stanza (Figura 4.5). Poi i due aspettano l’arrivo di altri utenti (Figura 4.6).

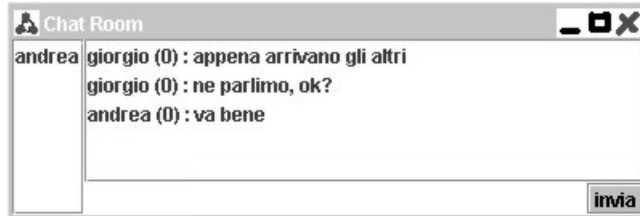


Figura 4.6: Una comunicazione di gruppo

Questo agente è molto semplice e può essere esteso aggiungendo varie funzionalità, ad esempio:

- Gestione degli smile;
- Registrazione (log) dei dialoghi;
- Miglioramenti grafici dell’interfaccia;
- Supporto per il Voice over ip (utilizzando un opportuno servizio dello strato System);

Anche senza le suddette funzioni rimane un semplice strumento per la comunicazione e un valido esempio per gli sviluppatori.

4.2 TicTacToe (Tris)

Una classica interazione fra utenti è il gioco, per questo motivo non poteva mancare un esempio di agente “ludico”. Il gioco Tic Tac Toe, conosciuto anche col nome Tris, è molto semplice ma ci dà la possibilità di osservare un altro esempio di plugin per lo strato Service.

4.2.1 Facciamo una partita

Clicchiamo con il tasto destro su un contatto, facendo apparire il menù contestuale (Figura 4.1) relativo ai servizi attivabili. Selezioniamo da



Figura 4.7: La scacchiera del Tris

esso il menù “tris” in questo modo attiviamo questo plugin. Appare, ora, l’interfaccia di gioco (Figura 4.7).

A turno i due giocatori fanno la loro mossa cliccando su una casella vuota ed in fine la partita termina quando uno dei giocatori mette in fila tre simboli uguali.

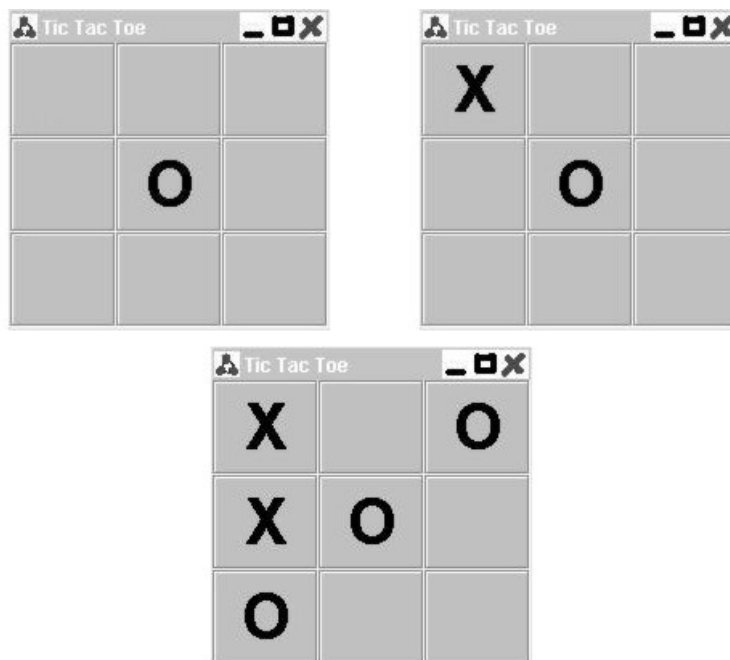


Figura 4.8: Una partita a Tris

Capitolo 5

Guida all'Uso



Peer for Peer è un mezzo di comunicazione, una piattaforma di cooperazione. L'interfaccia principale di questa applicazione è molto semplice da usare, essa infatti è simile a quella dei più comuni programmi di comunicazione testuale istantanea. Escludendo le interfacce dei plugin, ovviamente eterogenee, sono presenti cinque finestre:

- Log-In;
- Interfaccia Principale;
- Gestione utente;
- Richiesta di conferma;
- Selezione di un contatto;

Esse verranno mostrate in questo capitolo, che terminerà con la procedura di installazione e disinstallazione dei plugin.

5.1 Installazione

L'installazione di questa applicazione è consiste nel copiare il contenuto dell'archivio compresso in una cartella. Così facendo nella cartella di installazione troveremo le seguenti sottocartelle e file:

-  **lib**: contiene le librerie necessarie per il corretto funzionamento dell'applicazione;
-  **p4pData**: contiene le immagini della skin e i dati sugli utenti;
-  **plugins**: contiene il codice compilato dei plugin;
-  **gpl.txt**: licenza di utilizzo;
-  **p4p.jar**: l'applicazione;

5.2 Log-In

All'avvio, l'applicazione, mostra la finestra di log-in (Figura 5.1). L'utente può inserire l'username e password in suo possesso per accedere ai suoi dati, oppure può creare un nuovo account inserendone l'username e password e premendo il pulsante "New User", creando un nuovo utente che appartiene esclusivamente al gruppo locale. In entrambi i casi al termine di tale procedura questa finestra scompare lasciando il campo a quella principale.



Figura 5.1: Log-In

In caso di inserimento errato dell'username o della password verrà visualizzato un avviso nella barra di stato della finestra.

5.3 Interfaccia principale

Una rappresentazione ad albero mostra i contatti divisi per gruppo e si aggiorna quando un utente si collega o si scollega. Per attivare i vari plugin basta accedere ai menù contestuali, cliccando con il tasto destro del mouse sul nome di un gruppo o di un contatto.



Figura 5.2: Interfaccia principale

Infine un pulsante nella toolbar consente l'apertura della finestra di gestione utente.

5.4 Gestione utente

Cliccando sul pulsante della toolbar si accede alla gestione utente.



Figura 5.3: Gestione Utente

Questa interfaccia permette la creazione, l'importazione e l'esportazione dei descrittori dei gruppi. Tali descrittori, salvati come file

`nomefile.group`, possono essere condivisi con gli altri utenti che si desiderano invitare nel gruppo.

5.5 Richiesta di conferma

Una finestra che potete incontrare facilmente è il dialog di conferma. Si tratta generalmente di una domanda alla quale potete ri-

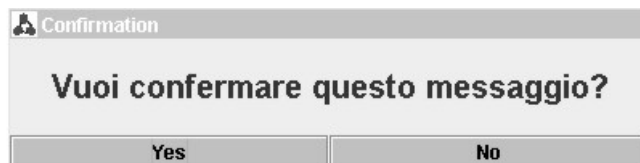


Figura 5.4: Finestra di conferma

spondere Sì oppure No, ad esempio una conferma di eliminazione o l'accettazione di una richiesta remota.

5.6 Selezione di un contatto

Alcuni plugin potrebbero richiedere una successiva selezione di un contatto, per esempio per invitarlo nella discussione corrente. Questa



Figura 5.5: Finestra di selezione di un contatto

finestra mostra un elenco dei contatti disponibili (relativi ad un dato gruppo), selezionando una riga e premendo il tasto "Ok" alla base della finestra si effettua la scelta.

5.7 Installare e disinstallare plugin

I plugin sono di due tipi e il loro codice compilato è contenuto in specifiche sottocartelle:

1. system: plugins/system/nomePlugin;
2. service: plugins/service/nomePlugin;

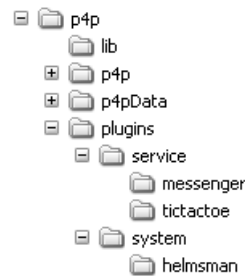


Figura 5.6: Cartelle plugin

Tale suddivisione corrisponde ai package e deve essere rispettata. Per installare un plugin è sufficiente copiare la relativa cartella plugins nella cartella di installazione. La disinstallazione è altrettanto semplice, infatti, basta eliminare la sottocartella relativa al plugin e riavviare l'applicazione.

Capitolo 6

Guida allo Sviluppo



Peer for Peer è un mezzo di comunicazione, una piattaforma di cooperazione. Lo sviluppo di plugin per questa infrastruttura richiede la conoscenza del linguaggio Javae poco altro. In questo capitolo verranno presentate le linee guida per la creazione di plugin corredate di alcuni esempi.

6.1 Analisi

Nello sviluppo di un plugin giocano un ruolo importante varie scelte, alcune ad alto livello di astrazione altre più tecniche. In primo luogo si deve capire dove collocare il plugin: nel layer System o nel layer Service. Questa scelta risulta fondamentale poichè ci sono grandi differenze nella funzione dei plugin in relazione al loro layer. In alcuni casi può essere necessario, o consigliato, creare due o più plugin appartenenti

ad entrambi i layer per poter ottenere le funzionalità richieste.

System: I plugin di questo layer offrono servizi agli altri agenti e non possono essere interrogati direttamente dall'utente.

- È buona norma imporre una comunicazione basata su documenti xml, validati attraverso un opportuno file xml schema e corredare il plugin con una classe message factory per facilitare la creazione dei messaggi;
- Ogni plugin svolge un solo compito, specializzare l'agente in modo da renderlo più semplice e massimizzarne la riusabilità;
- Se possibile evitare di assegnare uno stato globale all'agente, utilizzando gli stati legati alle sessioni;
- Creare nuovi trhead per gestire eventuali interfacce esterne;

Service: I plugin di questo layer sono progettati per essere interrogati direttamente dall'utente e utilizzano i servizi degli altri agenti.

- Comunicando prevalentemente con istanze della stessa classe è possibile utilizzare messaggi "proprietary", ad esempio oggetti java che vengono automaticamente serializzati e deserializzati;
- Ogni plugin rappresenta un'applicazione, non un insieme di funzioni sconnesse: un'interfaccia coerente e il più possibile usabile che permetta all'utente di utilizzare determinate funzionalità per realizzare una cooperazione.
- Se possibile evitare di assegnare uno stato globale all'agente, utilizzando gli stati legati alle sessioni;
- Usare sempre i servizi dell'infrastruttura, se un servizio non c'è lo si crea come plugin del layer System non si deve interfacciare direttamente un agente di questo layer con un servizio esterno (ad esempio interfacce grafiche non gestite dall'hostess, server di posta elettronica ecc);

Nelle sezioni successive verrà proposta una possibile lista di passi da seguire per lo sviluppo di un plugin.

6.2 Plugin di sistema

Per spiegare la progettazione di questo tipo di plugin, verrà utilizzato come esempio l'Helmsman (Timoniere, vedi Sezione 3.7) che presenta tutte le caratteristiche di un agente complesso. Seguendo la checklist riportata qui di seguito, verranno mostrate le scelte effettuate nella realizzazione del suddetto plugin.

1. Che servizio deve offrire il plugin?
2. Dobbiamo comunicare con entità esterne all'infrastruttura per svolgere tale servizio?
3. Abbiamo bisogno di usare servizi interni all'infrastruttura?
4. Quali sono le interrogazioni da supportare?
5. Abbiamo bisogno di uno stato globale? Se la risposta è affermativa, quale?
6. Che informazioni conserva lo stato di sessione?

Nel caso specifico del Timoniere queste sono state le risposte:

1. Interfacciamento con i navigatori (browser web), per permettere la realizzazione di interfacce utente basate sulla tecnologia web.
2. Sì; dobbiamo creare un server http che gestisca le connessioni con il navigatore. Per gestire il server http è stata estesa la classe `Thread` e al suo costruttore gli è stato passato il riferimento all'agente timoniere. In questo modo il server può comunicare al suo creatore le richieste che gli pervengono attraverso una funzione. Tale metodo, per mantenere la concorrenza, usa l'api `fireSelfEvent(...)` depositando un messaggio che in seguito l'agente riceverà.
3. No; non c'è la necessità di utilizzare altri servizi.
4. Le interrogazioni sono le seguenti:
 - Aggiunta di una risorsa web;

- Rimozione di una risorsa web;
- Invio di una pagina;
- Invio di una pagina di errore;

Le interrogazioni verranno validate attraverso un oggetto creato passando al costruttore della classe `JXMLSchema` lo schema xml contenuto nel file `helmsman.xsd`. Questo file è presente nella directory insieme ai file class del plugin e per essere identificato è stata usata una funzione che restituisce il path relativo ad un oggetto (`Constant.DIRECTORY.getDirFromClass(this)`).

5. Sì; per gestire la lista di risorse registrate esse verranno salvate in un `HashMap` contenuta nella classe `HelmsmanGlobalState`. Lo stato globale è presente nella classe `Helmsman` come campo privato e utilizza l'identificativo `global`.
 6. Lo stato di sessione conserva il riferimento all'agente che ha registrato una risorsa.
- * Le api a disposizione dello sviluppatore (Sezione 2.8) sono raggiungibili attraverso il puntatore `core` e gli interlocutori vengono identificati da un alias.

L'implementazione del plugin ora risulta molto semplice, in quanto si tratta di realizzare le seguenti classi e file:



plugins.system.helmsman (package)

Contiene i file base relativi all'agente:



Helmsman.java: implementa l'agente estendendo la classe `AbstractAgent<HelmsmanState>`;



HelmsmanGlobalState.java: lo stato globale;



HelmsmanState.java: lo stato di sessione;



HelmsmanMsgFactory.java: classe factory per i messaggi;



Helmsman.xsd: file Xml Schema per validare i messaggi;



manifest.plugin: file manifesto (Sezione 6.4);

**plugins.system.helmsman.http** (package)

Contiene i file relativi al server http:



IHttpListener.java: interfaccia che deve estendere l'agente per gestire il server http;



RequestHttp.java: richiesta dal navigatore;



ServerHttp.jav: server http (estende la classe **Therad**);



WebResource.java: risorsa web registrata;

**plugins.system.helmsman.skin** (package)

Contiene file aggiuntivi per il funzionamento del plugin:



error.html: pagina di errore;



list.html: pagina dell'elenco delle risorse registrate;



list_row.html: una riga della pagina delle risorse;



p4pLogo.gif: logo dell'infrastruttura;



p4pLogoSmall.gif: logo dell'infrastruttura (ridotta);

6.3 Plugin di servizio

Per spiegare la progettazione di questo tipo di plugin, verrà utilizzato come esempio il TicTacToe (Tris, vedi Sezione 4.2) che ne è un semplice esempio. Seguendo la checklist riportata qui di seguito, verranno mostrate le scelte effettuate nella realizzazione del suddetto plugin.

1. Quale cooperazione consente? E di che tipo?
2. Come interagisce l'utente?
3. Di quali servizi interni all'infrastruttura abbiamo bisogno?
4. Abbiamo bisogno di uno stato globale? Se la risposta è affermativa, quale?
5. Che informazioni conserva lo stato di sessione?
6. Con quali messaggi comunica?

Nel caso specifico del Tris queste sono state le risposte:

1. Permette all'utente di giocare a tris con un suo contatto. La cooperazione è di tipo uno a uno.
 2. L'utente utilizzerà un'interfaccia grafica per effettuare le mosse.
 3. Abbiamo bisogno di un'Hostess per gestire l'interfaccia grafica.
 4. No; non è necessario uno stato globale in quanto ogni partita è a se stante.
 5. Lo stato di sessione conserva lo stato della partita.
 6. Per comunicare una mossa, fra l'hostess e l'agente o fra i due giocatori viene utilizzato un intero opportunamente codificato, mentre per l'aggiornamento dell'interfaccia si utilizza lo stato di sessione.
- * Le api a disposizione dello sviluppatore (Sezione 3.8.2) sono raggiungibili attraverso il puntatore `body`. È da notare che ora gli interlocutori vengono identificati utilizzando un nome (stringa di testo) non più con un alias.

L'implementazione del plugin ora risulta molto semplice, in quanto si tratta di realizzare le seguenti classi e file:



plugins.service.tictactoe (package)

Contiene i file base relativi all'agente:



TicTacToe.java: implementa l'agente estendendo la classe `AbstractWorker<TicTacToeState>`;



TicTacToeGui.java: interfaccia grafica, estende la classe `AgentGui`;



TicTacToeState.java: lo stato di sessione;



manifest.plugin: file manifesto (Sezione 6.4);

6.4 Manifest.plugin

Questo file contiene le informazioni necessarie per gestire un plugin. Nell'esempio seguente vediamo il manifesto relativo all'agente Messenger per la messaggeria istantanea.

manifest.plugin
<pre><?xml version='1.0' encoding='UTF-8'?> <plugin> <name>Messenger</name> <version>1</version> <type>messenger</type> <className>plugins.service.messenger.Messenger </className> <groupMenu>chat room</groupMenu> <peerMenu>chiacchera</peerMenu> </plugin></pre>

Le informazioni più importanti contenute in questo file sono:

- type: serve per identificare il servizio svolto (possono esistere più di un plugin che offrono lo stesso servizio ma ce ne può essere uno solo installato);
- className: la classe principale del plug in, quella che estende la classe astratta **AbstractAgent** o **AbstractWorker**;
- groupMenu: la voce del menù contestuale relativo ai gruppi;
- peerMenu: la voce del menù contestuale relativo ai contatti;

Ogni plugin deve avere questo file anche se si tratta di un plugin dello strato System e che quindi non avrà voci di menù.

6.5 Interfacce grafiche

Sviluppando plugin per lo strato service spesso si ha la necessità di creare interfacce grafiche; la creazione di finestre deve essere demandata al servizio Hostess, la creazione di JFrame è caldamente sconsigliata

per questioni concettuali e pratiche. Per creare una nuova interfaccia è sufficiente estendere la classe `AgentGui` che trovate nel package `p4p.system.hostess.gui`; si tratta di un `JPanel` che rappresenta il contenuto della finestra. Tale estensione comporta l'implementazione dei seguenti metodi astratti:

- **protected void init():** Questa funzione viene chiamata durante la creazione della finestra; qui va inserito il codice relativo ai componenti della finestra (pulsanti, campi di testo ed etichette);
- **public PreferenceGui getPreference():** Deve ritornare le caratteristiche della finestra che si vuole creare, ad esempio il titolo e le dimensioni, per completare questa funzione basta inserire la seguente riga di codice opportunamente completata `return new PreferenceGui(...);`
- **public void event(Object dati):** Questa funzione viene richiamata quando l'interfaccia riceve un messaggio dall'agente che l'ha creata. Viene usata in genere per aggiornare il contenuto dell'interfaccia;

È consigliabile implementare anche le varie interfacce di ascolto degli eventi swing relativi ai componenti della finestra, in questo modo si può utilizzare il metodo `fireEvent(Object event)` per inviare eventi all'agente.

6.6 Prospettive future

Molti tipi di cooperazione possono essere favoriti e supportati attraverso questa infrastruttura. Il campo di applicazione è molto vasto e va dalla comunicazione interpersonale alla didattica, dalla condivisione di contenuti all'intrattenimento. Lasciando alla comunità l'onere di realizzare i relativi plugin riporto alcune possibili applicazioni:

System:

- Trasferimento File
- Xml: data base distribuiti

- Voice over Ip
- Audio e Video Streaming
- Riconoscimento vocale
- Sintetizzazione vocale
- Gestione Feed RSS

Service:

- Forum
- Applicazioni di comunicazione avanzata (voip, video conferenza)
- Gestione blog
- Motori di ricerca di contenuti
- File sharing
- Sistemi didattici
- Applicazioni ludiche

Appendice A

Pattern decorator-delegation

Si tratta dell'unione di un pattern decorator e di un pattern delegation. La classe decorarata delega alla classe decoratrice alcune sue funzionalità ma nasconde attraverso un'interfaccia i suoi tecnicismi offrendo esclusivamente le api prefissate. Mentre la classe decoratrice utilizza metodi protetti ed astratti come delegati. Essi vengono in fine implementati dalla classe che estenderà la classe decoratrice ma essi essendo protetti non saranno accessibili dall'esterno, ottenendo in questo modo un completo mascheramento del codice.

Viene di seguito riportato un esempio di questo pattern:

ICorpo.java
<pre>public interface ICorpo{ public void corri(); }</pre>

Corpo.java

```
public class Corpo implements ICorpo {
    private Mente mente;
    public void Corpo(Mente m){
        mente = m;
    }
    public void corri(){
        // delegation
        mente.stanco();
    }
}
```

Mente.java

```
public abstract class Mente {
    private ICorpo corpo;
    public Mente(){
        // decorator
        corpo = new Corpo(this);
    }
    protected abstract void stanco();
}
```

Uomo.java

```
public class Uomo extends Mente {
    protected void stanco() {
        System.out.println("Sono esausto!");
    }
    public void muoviti(){
        corpo.corri();
    }
    // funzione main
    public static void main(String[] args) {
        Uomo u = new Uomo();
        u.muoviti();
        // viene visualizzato: Sono esausto!
    }
}
```

Appendice B

Pattern decorator-polimorfico

Per aggirare la limitazione dell'ereditarietà singola propria del linguaggio Java, in genere, si utilizza il pattern decorator ma esso non supporta efficacemente il polimorfismo. Questo mi ha spinto ad ideare un pattern capace di decorare mantenendo questa proprietà valida. È comunque da ricordare che questo pattern comporta sequenze di reindirizzamento dei metodi che possono influire negativamente sulle prestazioni; quindi consiglio di usare questo sistema con parsimonia. In primo luogo le classi da estendere devono essere realizzate seguendo alcune direttive:

1. Devono implementare un'interfaccia con tutti i metodi che si intende rendere polimorfici.

```
public class Classe implements IClasse
```

2. Devono avere un campo privato del tipo dell'interfaccia e con l'identificativo uguale al nome dell'interfaccia ma con la I iniziale sostituita da un underscore (_).

```
private IClasse _Classe;
```

3. Si può decidere di assegnare tale campo nel costruttore oppure aggiungere un metodo apposito utilizzabile anche a runtime.

```
public void estesoDa(IClasse o){  
    _Classe = o;  
}
```

4. Ogni metodo dell'interfaccia deve essere duplicato con il nome preceduto da un underscore.

```
public void m(){...}  
public void _m(){...}
```

5. I metodi con l'underscore devono contenere il codice della funzione.

6. I metodi senza underscore devono contenere il seguente codice:

```
public void m(){  
    if(_Classe != null)  
        _Classe.m();  
    else  
        _m();  
}
```

Le classi che estendono, invece, devono essere realizzate seguendo altre direttive:

1. Devono implementare le interfacce delle classi che estendono.

```
public class NuovaClasse implements IClasse
```

2. Devono avere un campo privato del tipo della classe estesa e con l'identificativo uguale al nome dell'interfaccia ma con la I iniziale sostituita da un underscore (_).

```
private Classe _Classe;
```

3. Nel costruttore devono istanziare le classi da estendere, le devono associare al campo privato ed eventualmente utilizzare la funzione di estensione.

```
public NuovaClasse(){  
    _Classe = new Classe();  
    _Classe.estesoDa(this);  
}
```

4. I metodi dell'interfaccia devono contenere il seguente codice.

```
public void m(){  
    _Classe._m();  
}
```

Se si intende ridefinire il comportamento di un metodo è sufficiente modificare tale codice.

Per meglio comprendere il procedimento prendiamo ad esempio, una classe Gatto che ne estende altre due, Quadrupede e Mammifero. Le interfacce relative a quest'ultime hanno un metodo comune ere-

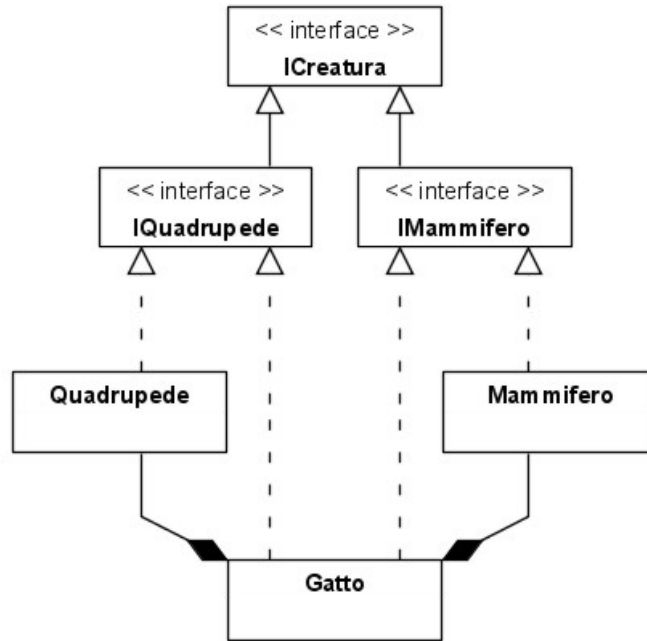


Figura B.1: Pattern decorator polimorfico: Schema UML

ditato dall'interfaccia ICreatura. È buona norma creare i nomi delle interfacce antepo-
nendo una I maiuscola ai nomi delle classi.

ICreatura.java

```
public interface ICreatura{
    public String getNome();
}
```

IQuadrupede.java

```
public interface IQuadrupede extends ICreatura{
    public void corri();
}
```

IMammifero.java

```
public interface IMammifero extends ICreatura{
    public String getVerso();
    public void parla();
}
```

Quadrupede.java

```
public class Quadrupede implements IQuadrupede{
    private IQuadrupede _Quadrupede;
    public Quadrupede(){
        _Quadrupede = null;
    }
    public void estesoDa(IQuadrupede o){
        _Quadrupede = o;
    }
    public String getNome(){
        if(_Quadrupede != null)
            return _Quadrupede.getNome();
        else
            return _getNome();
    }
    public void corri(){
        if(_Quadrupede != null)
            _Quadrupede.corri();
        else
            _corri();
    }
    public String _getNome(){
        return "Quadrupede";
    }
    public void _corri(){
        System.out.println(getNome() + "] Corre");
    }
}
```

Mammifero.java

```
public class Mammifero implements IMammifero{
    private IMammifero _Mammifero;
    public Mammifero(){
        _Mammifero = null;
    }
    public void estesoDa(IMammifero o){
        _Mammifero = o;
    }
    public String getNome(){
        if(_Mammifero != null)
            return _Mammifero.getNome();
        else
            return _getNome();
    }
    public String getVerso(){
        if(_Mammifero != null)
            return _Mammifero.getVerso();
        else
            return _getVerso();
    }
    public void parla(){
        if(_Mammifero != null)
            _Mammifero.parla();
        else
            _parla();
    }
    public String _getNome(){
        return "Mammifero";
    }
    public String _getVerso(){
        return "?????";
    }
    public void _parla(){
        System.out.println(getNome() + "] Dice: "
            + getVerso());
    }
}
```

Gatto.java

```
public class Gatto implements IQuadrupede, IMammifero{
    private Quadrupede _Quadrupede;
    private Mammifero _Mammifero;
    public Gatto(){
        _Quadrupede = new Quadrupede();
        _Quadrupede.estesoDa(this);
        _Mammifero = new Mammifero();
        _Mammifero.estesoDa(this);
    }
    public String getNome(){
        return "Gatto";
    }
    public void corri(){
        _Quadrupede._corri();
    }
    public String getVerso(){
        return "Miao";
    }
    public void parla(){
        _Mammifero._parla();
    }
}
```

Ringraziamenti

Ringrazio i miei familiari e i vecchi amici che mi hanno sopportato e sostenuto; sono certo che lo faranno anche in futuro.

Ringrazio gli amici e colleghi che mi hanno accompagnato in questo lungo viaggio, attraverso le belle giornate ventose e quelle interminabili di bonaccia, condividendo con mè le difficoltà e le soddisfazioni fra un Tresette ed una partita a Need for speed, fra una discussione politica ed una sportiva.

Infine ringrazio i professori, quelli che mi hanno fatto amare le loro materie e quelli che me le hanno fatte odiare; nessuno è perfetto, ma ognuno dà il suo contributo.

Grazie a tutti voi!

Bibliografia

- [1] Java: <http://java.sun.com/>
- [2] JavaMail: <http://java.sun.com/products/javamail/>
- [3] TuCSoN: <http://lia.deis.unibo.it/research/tucson/>
- [4] ReSpecT: *Fare riferimento alla documentazione di TuCSoN*
- [5] Prolog: <http://it.wikipedia.org/wiki/Prolog/>
- [6] XStream: <http://xstream.codehaus.org/>
- [7] JDOM: <http://www.jdom.org/>
- [8] Xerces: <http://xerces.apache.org/xerces2-j/>
- [9] HTTP: <http://it.wikipedia.org/wiki/HTTP/>

Elenco delle figure

1.1	Il modello di riferimento ISO OSI	2
1.2	Gli strati dell'infrastruttura p4p	2
1.3	Una regola base ReSpecT	4
1.4	Una regola condizionale ReSpecT	5
1.5	Un esempio di due regole ReSpecT	6
2.1	Rappresentazione logica di un nodo p4p	8
2.2	Schema UML delle classi principali presenti nel Core .	9
2.3	Invio e ricezione di un invito	13
2.4	Conferma di ricezione e generazione di un'eccezione . .	14
2.5	Reinstradamento della risposta dell'agente di look-up .	14
2.6	Comando di creazione e distruzione del descrittore di una risorsa	23
2.7	Comando di creazione e distruzione del descrittore di un contenitore di risorse	24
2.8	Reinstradamento richiesta e risposta di look-up	24
2.9	Reinstradamento eccezione e conversione di un invito remoto	25
2.10	Regole di supporto per convertire l'evento esterno out in evento interno out_r	25
2.11	Gestione richiesta di risorsa locale	26
2.12	Gestione richiesta di risorsa remota	26
2.13	Risveglio messaggi in attesa	27
2.14	Generazione eccezioni da timeout	27
2.15	Comando di creazione e distruzione di una sessione . .	37
2.16	Comando di creazione e distruzione di un alias	37
2.17	Gestione della ricezione di un invito	38

2.18	Reinstradamento messaggi dell'agente	39
2.19	Regola di supporto e tentativo di invio di un messaggio	40
2.20	Messaggio di conferma e generazione eccezioni	40
2.21	Comando remoto di distruzione di un alias e rimozione alias incompleto	41
2.22	Notifica stato dell'agente e dei descrittori di risorse . .	41
2.23	Richiesta di LookUp, reinstradamento, risposta al Loo- kUp	42
2.24	Reinstradamento di eccezioni e gestione massimo per la tupla next	42
2.25	Gestione validità della tupla next	43
2.26	Risveglio tuple in attesa	43
2.27	Eliminazione tuple in attesa	44
2.28	Creazione di un invito	44
2.29	Completamento connessione su invito	45
2.30	Funzionamento normale	50
2.31	Funzionamento in caso di eccezione	50
2.32	Trasferimento di un messaggio fra agenti remoti	51
2.33	Trasferimento di un messaggio fra agenti remoti con eccezioni	51
2.34	Richiesta	54
2.35	Localizzazione remota	55
2.36	Completamento	55
3.1	Architettura dello strato System	59
3.2	Schema UML dell'agente Operator	60
3.3	Risposta immediata	61
3.4	Risposta differita	61
3.5	Evento esterno	61
3.6	Invio di un messaggio di posta	62
3.7	Ricezione dei messaggi di posta	63
3.8	Rimozione di messaggi di posta	64
3.9	Richiesta di creazione di un'interfaccia	65
3.10	Richiesta di distruzione di un'interfaccia	65
3.11	Richiesta di visualizzazione di un'interfaccia	66
3.12	Richiesta di occultamento di un'interfaccia	66
3.13	Richiesta di ridimensionamento di un'interfaccia	66

3.14	Richiesta di aggiornamento di un'interfaccia	67
3.15	Evento dell'interfaccia grafica	67
3.16	Richiesta di Pivot Discovery	68
3.17	Richiesta di Service Discovery	69
3.18	Richiesta di Ip Discovery	70
3.19	Start-Up	72
3.20	Richiesta delle informazioni dell'utente	73
3.21	Richiesta della lista degli utenti di un gruppo	74
3.22	Richiesta di selezione di un contatto	75
3.23	Richiesta di conferma	75
3.24	Registrazione di una nuova risorsa web	77
3.25	Rimozione di una nuova risorsa web	78
3.26	Invio di una pagina web	78
3.27	invio di una pagina web di errore	79
3.28	Ricezione di una richiesta del navigatore	79
3.29	Rete a topologia mista	80
3.30	Notifica di Start Up	81
3.31	Richiesta di invito	81
3.32	Richiesta di servizio locale	82
4.1	Menù contestuale relativo ai contatti	88
4.2	Messaggeria istantanea	88
4.3	Menù contestuale relativo ai gruppi	89
4.4	Gestione Chat Room (creazione)	89
4.5	Gestione Chat Room (accesso)	89
4.6	Una comunicazione di gruppo	90
4.7	La scacchiera del Tris	91
4.8	Una partita a Tris	91
5.1	Log-In	94
5.2	Interfaccia principale	95
5.3	Gestione Utente	95
5.4	Finestra di conferma	96
5.5	Finestra di selezione di un contatto	96
5.6	Cartelle plugin	97
B.1	Pattern decorator polimorfico: Schema UML	113