UNIVERSITA' DEGLI STUDI DI PADOVA

Facoltà di Ingegneria

Corso di Laurea Triennale in Ingegneria Informatica

TESI DI LAUREA

Pari-IM: aggiornamento client MSN alla versione 15

RELATORE: Chiar.mo Prof. Enoch Peserico Stecchini Negri De Salvi

CORRELATORE: Ing. Paolo Bertasi

LAUREANDO: Gianluigi Disconzi

A.A. 2010-2011

Prefazione

Il presente elaborato costituisce la relazione del lavoro da me personalmente svolto, tra Settembre 2010 e Marzo 2011, sulla modifica del client di Instant Messaging di PariPari al fine di implementare una versione aggiornata del protocollo MSN.

Inizialmente si introducono le principali caratteristiche del progetto PariPari. Si passa quindi a presentare concetti, convenzioni e specifiche che saranno utili nell'affrontare le parti centrali della trattazione: si forniscono pertanto i rudimenti dell'Instant Messaging e della presenza online, si studia la composizione del protocollo MSN e come funziona l'infrastruttura ad esso sottesa, e viene infine introdotto il plugin di IM che nel progetto PariPari ha tra gli altri il compito di interfacciarsi correttamente alla rete Messenger.

La seconda parte descrive il lavoro svolto per l'adeguamento del client alla versione di protocollo a cui ci si è prefissi di aderire: in particolare vengono trattati gli interventi eseguiti per l'aggiornamento della procedura di login nella rete MSN, ivi compresa l'autenticazione e la gestione delle credenziali di accesso, sino a giungere alla gestione delle liste di contatti e delle sessioni di chat vere e proprie. Oltre che alla procedura di ingresso, questa parte spiega come si è intervenuti nella gestione dei parametri di riconoscimento durante la connessione.

Viene quindi descritto il lavoro eseguito sul file transfer: si approfondisce innanzitutto la conoscenza del protocollo MSN introducendone i concetti specifici per la trasmissione di dati per passare poi a presentare gli interventi effettuati sul client.

La relazione si conclude con una serie di riflessioni sullo stato attuale del plugin e si presentano le possibilità di sviluppo future.

Fig. 1.1: Logo di PariPari

Parte I. Introduzione

1 Il progetto PariPari

PariPari è una rete Peer-To-Peer serverless multifunzionale; il suo scopo principale è di fornire numerosi servizi, che oggi si possono avere solo utilizzando diverse applicazioni separate, in un unico programma - [Bertasi(2005)]. Il linguaggio di programmazione usato per PariPari è Java e, nonostante presenti degli svantaggi (principalmente scarse prestazioni) molti sono i vantaggi tra i quali:

- Maggior portabilità a causa dell'indipendenza dalla piattaforma;
- Maggior sicurezza dell'applicazione;
- Uso facilitato per l'utente finale grazie a Java Web Start che permette di scaricare ed eseguire le applicazioni direttamente dal web nonché di avere sempre l'ultima versione del software disponibile, evitando procedure di installazione;
- Sviluppo facilitato di nuovi servizi per il programmatore grazie alle numerose API disponibili;

L'adozione dell'Extreme Programming, una metodologia agile che prevede la scrittura a più mani del codice e il suo continuo testing per renderlo robusto e privo di errori, consente di coordinare gli interventi di più persone all'interno dei gruppi di lavoro.

1.1 La struttura della rete

PariPari non prevede l'utilizzo di nodi gerarchizzati (client o server) bensì un numero di nodi equivalenti (peer), che fungono sia da cliente sia da servente verso gli altri nodi della rete. L'assenza di server centralizzati nella struttura peer-to-peer comporta numerosi vantaggi:

- Indipendenza: si annulla il rischio di collasso nel caso in cui uno o più server non fossero disponibili;
- Affidabilità: si azzera il rischio di attacchi di tipo Denial of Service[1];
- Scalabilità: poiché ogni nodo contribuisce con le proprie risorse alla sopravvivenza della rete, non è necessario che la potenza dei server venga aumentata per scongiurare il collasso della rete stessa.

PariPari implementa una variante dell'algoritmo Kademlia, utilizzato anche da altri programmi peer-to-peer come, ad esempio, Emule.

1.2 I plug-in e i servizi

PariPari (inteso come programma applicativo) presenta una struttura modulare molto semplice ed efficace organizzata in plug-in. Un plug-in è un programma non autonomo che interagisce con un altro programma per ampliarne le funzioni. Grazie a questa strategia, il programma principale può essere ampliato senza necessità di essere modificato; inoltre possono essere aggiunte infinite funzioni mantenendo la medesima architettura principale. I plug-in in PariPari si dividono in due diverse tipologie:

- Cerchia interna: Core, Crediti, Connettività, Storage e DHT forniscono servizi di base per il funzionamento della rete e si coordinano con il Core per gestire richieste di risorse quali rete, disco, etc.
- Cerchia esterna: Emule, Torrent, Voip, IM, IRC, DNS, etc. Si basano sull'utilizzo dei plug-in della cerchia interna per fornire i servizi specifici.

Il core ha il compito di gestire le richieste ed assegnare equamente le risorse disponibili per evitare starvation[2] a favore di un singolo plug-in.

2 Il protocollo MSN Messenger

La dicitura "MSN Messenger" viene utilizzata per riferirsi generalmente all'infrastruttura di rete in cui si organizza il servizio di messaggistica istantanea sviluppato da Microsoft. Il protocollo che ne determina il funzionamento e ne codifica il comportamento è generalmente indicato con la sigla MSNP (**M**icro**s**oft **N**otification **P**rotocol) a cui segue la versione dello stesso quando vi sia necessità di indicarlo.

Il protocollo MSN appartiene alla famiglia dei protocolli di "presenza e messaggistica istantanea", i quali descrivono sistemi per lo scambio in tempo reale di informazioni sul proprio stato di presenza e di messaggi tra computer connessi alla rete. Una lettura interessante che ne descrive approfonditamente i concetti è l'RFC 2778: [Day et al.(2000)Day, Rosenberg, and Sugano].

Sia per il protocollo di comunicazione che per la struttura della rete e dei servizi Microsoft, non esistono documenti ufficiali sui quali basare il lavoro di sviluppo di un client (fatto salvo

[1] Denial of Service: impedire l'accesso autorizzato a risorse, oppure ritardare l'esecuzione di operazione time-critical.

[2] Starvation o stallo individuale : fenomeno per il quale un processo (o un insieme di processi) monopolizza gran parte delle risorse disponibili nel sistema

un internet draft di stampo informativo - [Movva and Lai(1999)]); è anzi vero che molte delle loro caratteristiche, come si vedrà, sono state modellate con l'intento di scoraggiare se non di impedire totalmente l'accesso al servizio da parte di client non ufficiali.

La mancanza di disponibilità di documentazione riguardante MSN Messenger ha portato alla creazione di diverse comunità virtuali dedite allo studio del sistema IM di Microsoft. La loro attività ha reso possibile conoscere in modo sufficientemente approfondito il servizio, al punto che al momento in cui si scrive si stima esistano almeno 20 client in grado di supportarlo, secondo [Wikipedia(2011a)] (conteggiando solamente quelli a grande diffusione e con codice stabile), mentre fonti del 2003 ne citano addirittura 41, vedi [Hypotetic.org(2003b)].

Il presente lavoro trae spunto da informazioni reperite presso summenzionate comunità (in particolar modo dal forum di [MSNFanatic(2011a)]); tuttavia la parte preponderante e sicuramente più interessante è stata infatti quella di validare tali informazioni (talvolta non del tutto complete o affidabili) mediante o verifica diretta contro altri client già funzionanti o mediante utilizzo di strumenti atti allo scopo, su tutti Wireshark.

Convenzioni Nel corso dell'esposizione si presenteranno diversi scambi di comandi tra client e server. Verranno utilizzate le seguenti due notazioni grafiche per indicare la direzione in cui viaggiano tali messaggi:

```
>>> Messaggio inviato dal client al server
<<< Messaggio inviato dal server al client
```

L'evidenziazione della sintassi di un comando in particolare avverrà inserendo tra parentesi uncinate le parti variabili.

2.1 Cenni sull'infrastruttura

L'infrastruttura su cui si basa il servizio MSN Messenger è composta da server organizzati funzionalmente secondo 3 gruppi:

- Dispatch Server: agli indirizzi http://messenger.hotmail.com:1863/ e (nel caso di connessione mediante tunnel HTTP) http://gateway.messenger.hotmail.com:80, fanno capo queste macchine a cui i client puntano al momento di instaurare la connessione al servizio; la loro funzione è di fornire al client l'indirizzo di un Notification Server sufficientemente libero da poter accettare una connessione di lunga durata;

- Notification Server (NS): è il server di presenza online, ovvero la macchina con cui si deve mantenere una connessione per tutto il tempo in cui si desidera usufruire del servizio MSN (la disconnessione dal NS che viene assegnato al client comporta la necessità di dover effettuare una nuova procedura di accesso per tornare online); i NS coordinano le presenze dei vari utenti connessi e permettono di iniziare sessioni di chat che verranno poi effettivamente svolte su delle macchine destinate all'uopo;

- Switchboard Server: sono le macchine che veicolano le comunicazioni tra gli utenti, facendo in un certo senso da proxy tra i partecipanti. Per un client ogni sessione di IM corrisponde ad una connessione con altrettanti SB, e per ognuna di esse il numero di partecipanti può essere maggiore di due: questo schema di funzionamento assume le caratteristiche di una vera e propria sessione di chat[3].

[3] Sarebbe eccessivamente pedante forzare la definizione di chat al suo significato originario in cui era prevista la possibilità di scambiare messaggi anche con utenti anonimi (come ad esempio poteva accadere nella rete IRC); tuttavia esiste pure qui la possibilità di ritrovarsi a chiacchierare con gente sconosciuta in un caso limite che si vedrà nella sezione dedicata alla chat.

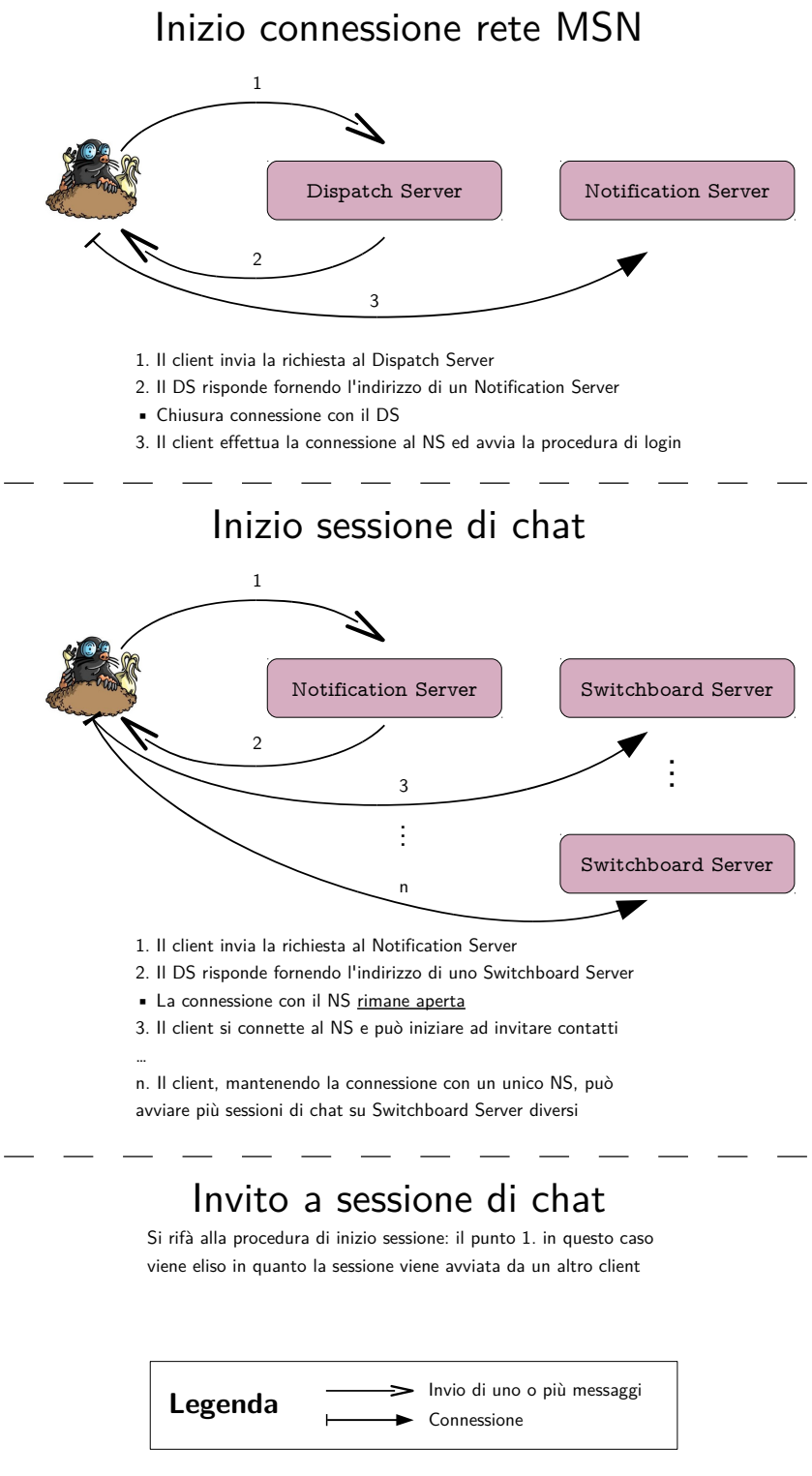

Fig. 2.1: Schema di funzionamento della rete MSN

Esiste infine una quarta famiglia di macchine a cui non è stato dato un nome formalmente, in quanto sono state introdotte nella versione 13 del protocollo MSNP (la nomenclatura utilizzata fa sempre riferimento al draft [Movva and Lai(1999)]): queste hanno la funzione di memorizzare e fornire a richiesta le liste di utenti associate a ciascun contatto, tipicamente mediante richieste SOAP, vedi [W3C(2007)].

2.2 MSNP

Il cuore del protocollo MSN è dato dall'insieme di regole di grammatica e di sintassi da applicare ai comandi scambiati tra client e server. Tutto ciò che viene trasmesso tra essi è un comando, il quale è rappresentato da una combinazione di 3 lettere maiuscole seguite da dei parametri (specifici per ciaschedun comando) separati da uno spazio; il tutto è terminato dal carattere speciale "`\r\n`".

Lo schema tipico prevede che sia il client ad inviare per primo un comando al server, prestando attenzione a numerarlo con un identificativo detto **TrID** (Transaction identifier)[4]; a questo il server risponde con un altro comando, citando il TrID fornito dal client.

Oltre ai comandi che prevedono questa semplice sintassi ve ne sono di altri tipi:

- con payload: prevedono la presenza di ulteriori dati dopo i caratteri di terminazione; la gestione del payload verrà discussa in seguito;
- di errore: inviati dal server al client, hanno la peculiarità di essere indicati da un codice numerico a 3 cifre in sostituzione delle 3 lettere maiuscole;
- comandi asincroni: sono inviati dal server senza che vi sia una richiesta esplicita da parte del client; talvolta alcuni di questi comandi utilizzano un TrID pari a zero.

Alcuni tra i comandi di maggiore importanza all'interno del protocollo verranno presentati alla bisogna durante questa presentazione.

2.2.1 La versione 15 del protocollo

La versione a cui ci si è prefissi di aderire durante il lavoro svolto sul plugin IM PariPari è la numero 15, rilasciata ad Ottobre 2006 con il client "MSN Live 8.1", e tutt'ora supportata dalla rete Messenger.

L'importanza di giungere ad un pieno supporto a detta versione risiede principalmente nel numero e nella tipologia di funzionalità implementate, e non secondariamente nella costante opera di epurazione delle versioni di protocollo obsolete costantemente compiuta da Microsoft (soprattutto in caso di riscontro di vulnerabilità, vedi [Microsoft.com(2007)]) mediante blocco dell'accesso a client di versioni eccessivamente vecchie.

Le novità più interessanti introdotte in MSNP15 includono:

- nuovo metodo di autenticazione basato su Single Sign On (SSO): una volta verificato l'account al momento dell'accesso, viene rilasciato un ticket valido per l'intera sessione con cui effettuare le operazioni che richiedono una verifica dell'identità; questo meccanismo di fatto integra la rete MSN all'interno del sistema di servizi Microsoft (Hotmail, Live Spaces, etc...);
- miglioramento delle procedure SOAP per la gestione delle liste dei contatti, soprattuto grazie all'impiego dei ticket di sessione;

[4] Il TrID deve essere univoco ed ogni successione di TrID è associata ad una connessione, pertanto è lecito che un client invii un comando con TrID 10 ad un server ne invii un altro ad un secondo server con lo stesso identificativo; se in una stessa connessione un TrID viene però ripetuto avviene la disconnessione immediata senza preavviso.

- proprietà dell'utente gestite in modalità roaming: le informazioni personali sono conservate su server e vengono inviate al client al momento del login, rendendo indipendenti tali informazioni dal particolare client utilizzato e, per estensione, dal luogo da cui ci si connette alla rete.

3 Pari-IM: introduzione all'architettura del plugin

Il plugin IM è stato progettato con l'intenzione di essere reso il più generico ed estensibile possibile; il suo scopo non è di implementare un protocollo di IM in particolare, bensì di fornire un'infrastruttura di base su cui costruire plugin specifici per i vari protocolli.

Come si vede il nucleo del plugin IM astrae dalla peculiare struttura di qualsivoglia protocollo di messaggistica istantanea. Il suo obbiettivo è quello di fornire ai progettisti di plugin specifici ed, in ultima istanza, agli utenti stessi un mezzo per comunicare automaticamente con tutti i protocolli desiderati, come descritto in [Vallini(2009)].

La costruzione di un plugin che vada ad innestarsi su quello già presente deve pertanto essere progettato pensando a due parti:

- il listener: la parte di plugin che si fa carico di inviare e ricevere comandi da/al server, i quali sono chiaramente dipendenti dal tipo di protocollo che si intende usare;

- il core vero e proprio: essendo la sua architettura relativamente indipendente dal plugin IM lo si può strutturare con un certo grado di libertà.

Il plugin MSN, già giunto ad un buon punto di maturazione dal punto di vista architetturale, vedi anche [Montini(2009)], può essere schematizzato secondo la figura 3.2.

Dall'immagine si possono collocare concettualmente tutti i vari componenti, ciascheduno implementato come classe. Con questo schema in mente è ora possibile presentare nel dettaglio gli interventi apportati ai vari componenti: nel prosieguo della trattazione si presenteranno di volta in volta gli aspetti peculiari che contraddistinguono il protocollo MSNP15 dalle versioni precedenti e verrà spiegato come tali caratteristiche sono state implementate nel plugin.

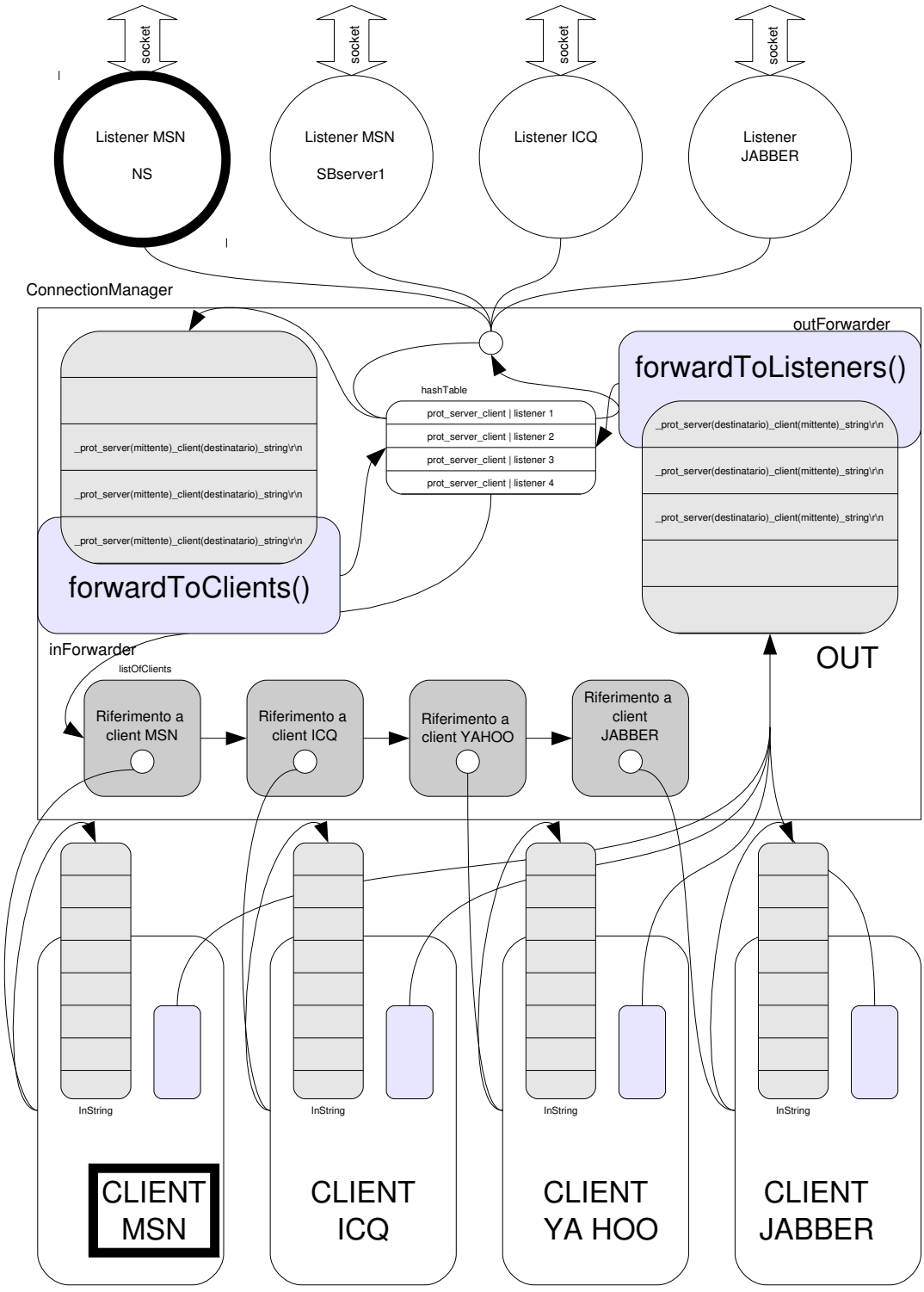

Fig. 3.1: Struttura del plugin IM

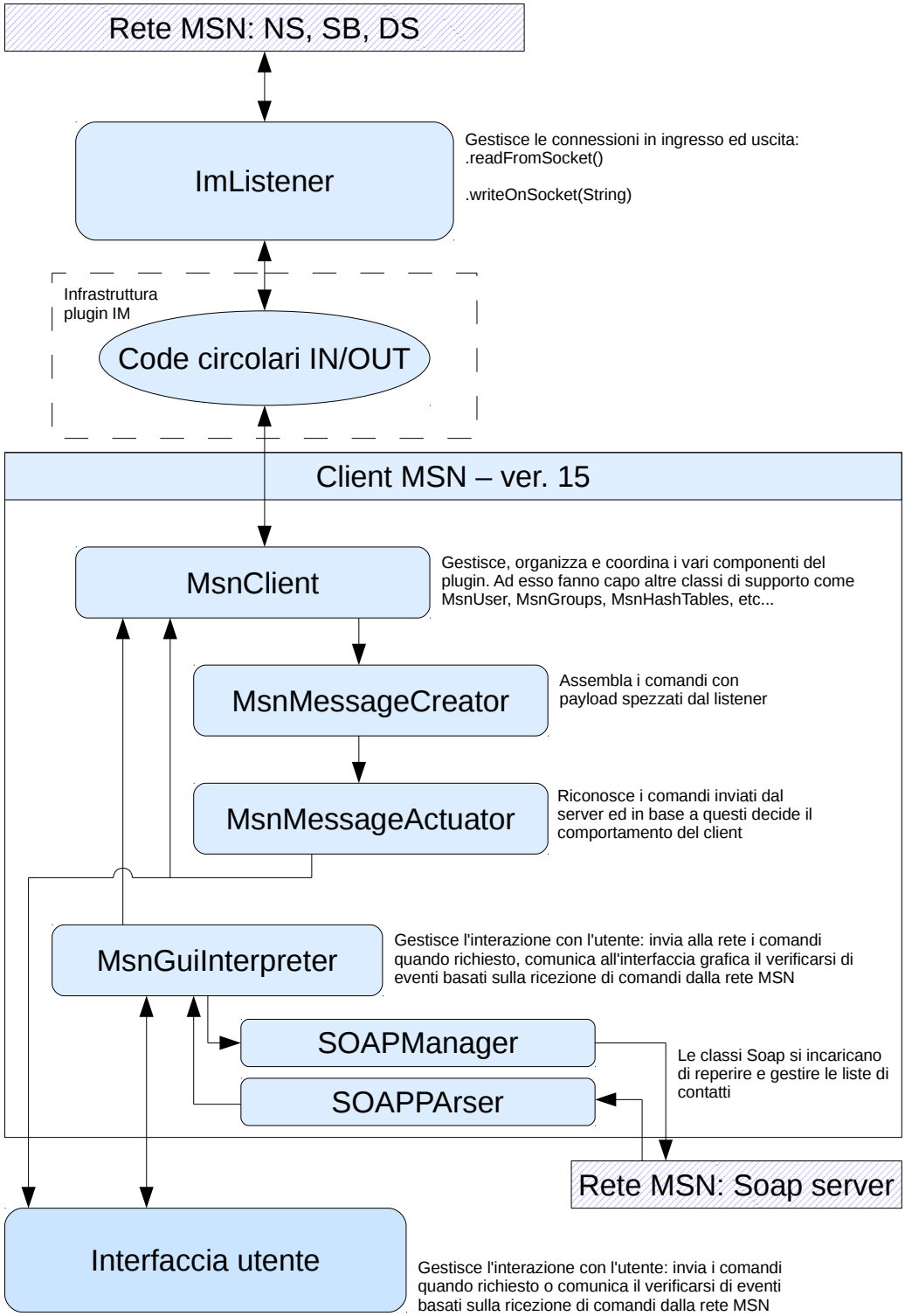

Fig. 3.2: Schema plugin MSN

Parte II. Interventi sul client

In questa parte si presenterà il lavoro svolto per aggiornare il client. In questa sezione nulla è stato fatto per alterare lo schema di funzionamento del client: i meccanismi di interrelazione tra i suoi componenti e le logiche di funzionamento sono rimaste immutate. Semplicemente si sono apportate modifiche mirate al codice deputato a svolgere talune operazioni, rispettando le modifiche apportate al protocollo dagli architetti Microsoft.

4 Procedura di autenticazione e login

La prima parte del lavoro svolta è stata finalizzata alla realizzazione di una procedura di login che rispettasse quanto previsto dal protocollo MSNP15. A differenza di quanto previsto nelle versioni precedenti la procedura di riconoscimento delle credenziali e di conferimento di certificato per l'accesso alla rete MSN prevede lo scambio di informazioni con più entità e con varie metodologie.

Questa scelta è stata dettata dall'impossibilità di compiere l'accesso alla rete Messenger con un qualsiasi algoritmo di autenticazione antecedente a quello previsto nella versione 15, algoritmo non implementato nel client PariPari al momento dell'inizio del lavoro di chi scrive.

Le classi interessate da questa prima azione di aggiornamento sono state MsnMessageActuator per quanto riguarda la risposta ai comandi ricevuti dal NS, SOAPManager e SOAPPArser per la gestione della richiesta dei token di sicurezza.

4.1 Ottenimento policy e nonce

Dopo aver negoziato la versione di protocollo e di client come da prassi generale, segue la comunicazione da parte del client del nome utente, a cui il server risponde fornendo dei security tokens, secondo il seguente abstract:

```
>>> USR <TrID> SSO I <email>\r\n
<<< USR <TrID> SSO S <policy> <base64 encoded nonce>\r\n
```

4.2 Ottenimento ticket token e binary secret

Avendo ora il token di policy ci si collega al sito `https://login.live.com/RST.srf` e si effettua un comando POST come segue:

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
   xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
   xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.
       xsd"
   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
   xmlns:wssc="http://schemas.xmlsoap.org/ws/2004/04/sc"
   xmlns:wst="http://schemas.xmlsoap.org/ws/2004/04/trust">
   <Header>
      <ps:AuthInfo   xmlns:ps="http://schemas.microsoft.com/Passport/SoapServices/PPCRL" Id="
          PPAuthInfo">
         <ps:HostingApp>{7108E71A-9926-4FCB-BCC9-9A9D3F32E423}</ps:HostingApp>
         <ps:BinaryVersion>4</ps:BinaryVersion>
         <ps:UIVersion>1</ps:UIVersion>
         <ps:Cookies></ps:Cookies>
         <ps:RequestParams>AQAAAAIAAABsYwQAAAAxMDMz</ps:RequestParams>
      </ps:AuthInfo>
      <wsse:Security>
         <wsse:UsernameToken Id="user">
            <wsse:Username>UserEmail</wsse:Username>
            <wsse:Password>UserPassword</wsse:Password>
         </wsse:UsernameToken>
      </wsse:Security>
   </Header>
```

```
24    <Body>
25          <ps:RequestMultipleSecurityTokens
26              xmlns:ps="http://schemas.microsoft.com/Passport/SoapServices/PPCRL" Id="RSTS">
27              <wst:RequestSecurityToken Id="RST0">
28                  <wst:RequestType> http://schemas.xmlsoap.org/ws/2004/04/security/trust/Issue
29                  </wst:RequestType>
30                  <wsp:AppliesTo>
31                      <wsa:EndpointReference>
32                          <wsa:Address>http://Passport.NET/tb</wsa:Address>
33                      </wsa:EndpointReference>
34                  </wsp:AppliesTo>
35              </wst:RequestSecurityToken>
36              <wst:RequestSecurityToken Id="RSTn">
37                  <wst:RequestType> http://schemas.xmlsoap.org/ws/2004/04/security/trust/Issue
38                  </wst:RequestType>
39                  <wsp:AppliesTo>
40                      <wsa:EndpointReference>
41                          <wsa:Address>domain</wsa:Address>
42                      </wsa:EndpointReference>
43                  </wsp:AppliesTo>
44                  <wsse:PolicyReference URI="policy␣parameter"></wsse:PolicyReference>
45              </wst:RequestSecurityToken>
46
47              ...
48              ...
49
50          </ps:RequestMultipleSecurityTokens>
51    </Body>
52  </Envelope>
```

Mediante questa operazione si comunicano al server le credenziali per l'accesso (vedi righe 19 e 20), ovvero nome utente (nella forma di indirizzo email scelto in fase di iscrizione al servizio), password ad esso associato, policy di accesso (secondo quanto ci è stato indicato dal NS con il precedente comando), ed indicazione sul servizio o sui servizi per i quali si richiede autenticazione.

Va infatti notato che questa procedura è utilizzata per l'accesso a tutta la rete di servizi pubblici Microsoft, tra cui Messenger, Hotmail, Spaces, Live Storage, etc. Nella pratica, lo schema soprastante può essere utilizzato per richiedere molteplici accessi simultanei ai servizi semplicemente aggiungendo al documento SOAP dei sotto alberi con radice <wst:RequestSecurityToken ID="**RSTn**"> seguendo l'opportuna numerazione RST0, RST1, etc.

A questo comando il server risponde con:

```
1   <S:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
2      <S:Header>
3         <!-- Useless data, omissis -->
4      </S:Header>
5      <S:Body>
6         <wst:RequestSecurityTokenResponseCollection
7            xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
8            xmlns:wst="http://schemas.xmlsoap.org/ws/2004/04/trust"
9            xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
10           xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
                 utility-1.0.xsd"
11           xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
12           xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
13           xmlns:psf="http://schemas.microsoft.com/Passport/SoapServices/SOAPFault">
14           <wst:RequestSecurityTokenResponse>
15              <wst:TokenType>urn:passport:legacy</wst:TokenType>
16              <wsp:AppliesTo xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
17                 <wsa:EndpointReference>
18                 <wsa:Address>http://Passport.NET/tb</wsa:Address>
19              </wsa:EndpointReference>
20           </wsp:AppliesTo>
21           <wst:LifeTime>
22              <wsu:Created>2006-12-06T05:12:10Z</wsu:Created>
23              <wsu:Expires>2006-12-07T05:12:10Z</wsu:Expires>
24           </wst:LifeTime>
25           <wst:RequestedSecurityToken>
26              <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
27              Id="BinaryDAToken0"
28              Type="http://www.w3.org/2001/04/xmlenc#Element">
29                 <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc
                       ">
```

```
30                      </EncryptionMethod>
31                   <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
32                      <ds:KeyName>http://Passport.NET/STS</ds:KeyName>
33                   </ds:KeyInfo>
34                   <CipherData>
35                      <CipherValue>
36                         <!--cipher data, also useless -->
37                      </CipherValue>
38                   </CipherData>
39                </EncryptedData>
40             </wst:RequestedSecurityToken>
41             <wst:RequestedTokenReference>
42                <wsse:KeyIdentifier ValueType="urn:passport"></wsse:KeyIdentifier>
43                <wsse:Reference URI="#BinaryDAToken0"></wsse:Reference>
44             </wst:RequestedTokenReference>
45             <wst:RequestedProofToken>
46                <wst:BinarySecret>ignore this one</wst:BinarySecret>
47             </wst:RequestedProofToken>
48          </wst:RequestSecurityTokenResponse>
49          <wst:RequestSecurityTokenResponse>
50             <wst:TokenType>urn:passport:compact</wst:TokenType>
51             <wsp:AppliesTo xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
52                <wsa:EndpointReference>
53                   <wsa:Address>messengerclear.live.com</wsa:Address>
54                </wsa:EndpointReference>
55             </wsp:AppliesTo>
56             <wst:LifeTime>
57                <wsu:Created>2006-12-06T05:12:10Z</wsu:Created>
58                <wsu:Expires>2006-12-06T13:12:10Z</wsu:Expires>
59             </wst:LifeTime>
60             <wst:RequestedSecurityToken>
61                <wsse:BinarySecurityToken Id="Compactn">
62                   t=<Ticket>&p=
63                </wsse:BinarySecurityToken>
64             </wst:RequestedSecurityToken>
65             <wst:RequestedTokenReference>
66                <wsse:KeyIdentifier ValueType="urn:passport:compact"></wsse:KeyIdentifier>
67                <wsse:Reference URI="#Compactn"></wsse:Reference>
68             </wst:RequestedTokenReference>
69             <wst:RequestedProofToken>
70                <wst:BinarySecret>Binary Secret</wst:BinarySecret>
71             </wst:RequestedProofToken>
72          </wst:RequestSecurityTokenResponse>
73          <wst:RequestSecurityTokenResponse>
74             <wst:TokenType>urn:passport:legacy</wst:TokenType>
75             <wsp:AppliesTo xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
76                <wsa:EndpointReference>
77                   <wsa:Address>site domain</wsa:Address>
78                </wsa:EndpointReference>
79             </wsp:AppliesTo>
80             <wst:LifeTime>
81                <wsu:Created>2006-12-06T05:12:10Z</wsu:Created>
82                <wsu:Expires>2006-12-06T05:20:30Z</wsu:Expires>
83             </wst:LifeTime>
84             <wst:RequestedSecurityToken>
85                <wsse:BinarySecurityToken Id="PPTokenn">t=<Ticket Token>&p=</wsse:
                     BinarySecurityToken>
86             </wst:RequestedSecurityToken>
87             <wst:RequestedTokenReference>
88                <wsse:KeyIdentifier ValueType="urn:passport"></wsse:KeyIdentifier>
89                <wsse:Reference URI="#PPTokenn"></wsse:Reference>
90             </wst:RequestedTokenReference>
91          </wst:RequestSecurityTokenResponse>
92
93          ...
94          ...
95
96       </wst:RequestSecurityTokenResponseCollection>
97    </S:Body>
98 </S:Envelope>
```

Come si nota, nel documento SOAP ad ogni nodo `<wst:RequestSecurityTokenResponse>` corrisponde la risposta del server per ogni richiesta di autenticazione ai servizi: alla numerazione imposta dal client con **RSTn** il server risponde con **Compactn**. Il sotto albero che contiene il segreto binario (security token) per l'autenticazione al servizio Messenger è contraddistinto

dal nodo `<wsa:Address>messengerclear.live.com</wsa:Address>`. Infine, nell'ultimo sotto albero `<wst:RequestSecurityTokenResponse>` viene riportato il Ticket Token. Diversamente dai security token i quali vengono utilizzati solamente al momento dell'autenticazione per ciascun servizio, il Ticket Token è unico ed ha validità per tutta la sessione (in caso di login eseguito con successo); con questo espediente si minimizzano gli scambi di dati tra server e client e si ha un modo sicuro per stabilire l'identità di un utente. Questa metodologia di identificazione prende il nome di **Single Sign-On (SSO)**.

4.3 Uso dei token

In questa sezione si utilizzeranno nonce e binary secret per elaborare una stringa da ritornare al NS in modo da dimostrare la nostra identità e poter accedere al servizio Messenger.

4.3.1 decodifica Base64

Per prima cosa il binary secret viene semplicemente decodificato dalla sua rappresentazione corrente in Base64. Si chiami `key1` la stringa risultante.

L'algoritmo di decifratura qui utilizzato è quello fornito dal pacchetto Bouncy Castle.

4.3.2 calcolo di key2 e key3

Si calcolano ora altre due chiavi, rispettivamente `key2` e `key3`, secondo l'algoritmo sotto riportato:

```
1 hash1 = SHA1-HMAC(key1,"WS-SecureConversationSESSION KEY HASH")
2 hash2 = SHA1-HMAC(key1,hash1+"WS-SecureConversationSESSION KEY HASH")
3 hash3 = SHA1-HMAC(key1,hash1)
4 hash4 = SHA1-HMAC(key1,hash3+"WS-SecureConversationSESSION KEY HASH")
```

`key2` viene creata come array di 24 byte: i primi 20 corrispondono ad `hash2`, gli ultimi 4 corrispondono ai primi 4 di `hash4`.

`key3` è calcolata mediante lo stesso algoritmo, tuttavia nel suo caso si utilizza la stringa "WS-SecureConversationSESSION KEY ENCRYPTION" come input dell'algoritmo di criptazione. Il calcolo della funzione di hash è stato eseguito mediante chiamata alla classe javax.crypto.Mac della libreria standard Java:

```
Mac mac = Mac.getInstance("HmacSHA1");
```

4.3.3 hash

Si applica al nonce il codice di autenticazione HMAC con algoritmo di hash SHA1 con chiave `key2`:

```
hash = SHA1-HMAC(key2, nonce)
```

4.3.4 padding del nonce

Il nonce viene ora esteso accodandovi 8 byte di valore 0x08.

4.3.5 vettore di numeri casuale

Si genera un vettore di 8 byte casuali.

4.3.6 criptazione triplo DES in modalità CBC

Si opera ora la cifratura del nonce paddato nel passo 2.3.4. Per fare questo si usa `key3` come chiave dell'algoritmo, come vettore di inizializzazione (IV) si usa il vettore di 8 byte casuali testé creato; l'output risultante sarà un vettore di 72 byte che verrà chiamato `encrypted`.

Anche in questo caso è stata usata la libreria standard di Java (javax.crypto.Cipher):

```
Cipher c2 = Cipher.getInstance( "DESede/CBC/PKCS5Padding" );
```

4.3.7 creazione struttura dati da inviare al server

Ottenuti questi dati, è stato necessario organizzarli all'interno di una struttura comprensibile dal server affinché possa avvenirne il controllo. La documentazione riportata in rete descrive la struttura in sintassi C come segue:

```
struct tagMSGRUSRKEY { // Header
   unsigned int uStructHeaderSize; // 28. Does not count data
   unsigned int uCryptMode; // CRYPT_MODE_CBC (1)
   unsigned int uCipherType; // TripleDES (0x6603)
   unsigned int uHashType; // SHA1 (0x8004)
   unsigned int uIVLen; // 8
   unsigned int uHashLen; // 20
   unsigned int uCipherLen; // 72 // Data
   unsigned char aIVBytes[8];
   unsigned char aHashBytes[20];
   unsigned char aCipherBytes[72];
}MSGUSRKEY;
```

Va notato che tutti i campi qui riportati sono da considerarsi Little-Endian; gli unsigned int sono da 4 byte (alcuni compilatori C per macchine da 64bit allocano 8 byte per tali dati). La traslazione in Java qui eseguita si è limitata alla creazione di un array di 128 byte così creato:

```
1 byte[] answer=new byte [128]; //the data header is always like this:
2 byte[] tmp=Base64.decode("HAAAAAEAAAADZgAABIAAAAgAAAAUAAAASAAAAE==");
3 // and then the body of the answer
4 for (int i=0; i<28;i++) { answer[i]=tmp[i]; } // the header
5 for (int i=28; i<36; i++) { answer[i]=rnd[i-28]; } // 8 random bytes
6 for (int i=36; i<56; i++) { answer[i]=hash[i-36]; }
7 for (int i=56; i<128; i++) { answer[i]=encrypted[i-56]; }
```

Come si può notare si è deciso di non soffermarsi sul formalismo dell'header, in quanto nelle prove effettuate si è riscontrato non essere mai soggetto a cambiamenti. Pertanto una volta identificata la forma, è stato scelto di memorizzarlo nella sua forma Base64 encoded e per poterlo riutilizzare nella creazione della struttura da inviare. Ovviamente gli altri campi sono frutto di elaborazioni eseguite su dati originati dal server e vengono ricalcolati di volta in volta.

Non è stato altresì necessaria alcuna operazione di correzione per passare da `unsigned int` agli interi Java (con segno) poiché i dati originati dagli algoritmi di criptazione non hanno subìto alcuna modifica prima di essere copiati nella struttura dati finale.

4.3.8 codifica Base64 della struttura dati

Una volta ottenuto l'array di 128 byte, lo si codifica in Base64, sempre mediante encoder del pacchetto Bouncy Castle. Chiameremo questo token `response`. La procedura di calcolo da effettuare sui token è stata portata così a termine.

4.3.9 Il codice

Di seguito si riporta l'implementazione dell'algoritmo testé esposto.

```
1  // Step 1: base64 decode the binary secret
2  byte[] key1 = Base64.decode(binarySecret);
3
4  // Step 2: Calculate key2 and key3 as:
5  byte[] key2 = deriveKey("WS-SecureConversationSESSION␣KEY␣HASH", key1);
6
7  // Step 3: For key3 we'll use "WS-SecureConversationSESSION KEY ENCRYPTION"
8  byte[] key3 = deriveKey("WS-SecureConversationSESSION␣KEY␣ENCRYPTION", key1);
9
10 // Step 4: hash = SHA1-HMAC(key2, nonce)
11 byte[] hash = new byte[20];
12 try {
13    Mac mac = Mac.getInstance("HmacSHA1");
14    SecretKeySpec secret = new SecretKeySpec(key2,"HmacSHA1");
15    mac.init(secret);
16    hash = mac.doFinal(base64nonce.getBytes());
17 } catch (Exception e) { System.out.println(e.getMessage()); }
18
19 // Step 5: pad the nonce with 8bytes which value is 0x08
20 // and creates a byte[8] of random values
21 byte[] nonceBytes = new byte[base64nonce.length()+8];
22 byte[] tmp1=base64nonce.getBytes();
23 for (int i=0; i<64; i++)
24    nonceBytes[i] = tmp1[i];
25
26 // and pads it
27 for (int i=0; i<8; i++)
28    nonceBytes[i+64] = 0x08;
29
30 byte[] rnd = new byte[8];
31 Random randomizer = new Random();
32 randomizer.nextBytes(rnd);
33
34 // Step 6: Triple DES the key
35 byte[] encrypted = new byte[72];
36 AlgorithmParameterSpec paramSpec = new IvParameterSpec(rnd);
37 DESedeKeySpec keyspec = null;
38 try {
39    keyspec = new DESedeKeySpec(key3);
40 } catch (InvalidKeyException e) { e.printStackTrace(); }
41 SecretKeyFactory keyfactory = null;
42 try {
43    keyfactory = SecretKeyFactory.getInstance("DESede");
44 } catch (NoSuchAlgorithmException e) { e.printStackTrace(); }
45 SecretKey key = null;
46 try {
47    key = keyfactory.generateSecret(keyspec);
48 } catch (InvalidKeySpecException e) { e.printStackTrace(); }
49 Cipher c2 = null;
50 try {
51    c2 = Cipher.getInstance( "DESede/CBC/PKCS5Padding" );
52 } catch (NoSuchAlgorithmException e) { e.printStackTrace(); }
```

```
53 catch (NoSuchPaddingException e) { e.printStackTrace(); }
54 try {
55    c2.init(Cipher.ENCRYPT_MODE, key, paramSpec);
56 } catch (InvalidKeyException e) { e.printStackTrace(); }
57 catch (InvalidAlgorithmParameterException e) { e.printStackTrace(); }
58 try {
59    encrypted = c2.doFinal(nonceBytes);
60 } catch (IllegalBlockSizeException e) { e.printStackTrace(); }
61 catch (BadPaddingException e) { e.printStackTrace(); }
62
63 // Step 7: create the final data object
64 byte[] answer = new byte [128];
65 //the data header is always like this:
66 byte[] tmp = Base64.decode("HAAAAAEAAAADZgAABIAAAAgAAAAUAAAASAAAAE==");
67
68 // and then the body of the answer
69 for (int i=0; i<28;i++) answer[i]=tmp[i];
70 for (int i=28; i<36; i++) answer[i]=rnd[i-28];
71 for (int i=36; i<56; i++) answer[i]=hash[i-36];
72 for (int i=56; i<128; i++) answer[i]=encrypted[i-56];
73
74 // Step 8: Base64 encode the data
75 byte[] responseBytes = Base64.encode(answer);
76 String response = new String (responseBytes);
```

La realizzazione di questa parte del lavoro si basa ancora una volta sulla documentazione fornita dalla wiki [MSNFanatic(2011b)]. La prima realizzazione di questo algoritmo si rifà ad un lavoro di *reverse engineering* come spiegato in [oleavr(2006)]. L'autore afferma di aver utilizzato come unico strumento il software oSpy (`http://code.google.com/p/ospy/`) per la sua ricostruzione.

4.4 Fine transazione con il NS

È ora possibile portare a termine la transazione con il NS, mediante il seguente scambio di messaggi:

```
>>> USR <TrID> SSO S <Ticket Token> <response>\r\n
<<< USR <TrID> OK <email> <verified> 0\r\n
```

Nel messaggio da inviare al server sono contenuti il Ticket, così come è stato fornito dal server di autenticazione e la struttura dati elaborata nel passo precedente. In caso di autenticazione eseguita con successo, il server risponde con un OK, seguito dalla mail dell'utente e dal valore della variabile `verified` posto ad 1. A questo punto il login alla rete Messenger è completato; al client non rimane che aggiornare le proprie liste di contatti e di rendere pubblica la sua presenza.

5 Sincronizzazione iniziale: Membership List ed Address Book

Dopo l'aggiornamento della procedura di autenticazione alla versione 15, si è reso disponibile un potente strumento per migliorare la gestione della connessione ed in particolare per la richiesta delle liste dei contatti: la gestione a Single Sign-On con ticket permette infatti una più agevole gestione dei contatti e delle azioni ad essi associati, senza la necessità di volta in volta di negoziare un cookie di autenticazione con il NS prima di eseguire ogni singola azione. Avendo il ticket disponibile per tutta la sessione, il suo utilizzo annulla il numero delle connessioni richieste per certificare la propria identità.

Terminata l'autenticazione si procede con la sincronizzazione iniziale del client: vanno eseguite in sequenza l'aggiornamento della Membership List e dell'Address Book. Entrambe queste azioni sono invocate dalla classe MsnGuiInterpreter durante procedura di login. La loro effettiva esecuzione è demandata alla classe SOAPManager per quanto riguarda l'invio della richiesta al server e la ricezione della risposta da quest'ultimo, ed alla classe SOAPPArser per l'analisi della risposta.

5.1 Membership List

La richiesta per la Membership List si esegue con il POST all'URL `http://contacts.msn.com/abservice/SharingService.asmx` del seguente oggetto SOAP:

```
Content-Type: text/xml, charset=utf-8
Connection: Keep-Alive
Cache-Control: no-cache
SOAPAction: http://www.msn.com/webservices/AddressBook/FindMembership
Host: contacts.msn.com
Content-Length: String.valueOf(message.length))

<?xml version='1.0' encoding='utf-8'?>
<soap:Envelope xmlns:soap=\http://schemas.xmlsoap.org/soap/envelope/\>
  <soap:Header xmlns:soap=\http://schemas.xmlsoap.org/soap/envelope/\>
    <ABApplicationHeader xmlns=\http://www.msn.com/webservices/AddressBook\>
          <ApplicationId xmlns=\http://www.msn.com/webservices/AddressBook\>
      CFE80F9D-180F-4399-82AB-413F33A1FA11
    </ApplicationId>
      <IsMigration xmlns=\http://www.msn.com/webservices/AddressBook\>false</IsMigration>
      <PartnerScenario xmlns=\http://www.msn.com/webservices/AddressBook\>Initial</
           PartnerScenario>
    </ABApplicationHeader>
    <ABAuthHeader xmlns=\http://www.msn.com/webservices/AddressBook\>
      <ManagedGroupRequest xmlns=\http://www.msn.com/webservices/AddressBook\>
        false
      </ManagedGroupRequest>
      <TicketToken> ticket </TicketToken>
    </ABAuthHeader>
  </soap:Header>
  <soap:Body xmlns:soap=\http://schemas.xmlsoap.org/soap/envelope/\>
    <FindMembership xmlns=\http://www.msn.com/webservices/AddressBook\>
      <serviceFilter xmlns=\http://www.msn.com/webservices/AddressBook\>
        <Types xmlns=\http://www.msn.com/webservices/AddressBook\>
          <ServiceType xmlns=\http://www.msn.com/webservices/AddressBook\>Messenger</
               ServiceType>
          <ServiceType xmlns=\http://www.msn.com/webservices/AddressBook\>Invitation</
               ServiceType>
          <ServiceType xmlns=\http://www.msn.com/webservices/AddressBook\>SocialNetwork</
               ServiceType>
          <ServiceType xmlns=\http://www.msn.com/webservices/AddressBook\>Space</ServiceType>
          <ServiceType xmlns=\http://www.msn.com/webservices/AddressBook\>Profile</
               ServiceType>
        </Types>
      </serviceFilter>
    </FindMembership>
  </soap:Body>
</soap:Envelope>
```

Come si può notare l'unica informazione che si trasmette al server è il Ticket ottenuto al login, in base ad esso il server riconosce l'utente e gli invia le informazioni corrette. La risposta del server ha una forma come segue:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Header>
    <ServiceHeader xmlns="http://www.msn.com/webservices/AddressBook">
      <Version>15.04.1114.0000</Version>
      <CacheKey>
14r2;lGDEA5Gkyc2MW2HdypHpx3RQbKOYPzi8Rd1uJNGOB/0xVS3xEsVOP/d+
     PL23oLbONg7znjBxBuCIJilaz8C6xifTfnjfu7d8Mr2IpJ4QfM81vmnMR+RnPne3vDpRr98HF/
     UNCIdmEwI4FOpgtGTkmhvTyVFdvxBhFyer6eVwGKI=
```

```
10        </CacheKey>
11        <CacheKeyChanged>true</CacheKeyChanged>
12        <PreferredHostName>proxy-bay.contacts.msn.com</PreferredHostName>
13        <SessionId>ABCH.9b61b6cd-c09a-4bb7-9429-9bec9b4e3204</SessionId>
14      </ServiceHeader>
15    </soap:Header>
16    <soap:Body>
17      <FindMembershipResponse xmlns="http://www.msn.com/webservices/AddressBook">
18        <FindMembershipResult>
19          <Services>
20            <Service>
21              <Memberships>
22                <Membership>
23                  <MemberRole>Reverse</MemberRole>
24                  <Members>
25                    <Member xsi:type="PassportMember">
26                      <MembershipId>23</MembershipId>
27                      <Type>Passport</Type>
28                      <DisplayName>buddy1</DisplayName>
29                      <State>Accepted</State>
30                      <Deleted>false</Deleted>
31                      <LastChanged>2010-07-27T13:30:38.56-07:00</LastChanged>
32                      <JoinedDate>2010-07-27T13:30:38.463-07:00</JoinedDate>
33                      <ExpirationDate>0001-01-01T00:00:00</ExpirationDate>
34                      <Changes />
35                      <PassportName>buddy1@hotmail.com</PassportName>
36                      <IsPassportNameHidden>false</IsPassportNameHidden>
37                      <PassportId>0</PassportId>
38                      <CID>-4695065377381910654</CID>
39                      <PassportChanges />
40                      <LookedupByCID>false</LookedupByCID>
41                    </Member>
42                  </Members>
43                  <MembershipIsComplete>true</MembershipIsComplete>
44                </Membership>
45                <Membership>
46                  <MemberRole>Pending</MemberRole>
47                  <Members>
48                    <Member xsi:type="PassportMember">
49                      <MembershipId>22</MembershipId>
50                      <Type>Passport</Type>
51                      <DisplayName>buddy2</DisplayName>
52                      <State>Accepted</State>
53                      <Deleted>false</Deleted>
54                      <LastChanged>2010-07-27T13:30:38.56-07:00</LastChanged>
55                      <JoinedDate>2010-07-27T13:30:38.463-07:00</JoinedDate>
56                      <ExpirationDate>0001-01-01T00:00:00</ExpirationDate>
57                      <Changes />
58                      <PassportName>buddy2@hotmail.com</PassportName>
59                      <IsPassportNameHidden>false</IsPassportNameHidden>
60                      <PassportId>0</PassportId>
61                      <CID>-4695065377381910654</CID>
62                      <PassportChanges />
63                      <LookedupByCID>false</LookedupByCID>
64                    </Member>
65                  </Members>
66                </Membership>
67
68                ...
69
70              </Memberships>
71              <Info>
72                <Handle>
73                  <Id>5</Id>
74                  <Type>SocialNetwork</Type>
75                  <ForeignId />
76                </Handle>
77                <InverseRequired>false</InverseRequired>
78                <AuthorizationCriteria>TrustLevel3FMBR;OwnerRM</AuthorizationCriteria>
79                <IsBot>false</IsBot>
80              </Info>
81              <Changes />
82              <LastChange>2010-11-26T02:49:52.747-08:00</LastChange>
83              <CreatedDate>2010-10-08T11:02:28.903-07:00</CreatedDate>
84              <Deleted>false</Deleted>
85            </Service>
```

```
86          </Services>
87          <OwnerNamespace>
88            <Info>
89              <Handle>
90                <Id>00000000-0000-0000-0000-000000000000</Id>
91                <IsPassportNameHidden>false</IsPassportNameHidden>
92                <CID>0</CID>
93              </Handle>
94              <CreatorPuid>0</CreatorPuid>
95              <CreatorCID>-79262244472721887l1</CreatorCID>
96              <CreatorPassportName>userMail@hotmail.com</CreatorPassportName>
97              <CircleAttributes>
98                <IsPresenceEnabled>false</IsPresenceEnabled>
99                <Domain>WindowsLive</Domain>
100             </CircleAttributes>
101             <MessengerApplicationServiceCreated>true</MessengerApplicationServiceCreated>
102           </Info>
103           <Changes />
104           <CreateDate>2010-07-27T09:29:15.22-07:00</CreateDate>
105           <LastChange>2010-12-10T01:08:00.75-08:00</LastChange>
106         </OwnerNamespace>
107       </FindMembershipResult>
108     </FindMembershipResponse>
109   </soap:Body>
110 </soap:Envelope>
```

In questo documento sono contenuti tutti i contatti dell'utente, suddivisi per ruoli. Dai nodi `<MemberRole>` dipartono infatti dei sotto alberi contenenti per ogni lista gli utenti ed i loro dati principali. Le liste più importanti sono:

- **Forward**: la lista forward è la lista principale dei contatti; dei contatti qui presenti è stata eseguita la "sottoscrizione della presenza", ovverosia si è fatta richiesta di ricevere tutte le informazioni personali e quelle che riguardo il loro stato (in linea, disconnesso, occupato, etc...).

- **Reverse**: costituisce la lista di contatti che hanno il nostro nominativo inserito nella loro forward list; si tratta di una lista informativa e non può essere chiaramente soggetta a modifiche da parte dell'utente.

- **Allow**: è la lista di contatti a cui è concesso di essere notificati sulla nostra presenza online; si noti la differenza tra reverse list, ovvero degli gli utenti che chiedono di poter essere informati sul nostro stato, ed allowed list, in cui compaiono solo quelli che si decide di tenere informati.

- **Block**: ovvero la lista dei contatti a cui non è concesso di ricevere notifiche sul nostro stato di presenza; questi non hanno la possibilità di vedere nemmeno gli aggiornamenti sui messaggi personali e se tentano di instaurare una sessione di chat vengono bloccati dal SB con un messaggio di errore "utente non in linea". Ovviamente la block list è disgiunta dalla allow list, ciononostante esiste un codice di errore apposito (il **219**) in risposta al tentativo di inserire un contatto in ambo le liste.

- **Pending**: la lista dei contatti che hanno inoltrato una richiesta di amicizia al nostro contatto e non sono ancora stati inseriti né nella allow né nella block list, pur essendo nella reverse.

5.2 Address Book

L'azione da eseguire mediante SOAP per la richiesta dell'Address Book è:

```
1 Content-Type: text/xml, charset=utf-8
2 Connection: Keep-Alive
3 Cache-Control: no-cache
4 SOAPAction: http://www.msn.com/webservices/AddressBook/FindMembership
5 Host: contacts.msn.com
```

```
Content-Length: String.valueOf(message.length))

<?xml version=\1.0\ encoding=\utf-8\?>
<soap:Envelope
xmlns:soap=\http://schemas.xmlsoap.org/soap/envelope/\
xmlns:xsi=\http://www.w3.org/2001/XMLSchema-instance\
xmlns:xsd=\http://www.w3.org/2001/XMLSchema\
xmlns:soapenc=\http://schemas.xmlsoap.org/soap/encoding/\>
  <soap:Header>
    <ABApplicationHeader xmlns=\http://www.msn.com/webservices/AddressBook\>
      <ApplicationId>CFE80F9D-180F-4399-82AB-413F33A1FA11</ApplicationId>
      <IsMigration>false</IsMigration>
      <PartnerScenario>Initial</PartnerScenario>
    </ABApplicationHeader>
    <ABAuthHeader xmlns=\http://www.msn.com/webservices/AddressBook\>
      <ManagedGroupRequest>false</ManagedGroupRequest>
      <TicketToken>  ticket </TicketToken>
    </ABAuthHeader>
  </soap:Header>
  <soap:Body>
  <ABFindAll xmlns=\http://www.msn.com/webservices/AddressBook\>
      <abId>00000000-0000-0000-0000-000000000000</abId>
      <abView>Full</abView>
      <deltasOnly>false</deltasOnly>
      <lastChange>0001-01-01T00:00:00.0000000-08:00</lastChange>
    </ABFindAll>
  </soap:Body>
</soap:Envelope>
```

e la relativa risposta inviata dal server:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.
    w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <soap:Header>
      <ServiceHeader xmlns="http://www.msn.com/webservices/AddressBook">
         <Version>16.00.1219.0001</Version>
         <CacheKey><!-- useless --></CacheKey>
         <CacheKeyChanged>true</CacheKeyChanged>
         <PreferredHostName>proxy-bay.contacts.msn.com</PreferredHostName>
         <SessionId>ABCH.4e7a061c-1352-483c-a377-2ee6d68e8669</SessionId>
      </ServiceHeader>
   </soap:Header>
   <soap:Body>
      <ABFindAllResponse xmlns="http://www.msn.com/webservices/AddressBook">
         <ABFindAllResult>
            <groups>
               <Group>
                  <groupId>da6c1de2-cda9-49de-b2eb-b53eb16bece9</groupId>
                  <groupInfo>
                     <groupType>c8529ce2-6ead-434d-881f-341e17db3ff8</groupType>
                     <name>Preferiti</name>
                     <IsNotMobileVisible>false</IsNotMobileVisible>
                     <IsPrivate>false</IsPrivate>
                     <IsFavorite>true</IsFavorite>
                  </groupInfo>
                  <propertiesChanged />
                  <fDeleted>false</fDeleted>
                  <lastChange>2010-12-03T08:49:29.037-08:00</lastChange>
               </Group>
            </groups>
            <contacts>
               <Contact>
                  <contactId>c5bd42a3-f949-46c3-ba24-295318cfb798</contactId>
                  <contactInfo>
                     <annotations>
                        <Annotation>
                           <Name>MSN.IM.MBEA</Name>
                           <Value>0</Value>
                        </Annotation>
                        <Annotation>
                           <Name>MSN.IM.GTC</Name>
                           <Value>1</Value>
                        </Annotation>
                        <Annotation>
                           <Name>MSN.IM.BLP</Name>
                           <Value>0</Value>
```

```
46                          </Annotation>
47                          <Annotation>
48                              <Name>Live.Locale</Name>
49                              <Value>1040</Value>
50                          </Annotation>
51                          <Annotation>
52                              <Name>Live.Profile.Expression.LastChanged</Name>
53                              <Value>2010-12-10T01:07:38.34-08:00</Value>
54                          </Annotation>
55                          <Annotation>
56                              <Name>Live.Passport.Birthdate</Name>
57                              <Value>01/01/1974 00:00:00</Value>
58                          </Annotation>
59                          <Annotation>
60                              <Name>Live.Passport.Country</Name>
61                              <Value>IT</Value>
62                          </Annotation>
63                      </annotations>
64                      <contactType>Me</contactType>
65                      <quickName>ME</quickName>
66                      <firstName>don</firstName>
67                      <lastName>chisciotte</lastName>
68                      <passportName>im-001@hotmail.it</passportName>
69                      <IsPassportNameHidden>false</IsPassportNameHidden>
70                      <displayName>don</displayName>
71                      <puid>0</puid>
72                      <CID>-7926224447272188711</CID>
73                      <IsNotMobileVisible>true</IsNotMobileVisible>
74                      <isMobileIMEnabled>false</isMobileIMEnabled>
75                      <isMessengerUser>false</isMessengerUser>
76                      <isFavorite>false</isFavorite>
77                      <isSmtp>false</isSmtp>
78                      <hasSpace>false</hasSpace>
79                      <spotWatchState>NoDevice</spotWatchState>
80                      <birthdate>0001-01-01T00:00:00</birthdate>
81                      <primaryEmailType>ContactEmailPersonal</primaryEmailType>
82                      <PrimaryLocation>ContactLocationPersonal</PrimaryLocation>
83                      <PrimaryPhone>ContactPhonePersonal</PrimaryPhone>
84                      <FileAs>chisciotte, don</FileAs>
85                      <IsPrivate>false</IsPrivate>
86                      <IsHidden>false</IsHidden>
87                      <Gender>Unspecified</Gender>
88                      <TimeZone>None</TimeZone>
89                      <PublicDisplayName>don</PublicDisplayName>
90                      <IsAutoUpdateDisabled>false</IsAutoUpdateDisabled>
91                      <IsShellContact>false</IsShellContact>
92                      <TrustLevel>0</TrustLevel>
93                      <PropertiesChanged />
94                  </contactInfo>
95                  <propertiesChanged />
96                  <fDeleted>false</fDeleted>
97                  <CreateDate>2010-07-27T09:29:00-07:00</CreateDate>
98                  <lastChange>2011-02-08T04:53:17.29-08:00</lastChange>
99                  <LastModifiedBy>8</LastModifiedBy>
100             </Contact>
101             <Contact>
102                 <contactId>90dd2299-c19b-4f51-90cd-776d82a14472</contactId>
103                 <contactInfo>
104                     <contactType>LivePending</contactType>
105                     <quickName>buddyName</quickName>
106                     <passportName>buddy@mail.com</passportName>
107                     <IsPassportNameHidden>false</IsPassportNameHidden>
108                     <displayName>MyBro</displayName>
109                     <puid>0</puid>
110                     <CID>-1273360585163271030</CID>
111                     <IsNotMobileVisible>false</IsNotMobileVisible>
112                     <isMobileIMEnabled>false</isMobileIMEnabled>
113                     <isMessengerUser>true</isMessengerUser>
114                     <isFavorite>false</isFavorite>
115                     <isSmtp>false</isSmtp>
116                     <hasSpace>false</hasSpace>
117                     <spotWatchState>NoDevice</spotWatchState>
118                     <birthdate>0001-01-01T00:00:00</birthdate>
119                     <primaryEmailType>ContactEmailPersonal</primaryEmailType>
120                     <PrimaryLocation>ContactLocationPersonal</PrimaryLocation>
121                     <PrimaryPhone>ContactPhonePersonal</PrimaryPhone>
```

```
122                     <IsPrivate>false</IsPrivate>
123                     <IsHidden>false</IsHidden>
124                     <Gender>Unspecified</Gender>
125                     <TimeZone>None</TimeZone>
126                     <IsAutoUpdateDisabled>false</IsAutoUpdateDisabled>
127                     <IsShellContact>false</IsShellContact>
128                     <TrustLevel>2</TrustLevel>
129                     <PropertiesChanged />
130                  </contactInfo>
131                  <propertiesChanged />
132                  <fDeleted>false</fDeleted>
133                  <CreateDate>2010-10-16T06:50:00-07:00</CreateDate>
134                  <lastChange>2010-10-16T06:50:20.15-07:00</lastChange>
135                  <CreatedBy>7</CreatedBy>
136                  <LastModifiedBy>7</LastModifiedBy>
137               </Contact>
138
139            ...
140
141            </contacts>
142            <ab>
143               <abId>00000000-0000-0000-0000-000000000000</abId>
144               <abInfo>
145                  <MigratedTo>6</MigratedTo>
146                  <ownerPuid>0</ownerPuid>
147                  <OwnerCID>-7926224447272188711</OwnerCID>
148                  <ownerEmail>im-001@hotmail.it</ownerEmail>
149                  <fDefault>true</fDefault>
150                  <joinedNamespace>false</joinedNamespace>
151                  <IsBot>false</IsBot>
152                  <IsParentManaged>false</IsParentManaged>
153                  <AccountTierLastChanged>0001-01-01T00:00:00</AccountTierLastChanged>
154                  <ProfileVersion>0</ProfileVersion>
155                  <SubscribeExternalPartner>false</SubscribeExternalPartner>
156                  <NotifyExternalPartner>false</NotifyExternalPartner>
157                  <AddressBookType>Individual</AddressBookType>
158                  <MessengerApplicationServiceCreated>true</
                         MessengerApplicationServiceCreated>
159                  <IsBetaMigrated>false</IsBetaMigrated>
160               </abInfo>
161               <lastChange>2011-02-08T04:53:17.29-08:00</lastChange>
162               <DynamicItemLastChanged>0001-01-01T00:00:00</DynamicItemLastChanged>
163               <RecentActivityItemLastChanged>0001-01-01T00:00:00</
                      RecentActivityItemLastChanged>
164               <createDate>2010-07-27T09:29:15.22-07:00</createDate>
165               <propertiesChanged />
166            </ab>
167         </ABFindAllResult>
168      </ABFindAllResponse>
169   </soap:Body>
170 </soap:Envelope>
```

Come si può vedere, mentre la Membership List fornisce i nominativi conosciuti dall'utente, suddividendoli per gruppi e liste in base a diritti di visibilità reciproca, lo scopo dell'Address Book è quello di fornire informazioni esaustive sui singoli contatti conosciuti. Mediante questo documento vengono inoltre rese note al client le informazioni relative all'utente che si sta collegando alla rete Messenger: per identificare questo utente speciale distinguendolo dai suoi contatti vengono utilizzati i nodi `<contactType>Me</contactType>` e `<quickName>ME</quickName>`. Questo espediente rende possibile all'utente l'utilizzo di più client che in automatico hanno sempre le informazioni personali aggiornate.

Per ovviare alla grandezza del documento inviato in caso di elevato numero di contatti, il recupero dell'Address Book può essere realizzato anche solo parzialmente. In questo caso dopo la sincronizzazione iniziale al primo login con un dato client si possono salvare in locale i nominativi, e per le successive connessioni sarà necessario fare una richiesta per le sole modifiche, indicando nel nodo `<lastChange>` quando è stata fatta l'ultima sincronizzazione (per la prima richiesta si indica `<lastChange>0001-01-01T00:00:00.0000000-08:00</lastChange>` come timestamp).

Implementazione

Sia per la Membership List che per l'Address Book è stato utilizzato il medesimo approccio nella progettazione e nello sviluppo di codice per la corretta gestione di liste, sottoliste, singoli utenti e relative proprietà. Entrambe le procedure condividono la struttura realizzativa: la loro invocazione comporta la chiamata alla classe SOAPManager che è deputata alla creazione del comando SOAP da inviare, all'instaurazione della connessione http al server, alla ricezione del messaggio di risposta dal server ed infine al salvataggio di quest'ultimo su disco.

Una volta effettuate questi passaggi, viene chiamata la classe SOAPPArser che si fa carico di interpretare la struttura della risposta mediante lettura dell'albero XML in esso contenuta. Questo passaggio viene realizzato creando un documento DOM parsando i file salvati mediante la classe DocumentBuilder (`javax.xml.parsers`)

```
1 DocumentBuilder builder =
2    DocumentBuilderFactory.newInstance().newDocumentBuilder();
3 File f = new File(root + "im/conf/membership.xml");
4 Document document = builder.parse(f);
```

Una volta creato l'albero del documento si estraggono le liste di utenti, facendo pesante uso del metodo `Document.getElementsByTagName(String tagname)`.

Estrazione Membership List

L' algoritmo per l'estrazione delle liste è il seguente

```
1 cerca i nodi con tag "MemberRole"
2 per ogni valore di tali nodi
3    cerca i nodi con tag "PassportName"
4    per ogni nodo trovato
5       aggiungi l'utente alla lista che si sta scorrendo
```

La sua implementazione in Java è risultata abbastanza banale, eccetto che per il nesting dei due cicli `for`: il metodo `getElementsByTagName` infatti esegue la ricerca all'interno di tutto un documento, e pertanto è stato necessario all'inizio di ogni ciclo `for` esterno creare un nuovo albero DOM su cui effettuare la ricerca:

```
1  NodeList nl = document.getElementsByTagName("MemberRole");
2     for (int i=0; i<nl.getLength(); i++) {
3        String listName = nl.item(i).getFirstChild().getNodeValue
             ();
4        // Creates the subtree
5        Document doc = createDomDocument(nl.item(i).getParentNode
             ());
6        NodeList contacts = doc.getElementsByTagName("PassportName
             ");
7        for (int j=0; j<contacts.getLength(); j++) {
8           // add the contact to the list listName
9           ...
10       }
11    }
```

Nessuna modifica è stata apportata al codice che gestisce le liste, poiché già presente e funzionante a partire dall'implementazione del protocollo versione 9.

Estrazione Address Book

La lettura dei dati contenuti nell'Address Book avviene con le stesse metodologie impiegate nel caso precedente. Non essendo qui presenti differenti liste da scorrere però l'implementazione è stata semplificata. Si è comunque ricorso allo stesso espediente di creare dei nuovi documenti per ogni contatto in modo da poter impiegare il metodo `getElementsByTagName` per la ricerca dei singoli nodi contenenti le diverse proprietà di un contatto.

Anche in questo caso i dati estratti sono stati memorizzati su strutture già implementate da versioni precedenti del client.

Gestione dei contatti

Con metodologie simili a quelle sin qui incontrate, il protocollo MSN permette di gestire ambedue le liste, fornendo ai client strumenti per aggiungere, rimuovere e modificare contatti.

Lo sviluppo di queste funzionalità e stato considerato da chi scrive non di primaria importanza ai fini di ottenere un client immediatamente funzionante e fruibile da parte dell'utente. Si è preferito infatti affrontare e risolvere altre problematiche relative alla stabilità della connessione (si veda ad esempio la prossima sezione) o implementare altre funzionalità a più alta priorità.

6 Server pings - Challenges

Avendo come obbiettivo primario quello di consentire una connessione stabile e duratura alla rete Messenger da parte del nostro client, si è reso necessario lavorare sulla corretta gestione dei challenge da parte del Notification Server.

Mentre il client è nella rete MSN, il NS gli invia ad intervalli casuali dei comandi di ping che hanno il duplice scopo di testare lo stato della connessione e al contempo di verificare l'autenticità del client connesso. Sulle motivazioni legate a questa verifica si possono solamente formulare delle ipotesi; per questo argomento si rimanda alla sezione conclusiva

Lo scambio di comandi avviene secondo la sintassi:

```
<<< CHL 0 <challenge string>\r\n
>>> QRY <TrID> <userEmail> 32\r\n <response string>
<<< QRY <TrID client>\r\n
```

Il comando di ping ha due parametri: il TrID, che è sempre posto uguale a zero, ed una ***challenge string*** formata da 20 cifre numeriche. La risposta sarà un comando con payload avente TrID (come richiesto per ogni comando inviato ad un server), nome dell'account Passport nella forma di email dell'utente e dimensione del payload; quest'ultimo costituisce la risposta al challenge, e va computato secondo l'algoritmo che si andrà ad esporre tra breve.

Se la risposta al challenge è corretta, il server risponde con un echo del comando QRY, avente come unico parametro il TrID del comando inviato dal client. In caso di errore nella risposta il server risponde inviando un codice di errore **540** a cui segue la disconnessione immediata. Una condizione d'errore si crea nei seguenti casi (fatta salva l'ipotesi di errore sintattico, a cui segue di norma la disconnessione senza nemmeno messaggio d'errore):

- risposta troppo lenta da parte del client (più di 50 secondi);
- stringa di risposta calcolata in modo errato;
- nome utente non valido.

La gestione di questo challenge, già implementata nel client, è stata totalmente riscritta a seguito di pesanti variazioni apportate nella versione di protocollo 15. Per comodità di test e di debug, si è scelto di raggruppare il codice che esegue tale funzione in una nuova classe chiamata

MsnChallengeHandler, invocata quando necessario da MsnMessageActuator in risposta al CHL del server.

6.1 Calcolo della risposta al challenge

Per poter calcolare la risposta, sono necessari oltre alla stringa di challenge anche due codici identificativi che sono propri di ogni build del client ufficiale (a partire da *MSN Messenger* agli ultimi *Live Messenger*): il "client ID string" ed il "client ID code". Attualmente per il client di PariPari vengono utilizzati i codici di "Windows Live Messenger 8.1" client build 8.1.0168.00:

```
String IDString = "PROD0113H11T8$X_";
String IDCode = "RG@XY*28Q5QHS%Q5";
```

Con questi dati si può iniziare l'elaborazione.

6.1.1 IDString: hash MD5

Si calcola innanzitutto un hash in MD5 della concatenazione tra challenge e IDString

```
hash = MD5(challenge + IDString)
```

il risultato è trasformato in una stringa di cifre esadecimali mediante chiamata del metodo statico

```
java.lang.Integer.toHexString(0xFF & hash[i])
```

su ogni byte del digest ed accodando i risultati in una stringa che è poi convertita in 4 array da 4 long ciascuno

```
java.lang.Long.parseLong(reverseHex(Long.toHexString(num)), 16);
```

questo risultato è infine sottoposto ad una operazione di AND con il valore 0x7FFFFFFF.

6.1.2 IDCode: divisione in DWord

Si crea una nuova stringa data dalla concatenazione di challenge e IDCode e la si rende di lunghezza pari ad un multiplo di 8 mediante padding con zeri (zeri in formato stringa, ovvero "0", non byte nulli). La stringa risultante viene divisa in parti da 4 byte, ciascuna delle quali deve essere interpretata come una DWord a rappresentazione testuale, ovvero per ogni byte di ogni array si esegue l'operazione

```
hex += Integer.toHexString((int)char);
```

6.1.3 Creazione chiave a 64 bit

E' necessario creare una chiave da 64 bit da usare nei passaggi seguenti. Per questo si cicla sugli array di byte creati nel passo precedente; creo due variabili `high = low = 0`, poi per ogni iterazione N:

- si crea una variabile temporanea `temp` contenente il valore dell'array corrente (N), la si moltiplica per 0x0E79A9C1 e se ne calcola il modulo su 0x7FFFFFFF per evitare overflow o inversione di segno; al risultato si aggiunge il valore di high;
- `temp` viene moltiplicata per la prima parte dell'hash MD5 ed al risultato si somma la seconda parte dello stesso hash; di nuovo se ne calcola il modulo su 0x7FFFFFFF;

- si aggiorna poi il valore di `high`: lo si aumenta del valore del (N+1)-esimo array, al solito calcolando il modulo su 0x7FFFFFFF;
- `high` viene quindi moltiplicata per la terza parte dell'hash MD5 e gli viene sommata la quarta parte di detto hash; segue il consueto modulo anti overflow/cambio di segno;
- si aggiorna infine il valore di `low`, la quale viene incrementata di un valore pari alla somma di `temp` ed `high`; di questo valore non si calcola il modulo su 0x7FFFFFFF.

Per chiarezza si riporta qui il codice Java per quanto testé esposto:

```
1     long high = 0, low = 0;
2     for (int i = 0; i < splitChallenge.length; i = i + 2) {
3        long temp = splitChallenge[i];
4        temp = (0x0E79A9C1 * temp) % 0x7FFFFFFF;
5        temp += high;
6        temp = splitHash[0] * temp + splitHash[1];
7        temp = temp % 0x7FFFFFFF;
8
9        high = splitChallenge[i + 1];
10       high = (high + temp) % 0x7FFFFFFF;
11       high = splitHash[2] * high + splitHash[3];
12       high = high % 0x7FFFFFFF;
13
14       low += high + temp;
15    }
```

Finito questo ciclo, alle variabili `high` e `low` vengono aggiunti rispettivamente il secondo ed il quarto pezzo dell'hash MD5, e di nuovo per entrambe si calcola il modulo su 0x7FFFFFFF. Fatto questo la chiave da 64 bit che si cercava è pronta: `high` e `low` hanno questo nome poiché rappresentano rispettivamente la DWord "alta" e quella "bassa" di una QWord da 64 bit. È pertanto sufficiente shiftare `high` di 32 bit a sinistra ed aggiungere il risultato a `low`.

6.1.4 Creazione risposta

La parte finale della computazione della risposta da inviare al server consiste fondamentalmente nell'operare uno XOR tra la chiave a 64 bit e l'hash calcolato nel punto iniziale.

L'implementazione di questa operazione è avvenuta tenendo separate le due variabili `high` e `low` del passo precedente ed eseguendo per ciascuna separatamente lo XOR con 2 delle 4 parti in cui si era diviso l'hash.

L'ultimo passaggio prevede la conversione del risultato ottenuto ad una rappresentazione decimale in formato stringa. Il tutto è stato riassunto nel seguente passaggio:

```
long highlow = (i == 0 || i == 2) ? high : low;
s += Long.toHexString(Long.parseLong(hash.substring(8 * i, 8 * i + 8), 16) ^ highlow);
```

A questo punto non rimane che assemblare il messaggio di risposta come da sintassi vista ad inizio capitolo inserendo nel payload la stringa qui creata.

7 Implementazione chat

Il lavoro svolto sinora ha dato al client MSN di PariPari la capacità di collegarsi alla rete Messenger correttamente e di rimanervi per un tempo indefinito, superando i controlli imposti dal

NS. Da questa situazione l'obiettivo successivo è stato quello di reintegrare la capacità di chat secondo le specifiche del protocollo adottato.

Si è trattato, in buona sostanza, di analizzare tutte le modifiche di grammatica, sintassi e semantica apportate dagli sviluppatori Microsoft ad alcuni dei comandi che permettono l'instaurazione e l'effettiva realizzazione di una sessione di chat tra due o più utenti. Con l'occasione si è avuto modo di intervenire su parti di codice già esistenti per correggerne e migliorarne alcuni comportamenti.

Ad una prima parte di capitolo che introdurrà brevemente i comandi principali, seguirà una parte in cui verranno presentati alcuni degli interventi eseguiti sul codice delle classi che implementano questo aspetto del plugin.

7.1 MSN: gestione delle chat

I tre momenti fondamentali in una sessione di chat si dividono in instaurazione attiva di una chat, accettazione passiva e scambio vero e proprio di messaggi. Verrà anche trattato come chiudere "elegantemente"[5] una sessione, sebbene questa operazione non sia strettamente necessaria: nella totalità dei casi è sufficiente lasciar andare in timeout la connessione con gli SB, oppure chiuderla direttamente con i NS.

7.1.1 Avvio di una sessione di chat

Si supponga di essere loggati al servizio Messenger come utente alice@hotmail.com e nick Alice93 e di voler instaurare una sessione di chat con il nostro contatto bob@hotmail.com avente nick Bob83. La sequenza di comandi tra client è server sarà questa:

```
1  >>> XFR <TrID> SB
2  <<< XFR <TrID> SB <SB_ip_address:SB_port> CKI <auth_string> U
       messenger.msn.com 1
3  >>> USR <TrID> alice@hotmail.com <auth_string>
4  <<< USR <TrID> OK alice@hotmail.com Alice93
5  >>> CAL <TrID> bob@hotmail.com
6  <<< CAL <TrID> RINGING <session_id>
7  <<< JOI bob@hotmail.com Bob83 <client_capabilities>
8  <<< MSG bob@hotmail.com Bob83 <payload_length>
9         MIME-Version: 1.0
10        Content-Type: text/x-clientcaps
11        Client-Name: aMSN 0.98.3
12        Chat-Logging: Y
```

Si analizzi ciò che succede per ogni comando inviato:

1. il comando **XFR** chiede al NS di fornire l'indirizzo di uno Switchboard Server libero per potersi connettere;

2. il NS risponde fornendo indirizzo IP e porta di uno SB libero; viene inoltre data una stringa di autenticazione da utilizzare al momento della connessione con lo SB, in modo da autenticarsi correttamente; da ultimo, i tre token finali presenti dopo i parametri del comando non sono mai stati visti cambiare (né da chi scrive né dalle fonti consultate per la stesura del presente elaborato), ergo una tassonomia rigorosa non dovrebbe considerarli veri e propri parametri[6];

[5] Dalla locuzione inglese "gracefully leave the session".

[6] Secondo alcune fonti questi sarebbero invece parametri da utilizzare in caso si desideri instaurare una connessione con un contatto di una rete non Microsoft, vedi anche la parte sugli sviluppi futuri.

3. una volta connessi allo SB il primo comando da inviare è un **USR** seguito al solito dal TrID, che ora può essere impostato ad 1, in quanto primo della sessione corrente; gli altri parametri da indicare nel comando sono la mail dell'utente e la stringa di autenticazione precedentemente fornita dal NS;

4. se le informazioni vengono riconosciute come corrette dal SB, questo ci risponde con un USR di conferma, in cui si comunica anche il nickname associato all'account alice@hotmail.com;

5. da qui in poi il client è libero di inviare richieste di sessioni di chat a tutti i suoi contatti, tenendo a mente che tutti quelli che saranno invitati a questa sessione (ovvero mediante questo SB) saranno inseriti nella stessa "chat room" (nei client solitamente ques'opzione è indicata come "aggiungi un amico/contatto a questa chat") e si vedranno tutti reciprocamente[7]; l'invito avviene col semplice comando **CAL** che ha come unico parametro, oltre al TrID, l'account dell'utente che si desidera contattare. Questo è uno dei comandi che più facilmente generano una condizione d'errore sullo SB, alcuni dei quali sono i seguenti:

   - `208 <TrID>` : account name del contatto non valido;
   - `215 <TrID>` : il contatto è già connesso o c'è già una richiesta di connessione pendente;
   - `216 <TrID>` : se ad esempio Bob sta bloccando Alice, oppure se Alice non è nella Allowed List di Bob;
   - `217 <TrID>` : contatto non esistente oppure offline;
   - `713 <TrID>` : (ad apparizione sporadica, documentato dalla community) ricevuto in caso di chiamate ripetute ad un contatto che sta bloccando le richieste mediante apposita lista;

6. il server risponde con lo stesso comando, includendo il token **RINGRING** il quale indica che la richiesta è stata inoltrata al contatto (Bob); viene inoltre indicato anche il numero identificativo della sessione;

7. non appena il contatto accetta la richiesta di sessione di chat, il server ne invia notifica mediante comando **JOI**, trasmettendo con esso la conferma del nome account, il nickname ad esso associato ed un numero che indica le varie caratteristiche di protocollo supportate dal client che sta usando il contatto (l'utilizzo delle "client capabilities" sarà di grande importanza nello sviluppo del trasferimento file, si rimanda pertanto una sua trattazione ai capitoli seguenti); si noti la mancanza di TrID tra i parametri del comando, a testimonianza del fatto che la fase sincrona iniziale della sessione è terminata;

8. lo scambio di comandi si chiude con un messaggio di presentazione da parte del client del contatto, in questo caso di Bob: viene comunicato il nome del client, opzionalmente la versione, e si informa del fatto che si sta operando una registrazione della chat. Lo stesso messaggio va inviato al contatto con i dati del client che si sta utilizzando.

Tutto è compiuto: dopo quest'ultimo comando i contatti possono iniziare a scambiarsi messaggi, inviarsi files, etc...

7.1.2 Ricezione inviti ad una sessione di chat

Come nel caso precedente, si veda prima la successione dei comandi e se ne analizzi poi il significato. Lo scenario questa volta prevede che Alice riceva una richiesta di chat da parte di Bob.

[7] L'aggiunta di contatti alla conversazione non richiede che un invitato sia presente nell'Address Book di tutti i partecipanti, pertanto può capitare di trovarsi a chattare con persone anche sconosciute, in casi particolari. Vedi anche la parte introduttiva al protocollo MSN Messenger.

```
1 <<< RNG <session_id> <SB_ip_address:SB_port> CKI <ticket>
      bob@hotmail.com Bob83 U messenger.msn.com 1
2 >>> ANS <TrID> alice@hotmail.com <ticket> <session_id>
3 <<< IRO <TrID> <index> <roster_count> bob@hotmail.com Bob83 <
      client_capabilities>
4 <<< ANS <TrID> OK
5 >>> MSG <TrID> <ack_type> <payload_length>
6       ...
```

1. il NS invia al client un comando **RNG** quando vi è una richiesta da parte di uno SB ad unirsi ad una chat. Tale comando contiene come parametri l'identificativo della sessione, l'indirizzo IP e la porta del SB a cui effettuare la connessione, un ticket per autenticarsi presso lo SB ed infine nome account e nickname del contatto il quale ha effettuato la richiesta; come per la sintassi del comando XFR inviato dal client (vedi sezione precedente), anche questo comando contiene 3 token non variabili dopo i parametri veri e propri;

2. dopo aver effettuato la connessione allo SB secondo i dati forniti dal comando precedente, il client invia il comando **ANS** con TrID che può essere posto ad 1 ed avente come ulteriori parametri il nome account dell'utente, il ticket ed il numero di sessione come da RNG ricevuto;

3. lo SB invia un comando **IRO** per ogni partecipante già presente nella sessione di chat, indicando per ognuno un indice incrementale univoco, il conteggio totale di partecipanti, il nome account ed il nickname di un contatto ed il relativo indicatore "client capabilities"; si è notato che il nickname qui fornito non è sempre coerente con quello conosciuto dal NS, tuttavia si crede che l'invio del nickname in questa sede serva a dare un nome anche ai contatti non presenti nell'Address Book dell'utente;

4. alla fine della presentazione dei contatti, il server invia il comando **ANS** di risposta;

5. la procedura si conclude con lo scambio dei comandi **MSG** di presentazione tra i due client.

7.1.3 Come si svolge una sessione di chat

Eseguite le operazioni preliminari, si è ora nelle condizioni di interagire con i partecipanti alla sessione di chat. Lo strumento principe che ne permette la realizzazione è il comando **MSG**[8].

MSG è un comando asincrono con payload, avente due sintassi separate tra invio e ricezione. In invio si utilizza la forma

```
>>> MSG <TrID> <ack_type> <payload_length>\r\n
```

Come per molti dei comandi già visti, è necessario indicare l'identificativo di trasmissione (TrID) come primo parametro; i rimanenti due sono rispettivamente il tipo di acknowledgement che si richiede al server (U = unack, non si riceve nessun ack; N = nack, si ricevono solo segnalazioni su mancato recapito del messaggio, A = ack, si riceve sempre conferma di invio; D = codice di ack speciale per messaggi p2p come si vedrà più avanti) e lunghezza in byte del payload che segue. In ricezione il comando ha la forma

```
<<< MSG <account_name> <nickname> <payload_length>\r\n
```

[8] Esistono, a partire dalla versione 14 in modo sperimentale, due famiglie di comandi contraddistinte dalla notazione UU* ed UB* deputate alla comunicazione tra diversi client su diverse reti (l'unica documentazione consultata al momento in cui si scrive parla della possibilità di interfacciarsi con la rete Yahoo! Messenger). L'implementazione di questa funzionalità è stata considerata non prioritaria.

Secondo quanto già visto, il server non comunica il TrID di comandi asincroni (a meno che questi siano una risposta a comandi già inviati o ne richiedano una), pertanto i parametri si limitano ad essere il nome account ed il nickname del contatto che invia il messaggio, oltre alla lunghezza del payload.

Un comando MSG può avere una lunghezza massima pari a 1664 byte, se si tenta di inviare un comando che eccede questa dimensione lo SB chiude la connessione senza recapitarlo.

Il payload per tutti i comandi MSG è formato da un'intestazione e da un corpo messaggio:

```
<header1>: <param1>\r\n
<header2>: <param2>\r\n
...\r\n
\r\n
<body>
```

Si noti la mancanza dei caratteri speciali di nuova linea al termine del payload: la lettura corretta del messaggio in arrivo viene affidata al conteggio dei byte come dichiarati nell'apposito parametro del comando.

Tra gli header ve ne sono due la cui presenza è sempre richiesta:

```
MIME-Version: 1.0\r\n
Content-Type: <content_type>\r\n
```

dove <content_type> indica qual'è lo scopo del messaggio, distinguendolo tra 12 tipi diversi, di cui 1 per la trasmissione di dati personali alla fine della procedura di login, 4 per le notifiche mail (se l'account dell'utente è anche iscritto ad Hotmail), 3 per i contenuti speciali quali emoticon e personalizzazione interfaccia client (ci si ricordi che per la rete Messenger tutti i client connessi sono prodotti da Microsoft), 2 per la gestione di sessioni di connessione fuori dallo SB (tra cui, come si vedrà, i comandi che serviranno a trasportare i dati per le sessioni di file transfer), ed infine 2 per la gestione della sessione di chat.

Typing message Un primo tipo di messaggio viene identificato inserendo la riga `Content-Type: text/x-msmsgscontrol` nell'intestazione. Viene inviato per notificare un partecipante alla chat che uno degli altri contatti sta scrivendo un messaggio. Non viene inviato alcun messaggio che attesti la fine della digitazione, pertanto è compito del client ricevente dare il giusto significato a questo messaggio. Una scelta tipica, realizzata anche nel client Pari-MSN, è quella di presentare in gui un messaggio del tipo "l'utente sta digitando un messaggio" che duri per 6-7 secondi, sino cioé all'invio dell'eventuale successivo messaggio di questo tipo.

Chat message Un messaggio di chat vero e proprio viene inoltrato mediante questo comando; di norma l'header completo ha questa forma:

```
MIME-Version: 1.0
Content-Type: text/plain; charset=UTF-8
User-Agent: <client-name>
X-MMS-IM-Format: FN=<font>; EF=<{B, I, U, S}>; CO=<font_color>; PF=0; RL=0
```

La seconda riga identifica il messaggio come testo semplice codificato in UTF-8, segue indicazione su nome e versione del client che ha inviato tale messaggio ed un ulteriore header che fornisce dettagli sulla formattazione che dovrà assumere il testo contenuto nel corpo del messaggio. I campi di quest'ultimo header indicano il font (FN), uno stile di font (EF, che può essere uno o più tra grassetto, corsivo, sottolineato o barrato, combinando in ordine le 4 opzioni indicate), il colore della scrittura (CO, in formato esadecimale BBGGRR) più ulteriori due parametri il cui significato rimane ad oggi sconosciuto.

A seguire, dopo una linea vuota che come sempre separa l'intestazione dal corpo, viene finalmente incorporato il messaggio digitato dall'utente e destinato ad essere visualizzato presso i client degli utenti partecipanti alla chat.

7.1.4 Chiusura sessione

Per la corretta terminazione di una sessione di chat non sono richiesti invii di comandi particolari, è sufficiente chiudere la connessione con lo SB.

Alla disconnessione di uno dei partecipanti alla chat, lo SB invia agli altri utenti il comando

```
BYE <account_name>\r\n
```

Nel caso invece ci sia un timeout nella conversazione, al client viene inviato il comando

```
BYE <account_name> 1\r\n
```

a cui segue la disconnessione.

7.2 Pari-IM: interventi sul codice

7.2.1 MsnGuiInterpreter

La classe MsnGuiInterpreter ha il ruolo di gestire le operazioni innescate da una richiesta da parte dell'utente veicolata tramite l'interfaccia grafica. Dette operazioni possono variare dalla modifica di parametri interni al client (ad esempio consultare una lista di contatti) all'invio di comandi ai server.

Il lavoro su questa classe è consistito nel controllo della correttezza dei comandi inviati, secondo le modifiche introdotte nel cambio di versione di protocollo. Oltre a questo, vengono presentate alcune migliorìe apportate in corso d'opera.

Rimozione di azioni interfaccia attive In un'ottica di razionalizzazione dei comportamenti delle varie classi componenti il plugin, sono state isolate e rimosse tutte le chiamate dirette all'interfaccia grafica nell'ambito delle sessioni di chat. In particolare sono stati eliminati i comandi per disegnare, aggiornare ed eliminare le finestre di chat, in quanto si è ritenuto opportuno far eseguire le stesse dalla classe MsnMessageActuator in risposta agli appositi comandi inviati dallo SB.

Modifica invio messaggi È stato rivisto il meccanismo di inoltro dei messaggi per due motivi.

Il primo e più banale consiste nella mancanza di un controllo della lunghezza dei messaggi all'interno del vecchio metodo. La rete MSN permette infatti l'invio di comandi MSG con payload massimo di 1664byte, il superamento di questa soglia comporta l'immediata perdita di connessione con lo SB ed il mancato recapito del messaggio. Per tale motivo la procedura è stata modificata aggiungendo un controllo sul numero di byte componenti il payload; in caso di dimensioni eccessive detto payload è suddiviso in più parti ed inviato mediante più comandi MSG in modo ricorsivo.

```
1 byte[] text = null; // the message that can be sent now
2 byte[] remainingText=null; // the part that need to be sent later
3 try {
4    // convert the string in its byte representation
5    text=usertyped.getBytes("UTF-8");
6 } catch (UnsupportedEncodingException e1) {
7    e1.printStackTrace();
```

```
8  }
9  if (text.length>MESSAGE_MAXIMUM_SIZE) {
10    byte[] tmp = new byte[MESSAGE_MAXIMUM_SIZE];
11    for (int i=0; i<MESSAGE_MAXIMUM_SIZE; i++)
12       tmp[i]=text[i];
13
14    // creates the rest of the message, regardless of its size
15    remainingText = new byte[text.length-MESSAGE_MAXIMUM_SIZE];
16    for (int i=MESSAGE_MAXIMUM_SIZE; i<text.length; i++)
17       remainingText[i-MESSAGE_MAXIMUM_SIZE]=text[i];
18    usertyped = new String(tmp); // reconvert the message in a String
19 }
20 ...
21 // sends the first part of the text inserted by user...
22 ...
23
24 // in the end check if something is still to be sent...
25 if(remainingText!=null)
26    sendMessage(new String(remainingText),senderEmail,ricEmail,prot);
```

Il secondo motivo è parzialmente correlato al punto antecedente. Come si è scritto, il conteggio della lunghezza di un payload avviene secondo byte. Ciò di per sé sarebbe un fatto non degno di menzione, non fosse che il protocollo di scambio dei messaggi prevede di utilizzare la codifica **UTF-8**: in questo formato i caratteri UNICODE sono identificati mediante sequenze di byte di lunghezza variabile da 1 a 4, ed il suo sottoinsieme ASCII è interamente codificato con i simboli da un byte - [Yergeau(2003)]. Questo comportamento della codifica ha portato a pensare che la lunghezza di una data stringa fosse uguale alla lunghezza della sua rappresentazione in byte, quando in realtà tale affermazione non è vera in generale. Pertanto, dopo una serie di errori di invio e di ricezione di messaggi contenenti lettere accentate (caratteri UNICODE ma non ASCII, quindi codificate con più di un byte), tutte le chiamate di tipo `String.length()` sono state sostituite con `String.getBytes("UTF-8").length`, che restituisce la dimensione corretta del payload secondo la codifica adottata.

Lo stesso intervento correttivo è stato apportato alle classi ImListener e MsnMessageCreator, in quanto per entrambe si esegue il conteggio dei byte di payload per la corretta gestione dei comandi che ne sono dotati.

Rimozione di metodi obsoleti Sono stati infine tolti i metodi di gestione per il trasferimento file, implementati per MSNP 8 e non più gestiti dai server MSN. Per il trasferimento file aggiornato si rimanda alla parte apposita.

7.2.2 MsnMessageActuator

La classe MsnMessageActuator ha il compito di ricevere in input un comando inviato da un server ed eseguire le azioni che questo richiede. In questo caso si è dovuto prestare particolare attenzione ad alcuni comandi la cui struttura è cambiata, in alcuni anche solo per l'aggiunta di nuovi parametri (ad esempio per i comandi XFR e RNG). Con l'occasione si è provveduto a snellire la classe togliendo dei comandi obsoleti come LST, ADC e REA.

Connessione con gli SB e finestre di chat All'interno della classe è stato svolto un lavoro di razionalizzazione similare a quanto fatto per MsnGuiInterpreter. In particolare sono state corrette le chiamate alla gui per la gestione delle window chat: per ogni utente (visto come istanza della classe MsnUser) sono stati aggiunti due flag ritornati dai metodi `isReadyToChat()` e `hasGUI()` in modo da razionalizzare le chiamate e non aprire finestre doppie: prima di questo

intervento ogni finestra di chat era identificata da un id (l'identificativo di connessione fornito dai comandi ANS e CAL) e nessun controllo era fatto al momento di avviare una connessione con un nuovo SB; il risultato normalmente era la possibilità di aprire due finestre di chat con lo stesso contatto su due SB diversi.

Con i due nuovi flag impostati a livello utente, è ora facile controllare la presenza di una finestra già aperta e bloccare all'inizio un nuovo tentativo di connessione ridondante. È inoltre possibile ripristinare una sessione di chat su di una medesima finestra in caso di caduta di connessione da parte di uno dei partecipanti.

Notifiche Un ulteriore miglioramento che discende direttamente da quanto visto poc'anzi è stata l'introduzione delle notifiche sulla finestra di chat. Si è fatto in modo che l'utente venga a conoscenza di eventi quali:

- contatto non più presente in chat, a seguito di disconnessione o timeout della sessione, sempre notificati dal comando BYE;
- contatto andato offline, segnalato dal comando OUT;
- contatto offline al momento di instaurare una sessione di chat, come da notifica d'errore 217;
- messaggio non recapitato, come indicato dal messaggio NAK.

Parte III. Il file transfer

8 Introduzione al trasferimento file su MSN

Una delle opzioni ma più utili (anche se non strettamente indispensabile) per gli utenti di un servizio di chat è la possibilità di scambiare file con i contatti con cui ci si sta intrattenendo.

Il servizio MSN non solo implementa questa caratteristica, ma nel susseguirsi delle sue varie versioni, il concetto di "trasferimento file" è andato via via a confluire nel più generico concetto di "trasferimento contenuti": ad esso viene dedicato un protocollo indipendente chiamato **Mobile Status Notification Client** (**MSNC**). Ad MSNP 15 viene associato per il trasferimento di contenuti la versione 1 di MSNC.

In questo contesto si parla di trasferimento di contenuti poiché il meccanismo di comunicazione tra i client è progettato in maniera da supportare l'invio di oggetti (chiamati per l'appunto MSNObjects) quali immagini del profilo, suoni, emoticon, ed una serie di altre entità create ah hoc per la personalizzazione del client.

Essendo questo protocollo di comunicazione del tutto generale ed indipendente dal contenuto che si va trasmettere, l'implementazione del trasferimento file può essere vista come un primo obiettivo raggiunto ma soprattutto come testa di ponte per la realizzazione di un lavoro a più ampio respiro nell'ottica di implementare numerose altre funzionalità del client ufficiale.

8.1 MSNC 1

La prima grande novità di questo protocollo è l'introduzione del concetto di "**transport**": questo altro non è che un'astrazione del mezzo comunicativo utilizzato per il recapito di dati: la connessione ad uno Switchboard Server non viene più considerata come l'unica possibile per mettersi in contatto con un altro utente. MSNC 1 permette in questo modo ai client la possibilità di comunicare anche in modalità p2p[9].

Come si vede dalla figura 9.1, un messaggio p2p consiste in un header binario da 48 byte, 0 o più byte di dati ed un footer binario da 4 byte.

8.1.1 Client Capabilities

Prima di passare all'analisi della struttura del messaggio p2p, è necessario chiarire come i client gestiscano la loro capacità di accettare e gestire tali modalità di comunicazione. Parte integrante del protocollo MSNC 1 è rappresentata dal concetto di client capability: ogni client al momento di dichiarare la sua presenza online mediante il comando CHG, invia come parametro un numero che rappresenta una codifica di tutte le funzionalità supportate.

Questo numero è calcolato come somma di diversi valori, ciascuno dei quali rappresenta una singola feature del client. Alcuni esempi sono:

- 0x4000(CapabilitySupportsDirectIM): supporto per 'DirectIM' ovvero possibilità di creare connessioni dirette tra client per effettuare conversazioni senza passare per gli SB;
- 0x100000(CapabilitySupportsSipInvite): supporta gli inviti SIP;
- 0x2000000(CapabilityP2PSupportsTurn): supporta il P2P TURN (Traversal Using Relay NAT);
- 0x10000000(CapabilityMsgrVersion1): supporta il protocollo MSNC 1 (a partire da MSN Messenger 6.0).

[9] Forzandone la definizione, si può in questo modo considerare gli SB come "bootstrap nodes" della rete p2p sottesa.

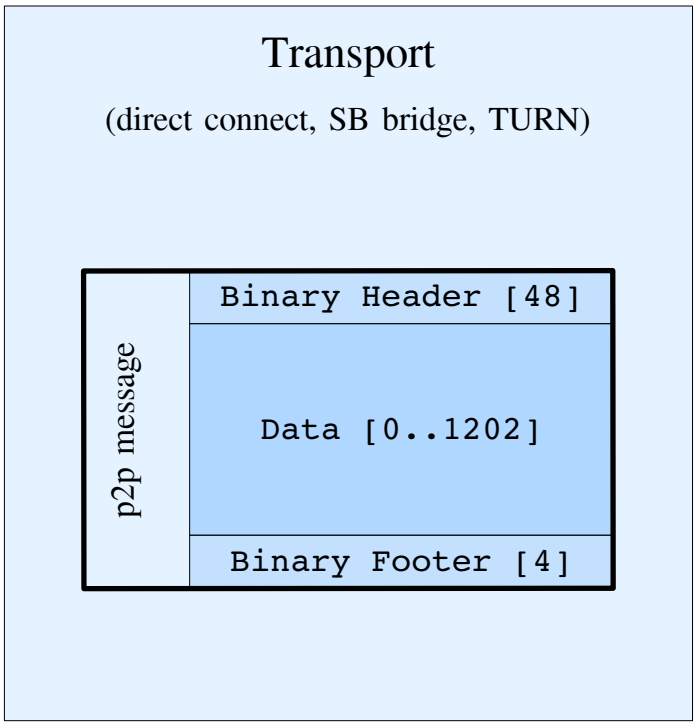

Fig. 8.1: Schema transport

Per una lista più esaustiva consultare [MSNFanatic(2009)].

I singoli valori sono pensati in modo da essere linearmente indipendenti tra loro[10]; in questo modo quando un client riceve le client capabilities di un altro (mediante i comandi JOI, v.p. 30 o IRO, v.p. 31) le può interrogare mediante semplice AND binario con il numero della capability che gli interessa verificare[11].

Una volta che un client ha stabilito quale tipo di comunicazione il suo interlocutore può o non può instaurare, si può avviare la negoziazione per una sessione.

8.1.2 Transport: SB bridge

Il transport mediante Switchboard server è la prima e più naturale maniera di realizzare la comunicazione di dati tra due client connessi alla rete MSN. Questo transport consiste nell'utilizzare il comando MSG come veicolo per i dati, utilizzando convenientemente la sua natura di comando con payload.

Un comando MSG che trasporta dati si distingue da uno contenente un messaggio di chat o di controllo sulla sessione per queste peculiarità:

- tipo di acknowledgement impostato su "D";
- nell'header del payload viene indicato `Content-Type: application/x-msnmsgrp2p`;

[10] Con l'eccezione di quelli eccedenti il 0x10000000 per le versioni di MSNC supportate. In questo caso si considera solo il valore più alto, con questo dando per scontato che una versione di protocollo superiore implementi tutte le versioni inferiori.

[11] Oppure mediante divisione per 0x10000000 per conoscere il numero di protocollo MSNC supportato.

- sempre nell'header viene aggiunta la riga P2P-Dest: `<contact_email>`.

Questo tipo di messaggio permette l'invio di pacchetti di dati di dimensione massima pari a 1202 bytes (il messaggio, comprensivo di intestazione, binary header e binary footer può arrivare sino a 1343 bytes). Ogni messaggio inviato tramite SB bridge viene analizzato dal server, se non vi sono errori di trasmissione, di sintassi, o di congruenza nei dati all'interno del messaggio p2p, viene mandato un comando ACK.

8.1.3 Il Binary Header

A separare il trasporto dall'effettivo contenuto del pacchetto di dati vi è il binary header: 48 byte in cui sono racchiuse le informazioni relative al pacchetto.

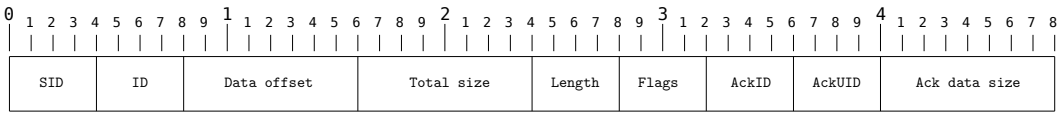

Fig. 8.2: Contenuto del binary header

I 48 byte si dividono in 6 DWORD e 3 QWORD, in formato little endian:

1. **SID** - session identifier: vale 0 durante la negoziazione del trasferimento, altrimenti è un numero casuale generato dal client da cui la comunicazione ha origine; una volta stabilito rimane univoco per tutto il trasferimento.
2. **ID** - identifier, per distinguerlo dal SID è anche detto "base identifier": è un numero casuale generato dal client che inizia la trasmissione; per ogni messaggio di negoziazione della connessione successivo al primo, il base identifier viene incrementato di una unità, mentre rimane lo stesso durante la trasmissione vera e propria.
3. **Data offset**: utilizzato quando un messaggio p2p eccede la dimensione massima consentita dal tipo di trasporto; in questo caso questo è diviso in più messaggi p2p completi di header e footer binari, per ognuno di questi il campo data offset indica quanti byte sono stati trasmessi prima del pacchetto che si sta per andare a leggere.
4. **Total size**: la lunghezza totale in byte del messaggio inviato, comprensivo di tutte le sue parti nel caso sia diviso in più pacchetti; nel caso si stia inviando un ack, questo campo contiene la dimensione del messaggio che si va ad ackare.
5. **Flags**: questo campo può contenere i seguenti valori a seconda del tipo di messaggio che si sta inviando:
   - 0x0: nessun flag specificato
   - 0x1: parte di messaggio fuori ordine
   - 0x2: acknowledgement
   - 0x4: esiste un invito in corso
   - 0x8: errore a livello binario
   - 0x20: immagine personale o emoticon personalizzata
   - 0x100: messaggio di handshake per connessione diretta

- 0x01000030: pacchetto dati per trasferimento file

6. **AckID** - acknowledged identifier: se si sta inviando un ack, questo campo deve essere uguale al campo SID del il messaggio di cui si invia l'ack, altrimenti va inserito un numero casuale.

7. **AckUID** - acknowledged unique identifier: se si sta inviando un ack, questo campo contiene una copia del campo AckID del messaggio di cui si invia l'ack, altrimenti vale 0.

8. **Ack data size**: in caso si invii un ack, questo campo contiene il Total data size del messaggio di cui si invia l'ack, altrimenti è 0.

Messaggi di acknowledgement Come si è visto, molti campi prevedono un un uso particolare in caso si debba inviare un acknowledgement, addirittura alcuni esistono esclusivamente per questo scopo.

Questo aspetto del protocollo non è in contraddizione con quanto visto nella parte riguardante lo SB bridge: in quel caso lo Switchboard server invia un ack per un comando MSG, qui invece sono direttamente i client che si inviano le conferme per l'avvenuta ricezione di un messaggio p2p. In questo modo il trasporto viene ignorato, e si dispone di uno strumento per il controllo della trasmissione anche in caso di connessioni direte tra client.

Le regole per la composizione di un messaggio di acknowledgement sono in buona sostanza quelle viste durante l'analisi dei campi del binary header: il corpo del messaggio è composto da 0 byte ma deve essere chiuso dal binary footer ed incapsulato nel trasporto.

Per quanto riguarda i messaggi splittati, va notato che il protocollo considera i vari chunk come una unica entità messaggio, di cui pertanto va inviato l'ack solo alla ricezione completa.

8.1.4 Contenuto del messaggio p2p

La lunghezza del messaggio p2p vero e proprio varia in base al trasporto mediante cui avviene la comunicazione: si tratta come già visto di 1202 byte nel caso di SB bridge, mentre nel caso di connessioni dirette o mediante NAT la sua dimensione può arrivare a 1352 byte. È contemplato il caso degenere di contenuto pari a 0 byte, corrispondente all'invio di un messaggio di acknowledgement.

Il contenuto non nullo di un messaggio p2p si divide grossolanamente in messaggio di trasmissione dati e in messaggi di invito o handshake, in cui i client tentano di instaurare una comunicazione decidendo il tipo di transport ed i vari altri parametri.

Mentre per quanto riguarda la trasmissione di dati non è necessario scendere più in profondità nell'esposizione, sarà necessario presentare le modalità di instaurazione e di negoziazione della comunicazione tra i client. A questo proposito è stato creato un protocollo ad hoc chiamato **MSNSLP**, che verrà trattato nel dettaglio più avanti.

8.1.5 Footer binario

Il footer binario è una serie di 4 byte che chiudono il messaggio p2p. Di norma essi sono tutti posti a null (ovvero '\0\0\0\0'), tranne che nei seguenti casi:

- durante l'handshake per la connessione diretta questo campo viene omesso;

- vale 1 in formato big endian (ovvero '0x00, 0x00, 0x00, 0x01') quando si stano inviando i pacchetti contenenti l'immagine personale di un contatto.

8.2 MSNSLP

MSNSLP[12] è un protocollo ideato per la creazione, modifica e terminazione di sessioni di comunicazione tra diversi utenti. Discende in maniera diretta dal più generico Session Initiation Protocol (SIP) - [Handley et al.(1999)Handley, Schulzrinne, Schooler, and Rosenberg], da cui eredita solamente un sottoinsieme dei metodi di richiesta.

8.2.1 Struttura richieste MSNSLP

I messaggi MSNSLP assumono questa conformazione:

```
linea iniziale\r\n
intestazione_messaggio_1: valore_1\r\n
intestazione_messaggio_2: valore_2\r\n
...
intestazione_messaggio_n: valore_n\r\n
\r\n
corpo messaggio, composto da 0 o più bytes
NULL ('\0')
```

Linea iniziale Se si sta inviando una richiesta, la prima linea in un messaggio MSNSLP ne indica il metodo, secondo la sintassi:

```
<richiesta> MSNMSGR:<mail_contatto> <versione_protocollo>\r\n
```

La richiesta consiste in uno tra quelle contemplate: INVITE per iniziare una sessione e BYE per chiuderla. Il metodo INVITE può venire impiegato anche nel caso in cui si desideri cambiare parametri ad una sessione già iniziata. Deve essere poi indicata la mail del contatto con cui si andrà ad instaurare la sessione. A chiudere la riga, la versione del protocollo utilizzata, la quale è sempre MSNSLP/1.0, anche nei client più recenti.

Nel caso si invii la risposta ad una richiesta, la riga diviene:

```
<versione> <codice_di_stato> <frase_esplicativa>\r\n
```

Come nel precedente caso si dichiara la versione, a cui segue il codice di stato relativo alla risposta, rappresentato da un numero a 3 cifre, a cui infine segue una breve descrizione testuale. Il codice di stato rappresenta l'esito del tentativo di capire e soddisfare una richiesta ricevuta. Le quattro risposte tipiche sono:

- `MSNSLP/1.0 200 OK\r\n` : messaggio di conferma;
- `MSNSLP/1.0 404 Not Found\r\n` : indirizzo errato[13];
- `MSNSLP/1.0 500 Internal server error\r\n` : segnala errori di comunicazione;
- `MSNSLP/1.0 603 Decline\r\n` : declina l'invito alla sessione.

[12] Per inciso si tratta dell'unico protocollo il cui acronimo non è stato chiarito da parte di Microsoft.

[13] Per quanto strano possa sembrarne il suo uso in caso di comunicazione p2p, il protocollo prevede l'invio di questo messaggio in risposta ad un invito recante un indirizzo email errato nel campo `'To:'`.

8.2.2 Intestazione del messaggio

Tra le intestazioni disponibili, le seguenti vengono sempre utilizzate nel seguente ordine:

- **To** e **From**: questi campi contengono gli indirizzi email degli utenti che stanno instaurando la sessione; il valore di questi campi ha sempre il formato `<msnmsgr:mail@hotmail.com>`;
- **Via**: indica quale percorso è stato seguito dalla richiesta; assume la forma

  ```
  MSNSLP/1.0/TLP ;branch={BranchUID}
  ```

  dove BranchUID indica l'identificatore unico per il messaggio inviato;
- **CSeq**: command sequence, indica il numero sequenziale della richiesta; quando si riceve un messaggio di tipo INVITE si risponde incrementando il CSeq e così via per i successivi inviti nel caso ce ne siano;
- **Call-ID**: identificatore univoco per la sessione, ogni messaggio che ad essa fa capo deve riportare lo stesso Call-ID riportato nel primo messaggio di invito; questo campo deve assumere la forma di un un GUID - [Wikipedia(2010)];
- **Max-Forwards**: nelle intenzioni originarie degli architetti Microsoft questo campo doveva servire a limitare il numero massimo di forward da parte di *gateway* e *proxy* ai server successivi, nella pratica non è mai stato osservato alcun valore diverso da 0;
- **Content-Type**: specifica il tipo di contenuto del messaggio, per le richieste può essere `application/x-msnmsgr-sessionreqbody` o `application/x-msnmsgr-transreqbody`, per quanto riguarda le relative risposte `application/x-msnmsgr-sessionreqbody` oppure infine `application/x-msnmsgr-transrespbody`.
- **Content-Length**: indica la lunghezza in byte del corpo del messaggio MSNSLP.

8.2.3 Corpo del messaggio

Il corpo del messaggio dipende dal tipo di comunicazione che si effettua; in questa sede verrà esplorato il solo caso di trasferimento file.

Esistono quattro varianti per la forma del corpo di un messaggio MSNSLP, e corrispondono ai momenti chiave per la realizzazione di una connessione: la sua instaurazione, la definizione dei suoi parametri, il trasferimento vero e proprio e la sua chiusura.

Inizio sessione Se si sta inviando un messaggio di invito a sessione, la sua intestazione avrà il campo Content-Type impostato ad `application/x-msnmsgr-sessionreqbody` mentre il suo corpo dovrà contenere i seguenti campi:

- **EUF-GUID**: in generale indica il tipo di contenuto che verrà trasferito durante la sessione; al il trasferimento file è associato il valore {5D3E02AB-6190-11D3-BBBB-00C04F795683};
- **SessionID**: contiene l'identificativo della sessione in corso; il protocollo prevede di indicarlo in questa circostanza poiché nel binary header lo stesso campo ha valore impostato a 0 durante la negoziazione (v.p. 38);
- **AppID**: contiene l'identificatore di applicazione; vale 1 quando si sta inviando un'immagine, 2 quando si sta per procedere con l'invio di un file;
- **Context**: questo campo contiene una struttura dati codificata in Base64; nel caso di trasferimento di immagine la struttura dati è quella di un MSNObject, nel caso invece di trasferimento file vi è contenuta una serie di informazioni sul file stesso.

Context Il campo precipuo per il trasferimento file in questo frangente è il campo Context. Una volta che la stringa di questo campo è stata decodificata dal formato Base64, si ottiene una struttura dati, memorizzati in little-endian, che va interpretata come segue (tra parentesi graffe sono indicati gli indici del primo e dell'ultimo byte del dato):

1. Lunghezza dell'header [0..3]: indica il numero di byte di cui si compone l'intestazione della struttura, comprendendo questi 4 byte; di norma la sua lunghezza è pari a 574 byte, anche se in versioni più recenti del protocollo aumenta a 638 byte. Ciò significa che tutti i dati, se presenti, oltre il 574° o il 638° byte sono da interpretarsi come preview del file;
2. Versione MSNC? [4..7]: questo valore è poco conosciuto al momento attuale, si suppone corrisponda alla versione del protocollo MSNC supportata decrementata di una unità, anche se varie prove con diversi valori non hanno mai arrecato alcun tipo di inconveniente o errore;
3. File size [8..15]: in questa QWORD è inserita la lunghezza del file da inviare;
4. Tipo [16..19]: detto anche flag, contiene un identificatore del tipo di oggetto che si vuole inviare, e può assumere i valori:
   - 0x00 - trasferimento file con preview
   - 0x01 - trasferimento file senza preview
   - 0x04 - Background sharing (personalizzazione client ufficiale)
5. Nome file [20..539]: 520 byte contenenti il nome del file codificato in UTF-16LE (little endian); non è nua coincidenza che il numero massimo di caratteri rappresentabili da questa stringa in questo formato è 256, pari al numero massimo di caratteri utilizzabili per un path in ambiente Windows;
6. Null [540..569]: 30 byte che sono sempre stati osservati impostati a null, utilizzo sconosciuto;
7. Sconosciuto [570..573]: una DWORD il cui valore è sempre posto uguale a 0xFFFFFFFE, utilizzo sconosciuto;
8. Sconosciuto [574..637]: quando presenti, questi 64 byte sono impostati a null, utilizzo sconosciuto;
9. Ove disponibili, tutti i byte che seguono vanno a comporre la preview del file: in fase di preparazione al trasferimento di un generico file, il client Messenger ufficiale ne controlla l'estensione e se risulta essere un file immagine ne viene creata una *thumbnail* che viene inserita nel context, sottoforma di immagine PNG di risoluzione 96x96.

Per ulteriori approfondimenti si rimanda a [Tolsma(2005)].

Inizio sessione di trasferimento Nel paragrafo precedente si è visto uno schema che può essere riutilizzato anche nel caso di trasferimenti di immagini o di generici MSNObject. A seguito del primo messaggio che deve rispettare quelle specifiche, nel caso di trasferimento file si invierà un secondo messaggio di invito specifico[14]. Detto messaggio avrà nell'intestazione il valore di Content-Type posto ad `application/x-msnmsgr-transreqbody` e nel corpo i seguenti campi:

- **Bridges**: in cui viene inserita una lista di trasporti supportata dal client;
- **Conn-Type**: specifica che tipo di connessione ha il client alla rete MSN, e può assumere i valori `Firewall`, `Symmetric-NAT`, `Unknown-Connect` oppure `Direct-Connect`;

[14] Chiaramente se il primo messaggio è andato a buon fine e si è ricevuto un `MSNSLP/1.0 200 OK` in risposta.

Fig. 8.3: Esempio di messaggio p2p di invito a sessione

- **UPnPNat**: assume un valore booleano basato sulla disponibilità del relativo servizio presso la rete da cui il client si connette;
- **ICF**: pure booleano, dice se il client si trova dietro ad un firewall.

Risposta e conferma Quando un client riceve un messaggio di invito come quello sopra descritto, è tenuto a rispondere con un nuovo messaggio di tipo `MSNSLP/1.0 200 OK`[15].

In generale, anche per la risposta trattata in precedenza, un messaggio di questo tipo è un messaggio MSNSLP completo di intestazione e corpo. Nella sua forma più semplice e più utilizzata un messaggio di risposta contiene il solo campo '`SessionID:`' all'interno del suo corpo. Nel caso specifico, la risposta deve avere nell'intestazione il campo "`Content-Type`" posto uguale ad "`application/x-msnmsgr-sessionreqbody`", mentre per il corpo si impone l'inserimento di ulteriori campi:

- **Bridge**: comunica il trasporto scelto tra quelli supportati dall'altro client;
- **Listening**: in caso di connessione diretta su ip pubblico, indica la coppia IP - porta su cui il client si è messo in ascolto per ricevere la connessione;
- **Nonce**: in caso di connessione p2p (diretta o tramite NAT) riporta il nonce con cui l'altro client dovrà identificarsi;

[15] Il trasferimento del file è già stato accettato con il precedente '200 OK', pertanto ora non ha senso inviare ora un '603 Decline'.

- se il client è dietro ad una NAT, i campi "**IPv4Internal-Addrs**" e "**IPv4External-Addrs**" contengono rispettivamente i suoi indirizzi IP locali e pubblici, altrimenti viene utilizzato il solo campo "**IPv4Internal-Addrs**" che indica l'indirizzo IP pubblico;

- con uso simile a quello visto per gli indirizzi IP si impiegano i campi "**IPv4Internal-Port**" e "**IPv4External-Port**" per indicare le porte utilizzate dal client.

9 Implementazione trasferimento file

L'implementazione seguita per il plugin di Paripari prevede al momento attuale il solo trasporto mediante "SB bridge", mentre è già pianificata la possibilità di aggiungere la connessione diretta in tempi brevi, alla quale seguirà la realizzazione del transport attraverso NAT.

Nel seguito di questa sezione verrà presentata l'implementazione di questa *feature* all'interno del client Pari-MSN. Per l'esposizione si è scelto di rimanere aderenti alle fasi del suo sviluppo vero e proprio, per poter rendere al meglio i nessi di causalità tra problematiche affrontate e soluzioni adottate al fine di superarle.

9.1 Progettazione file transfer

Una volta capito il funzionamento dei protocolli impiegati nel trasferimento file, una prima bozza architetturale per la sua implementazione è stata immediata. Si è scelto di creare tre entità distinte e di demadare a ciascuna di queste delle funzionalità specifiche: una per l'organizzazione delle sessioni e delle connessioni, ovverosia un Manager per il p2p, e due entità che gestissero l'una la ricezione di file, l'altra il loro invio.

9.1.1 P2PManager

La classe P2PManager costituisce l'anello di congiunzione tra il client preesistente ed il nuovo modulo di trasferimento file. Il suo scopo primario è quello di gestire le connessioni in entrata e di iniziare le connessioni in uscita. Questo scopo a prima vista banale si estrinseca in vari aspetti che vale la pena di vedere più approfonditamente.

Gestione sessioni In prima istanza si è cercato di capire come far gestire al P2PManager la possibilità irrinunciabile di avere in un dato momento più sessioni attive. Per gestire questo aspetto sono state create due ArrayList, una per le classi di invio, una per le classi di ricezione: in questo modo in ogni momento il manager ha i riferimenti alle istanze di ciascuna classe che effettua comunicazione. È possibile così istanziare un comunicatore alla bisogna e rimuoverlo dalla lista di appartenenza una volta che questi ha terminato il suo compito, esercitando un buon controllo della memoria allocata.

Gestione connessioni Il suo ruolo di isolante tra parte di '*controller*' che va a gestire le logiche del trasferimento file e resto del plugin, permette al P2PManager la possibilità di implementare virtualmente tutti i tipi di trasporto previsti dal protocollo. La connessione tramite SB bridge viene "in omaggio" al fatto di appoggiarsi a Pari-MSN: al plugin basta intercettare i messaggi MSG forgiati in una certa maniera ed indirizzarli al manager. Concettualmente non più difficile sarà l'implementazione di connessione diretta o mediata da NAT, in quanto una volta approntata l'infrastruttura di comunicazione costruita appositamente, basterà farla colloquiare con il manager.

Servizi per i comunicatori Dovendo veicolare tutti i messaggi, si è ritenuto opportuno che il P2PManager svolgesse una serie di operazioni su di essi che sono comuni per entrambi i tipi di sessioni (in entrata ed in uscita). Il primo servizio svolto è quello di suddividere tutti i messaggi nelle loro parti fondamentali. Come si ricorda dalla figura a pag 36, lo schema di un messaggio p2p è fisso nelle sue componenti, ma non nella lunghezza delle stesse. Compito qui svolto è pertanto quello di riconoscere le varie parti costituenti il messaggio e dividerle tra di loro, inserendole in strutture dati specifiche: ArrayList di array di byte per le parti testuali del messaggio, che ne evidenziano la struttura a testo (ogni array di byte ne contiene una riga) e ByteBuffer per il binary header ed il binary footer.

Un altro servizio offerto dal manager è quello di riconoscere e ricostruire i messaggi di negoziazione splittati. Come si è visto, i messaggi MSNSLP eccedenti date lunghezze vengono inviati suddivisi in più messaggi p2p. Si è ritenuto opportuno utilizzare il P2PManager come strumento di caching per gestire queste situazioni: esso ne riceve tutti i chunk, attende che il messaggio sia completo e solamente allora lo invia al comunicatore che lo sta attendendo. Quest'ultimo lo potrà gestire come messaggio intero, non preoccupandosi di gestire gli aspetti riguardanti la sua ricombinazione. Non si è ritenuto opportuno estendere questo comportamento anche alla ricezione di un file (che di fatto è realizzata mediante invio di chunk di un unico messaggio), poiché il ricevitore è in grado di gestire più efficacemente questo scenario: ogni chunk è scritto su disco non appena arriva, senza crearne copie cache in memoria.

9.1.2 P2PSender e P2PReceiver

L'attività di gestione delle sessioni di trasferimento file viene deputata alle classi P2PSender e P2PReceiver. La loro progettazione è stata chiaramente vincolata dalle specifiche del protocollo di trasferimento file, la loro analisi sarà approfondita durante la discussione dell'implementazione.

Una scelta architetturale che qui va discussa è costituita dall'interfacciamento delle classi di comunicazione con il manager: come fa una loro l'istanza ad essere vista ed utilizzata dal manager? Come si instrada un messaggio in arrivo al corretto comunicatore?

Interfacciamento con il P2PManager Si assume che ogni sessione di trasferimento file sia univocamente identificata dal SessionID. Una volta che il P2PManager ha ricevuto un messaggio, questi ne estrae l'identificatore di sessione (dal binary header in caso di messaggio di dati, dal corpo del MSNSLP in caso si stia ancora negoziando la sessione). Ora diventa necessario stabilire a chi va forwardato il messaggio. Il manager risolve questo problema mediante poll diretto su tutti i comunicatori, seguendo questo algoritmo:

```
1  Per ogni Sender
2     Scopri il suo SessionID
3     Se il SessionID del messaggio corrisponde
4        Invia il messaggio
5     Altrimenti se msg_AckID == Sender_SessionID
6        Invia il messaggio
7  Altrimenti per ogni Receiver
8     Scopri il suo SessionID
9     Se il SessionID del messaggio corrisponde
10       Invia il messaggio
11    Altrimenti se msg_AckUID == Sender_Last_AckID
12       Invia il messaggio
```

L'algoritmo testè esposto riesce a gestire tutti i casi possibili di accoppiamento tra messaggio e comnuicatore delegato a gestirlo. Nel dettaglio, si può notare che il primo tentativo di accoppiamento tra i due avviene mediante corrispondenza di SessionID, ove disponibile. Quando il

SessionID non viene riportato nel messaggio (tipicamente negli acknowledgement od in alcuni messaggi di negoziazione), il manager svolge ulteriori controlli al fine di inoltrare correttamente il messaggio: prima verifica la corrispondenza tra AckID del messaggio e SessionID, poi svolge lo stesso tipo di check tra AckUID del presente messaggio e AckID dell'ultimo inviato dal sender (per il significato di ciascun campo si ricordi la trattazione sul binary header, pag. 38).

Nel senso opposto, ovvero in caso di invio di messaggi tramite P2PManager, non è stato necessario alcun particolare accorgimento; sia il receiver che il sender una volta che hanno generato il messaggio lo passano al manager che si incarica di adattarlo al trasporto utilizzato e tramite esso lo instrada.

9.2 Integrazione al plugin MSN

Una volta progettata la sezione preposta ad eseguire il file transfer, si è dovuto pensare a come integrarla nel client esistente. Questo lavoro, all'apparenza non di grande difficoltà, è stato reso non banale da aclune caratteristiche connaturate del plugin IM; alla sua esecuzione ed alle problematiche affrontate si dedica pertanto questa apposita parte. Va inoltre notato che questo è stato il primo passo implementativo nella realizzazione del trasferimento file, pertanto si cominceranno ad esporre anche i primi dettaglio realizzativi.

9.2.1 Ricezione messaggi mediante MsnMessageActuator

Se si pensa al fatto che tutte le sessioni avvengono mediante bootstrap da SB bridge, ed in taluni casi queste possono essere interamente svolte senza mai lasciare questo trasporto, la scelta quasi obbligata è quella di cercare nel client la parte in cui si smistano i comandi MSG e lì inserire un meccanismo che devii verso il P2PManager quei comandi che veicolano messaggi MSNC 1. La classe che fornisce questo particolare servizio è MsnMessageActuator.

L'integrazione dei due plugin MSN ed IM permette di avere un ottimo controllo sul tipo di comandi da trattare, sono state quindi necessarie poche modifiche al codice in tal senso:

1. inserimento del tipo di messaggio `"application/x-msnmsgrp2p"` nelle tabelle di hash nella classe MsnHashTables;

2. nella classe MsnMessageActuator, all'interno del metodo `realizeMSG(MsnMessage mess)`, introduzione di una clausola nello switch per poter gestire il presente caso;

3. nel punto appena creato, inserimento della chiamata a P2PManager con invio del messaggio.

Una volta apportate queste modifiche è stato possibile cominciare la stesura del codice per il P2PManager, nella fattispecie la parte di suddivisione dei messaggi nelle loro parti fondamentali.

Questo punto del lavoro svolto sul plugin è stato uno dei più laboriosi, in quanto il codice, scritto daccapo diverse volte, non riusciva mai a dare i risultati sperati.

9.2.2 String vs. byte[]

Dopo varie analisi del codice e dell'output da questo prodotto, ci si è accorti che nell'header binario si ripeteva diverse volte il valore intero 65533 anche in campi dove il protocollo prevedeva necessariamente valori diversi. Una volta decodificato, se ne otteneva una rappresentazione del tipo riportato in figura 9.1, senza nessun significato apparente.

Effettuando ricerche in merito, si è riscontrato che quel particolare codice ed il relativo simbolo stanno ad indicare un errore da parte del sistema nella lettura dello stream: trattasi nella fattispecie di errore di decodifica, sollevato quando vi è l'impossibilità da parte del parser di abbinare un dato ad un particolare simbolo, come viene chiarito in [Wikipedia(2011b)].

Ancora una volta, ci si è imbattuti in una discrepanza tra dato e sua rappresentazione, ovvero più banalmente tra stringhe ed array di byte. È utile a questo punto notare, quando si presentava lo schema a pag. 10, che il plugin IM fornisce ai suoi "sotto-plugin" un servizio di recapito messaggi tramite inString. È pure utile ricordare che tutti i tipi di trasmissione di comandi tra client e server all'interno della rete MSN avvengono mediante utilizzo della codifica UTF-8.

Fig. 9.1: Carattere sotitutivo: U+FFFD

In buona sostanza, ciò che mediante MsnMessageActuator si inviava al P2PManager era di nessuna utilità, in quanto la presenza di un binary header contenente sequenze qualsiasi di byte poteva venire danneggiata dal parser UTF-8 nel momento in cui il Listener suddivideva il comando MSG in stringe per passarlo alle code circolari del plugin IM.

9.2.3 Adattamento di ImListener

L'unico punto in tutto il plugin in cui lo stream di byte in arrivo dal server MSN era conservato tale senza forzarne una qualche decodifica era all'interno del metodo che esegue la lettura dal socket. Non potendo mettere mano al codice del plugin IM per evitare di forzare la riscrittura di altri sotto-plugin in fase di sviluppo, l'unica alternativa rimasta era di riscrivere la procedura eseguita durante la lettura dello stream.

Lettura dello stream Una volta cambiati i lettori e gli scrittori su socket (da `BufferedReader` e `BufferedWriter` si è passati rispettivamente a `BufferedInputStream` e `BufferedOutputStream`), la prima parte di riscrittura del listener è consistita nella razionalizzazione della fase di lettura degli stream in ingresso, in particolare nel metodo che si fa carico di leggere il payload di un comando. Dal vecchio metodo che estraeva un byte alla volta facendo ad ogni passo un check sui caratteri per determinare un "end of line", si è passati alla scansione dello stream in una unica chiamata, con gestione della concorrenza:

```
1  BufferedInputStream streamBuf ...
2  int rem = ... // read the payload length from the command line
3  byte[] payload = new byte[rem];
4  int read = 0;
5  synchronized(streamBuf) {
6     while (read < rem) { // reads as many bytes as possible within the
7                             message limits
8        try {
9           read +=  streamBuf.read(payload,read,rem - read);
10    } catch (IOException e) { e.printStackTrace(); }
11 }
```

questo codice garantisce diversi vantaggi:

- una lettura veloce dello stream, in modo da liberare il lettore affinché serva altre chiamate;
- preserva l'integrità del flusso di dati all'interno del singolo comando che si va a leggere;
- estrae esattamente il numero di byte dichiarati nel comando, senza danneggiare eventuali altri comandi già accodati nello stream.

Rimane comunque una parte in cui si legge interamente il payload alla ricerca degli EOL, poiché le InString del plugin IM trattano linee singole di payload. Non si è potuto utilizzare un tokenizer poiché l'ultima riga di un payload non è correttamente terminata, e l'inserimento di un carattere spurio al suo interno creerebbe errori nel conteggio delle lunghezze dei payload quando i comandi sono ricostruiti all'interno della classe MsnMessageCreator. Le righe così trovate sono memorizzate in un ArrayList di stringhe, ma l'array di byte precedentemente letto viene comunque tenuto in memoria per il momento.

Gestione dei messaggi in entrata A questo punto è stato inserito un controllo sul tipo di messaggio che si sta inviando al plugin: pur rendendosi conto che si tratta di un errore concettuale demandare al listener un compito - come il parsing dei messaggi - che non gli può spettare, come si è visto non rimane che questa strada da percorrere per avere un trasferimento file funzionante.

In realtà non si tratta di un vero e proprio parsing completo dei messaggi: dopo la lettura dello stream di byte e dopo averlo suddiviso in righe mediante ispezione diretta, il Listener scorre il payload alla ricerca di una stringa identificativa dei messaggi p2p: "`application/x-msnmsgrp2p`". Se questa viene trovata si invia l'array di byte, così com'è stato letto, al P2PManager, altrimenti la serie di stringhe già pronte sono inviate alla InQueue per la loro successiva consegna al client MSN vero e proprio.

Scrittura sullo stream Durante questa fase di lavorazione sul listener, se ne è voluto correggere il comportamento anche in merito all'invio dei messaggi. L'unico metodo esistente per scrivere su socket era infatti il solo `writeOnSocket(String)`, il che si prestava agli stessi problemi già discussi poc'anzi.

Si è quindi pensato di creare un nuovo metodo in sovraccarico `writeOnSocket(byte[])`, che accetta in input un array di byte e lo scrive direttamente su socket, gestendone la concorrenza con il costrutto sintattico **synchronized**, esattamente come nel caso precedente. Il metodo precedente è stato conservato nello scopo ma migliorato nella realizzazione. Va detto infatti che la OutQueue, al pari della InQueue, è progettata per gestire singole stringhe, non messaggi interi; poteva pertanto capitare che due comandi con payload inviati dal client con tempi troppo ravvicinati tra di loro venissero mescolati nella OutQueue, scritti in modo errato sul socket ed interpretati come errore di comunicazione dal server. Risultato: connessione caduta il più delle volte.

Il metodo `writeOnSocket(String)` ha pertanto dovuto subire due revisioni non marginali: da una parte è stato migliorato in modo da tenere in cache le parti di un messaggio non complete che arrivano a blocchi dalla OutQueue, dall'altra è stato fatto in modo che il suo output non sia direzionato al socket ma al metodo `writeOnSocket(byte[])`, in modo da poter gestire correttamente la concorrenza sullo stream di output.

Gestione errori Un'ultima serie di migliorìe apportate al listener consiste nella gestione delle connessioni con gli SB e di alcune situazioni d'errore non previste.

In particolare, si è aggiunto ad ogni metodo incaricato di eseguire letture dal socket (solo per la lettura dei payload ne esistono 3 diversi, per i comandi UBX, MSG e GCF) la possibilità di ritornare un codice d'errore al gestore del thread del listener per effettuarne la terminazione in caso di errore imprevisto e non gestito. Questo permette di fermare l'esecuzione del listener in modo controllato, demandando al resto del plugin la gestione del da farsi, tipicamente notificando all'utente la disconnessione e chiedendogli se di desidera riconnettersi. Come secondo effetto lo stop controllato del listener permette di chiudere correttamente i socket, evitando così l'insorgere di situazioni di blocco degli stessi.

È stato possibile in questo contesto realizzare un controllo su un altro tipo di errore generato dal solo protocollo di file transfer: si è verificato spesse volte durante i primi test che, a seguito di un errore di sintassi in un comando MSG, lo Switchboard Server rispondesse con dei burst di

rumore a 32 bit di discreta lunghezza temporale, i quali portavano rapidamente a saturazione del socket ed al lancio di una "broken pipe exception" di non facile gestione. Per arginare sul nascere questo tipo di risposta degli SB, è stato inserito un controllo preventivo al momento della lettura del primo simbolo da socket:

```
1 char ch = 0;
2 try {
3     ch = (char) streamBuf.read();//Read the first character, blocking...
4     if (ch == (char)Integer.parseInt("ffff", 16 )) {
5     // omg something is wrong!
6     ...
7 }
8 ...
```

9.3 Implementazione

Con un sistema di comunicazione finalmente completo e funzionante, si è potuto procedere con lo sviluppo delle funzionalità di trasferimento file.

9.3.1 Strumenti e metodologie

La scrittura di codice è stata accompagnata in ogni momento dal suo testing immediato sul campo: la documentazione di cui si è potuto disporre consisteva in una mezza dozzina di pagine wiki - [MSNFanatic(2008)], talvolta fatte bersaglio di atti di vandalismo[16], ed il già citato forum [MSNFanatic(2011a)], per cui ogni modifica ed ogni aggiunta, seppur minima, doveva essere provata empiricamente.

Non è stato ritenuto indispensabile ricorrere a strumenti di sniffing dei pacchetti come Wireshark, perché il buon lavoro svolto sul listener ha fatto in modo di avere in ingresso al plugin esattamente gli stessi dati disponibili su cavo.

Di contro, si è ritenuto opportuno testare il client Pari-MSN contro altri client. Per questo tipo di prove sono stati utilizzati:

- **Pidgin**: per il suo debugger molto verboso e dettagliato, utilissimo soprattutto nel raffinare il codice per la creazione e la lettura dei messaggi MSNSLP;
- **aMSN**: per il suo supporto molto rigoroso al protocollo, ha premesso di individuare errori non segnalati da altri client;
- **Emesene**: molto utilizzato presso la comunità di sviluppatori MSN non ufficiali;
- **Windows Live Messenger 2011**: usato soprattutto per collaudare le modifiche apportate alla chat, costituisce ovviamente la prova finale per stabilire il corretto funzionamento di un client[17].

Nello sviluppare la funzionalità di file transfer si è sempre proceduto con lo scrivere prima il codice del P2PSender, testarne la funzionalità contro uno dei client, perfezionarlo sino ad ottenere una risposta coerente con la nostra richiesta. Arrivati a questo punto si passava con l'implementare

[16] Soprattutto nella forma di spam pubblicitario. Si segnala inoltre un caso di disinteresse da parte dei proprietari: in modo peculiarmente bizzarro il sito è andato offline per mancato rinnovo del contratto di hosting nei giorni in cui si iniziava a stendere il presente lavoro. Il fatto che lo stesso sito sia stato riattivato dopo pochi giorni non ha impedito a chi scrive di apprezzare le funzionalità di HTTrack in combinazione con la cache di Google.

[17] Anticipando la chiusura della relazione, si può affermare che tale prova è stata superata anche per il file transfer!

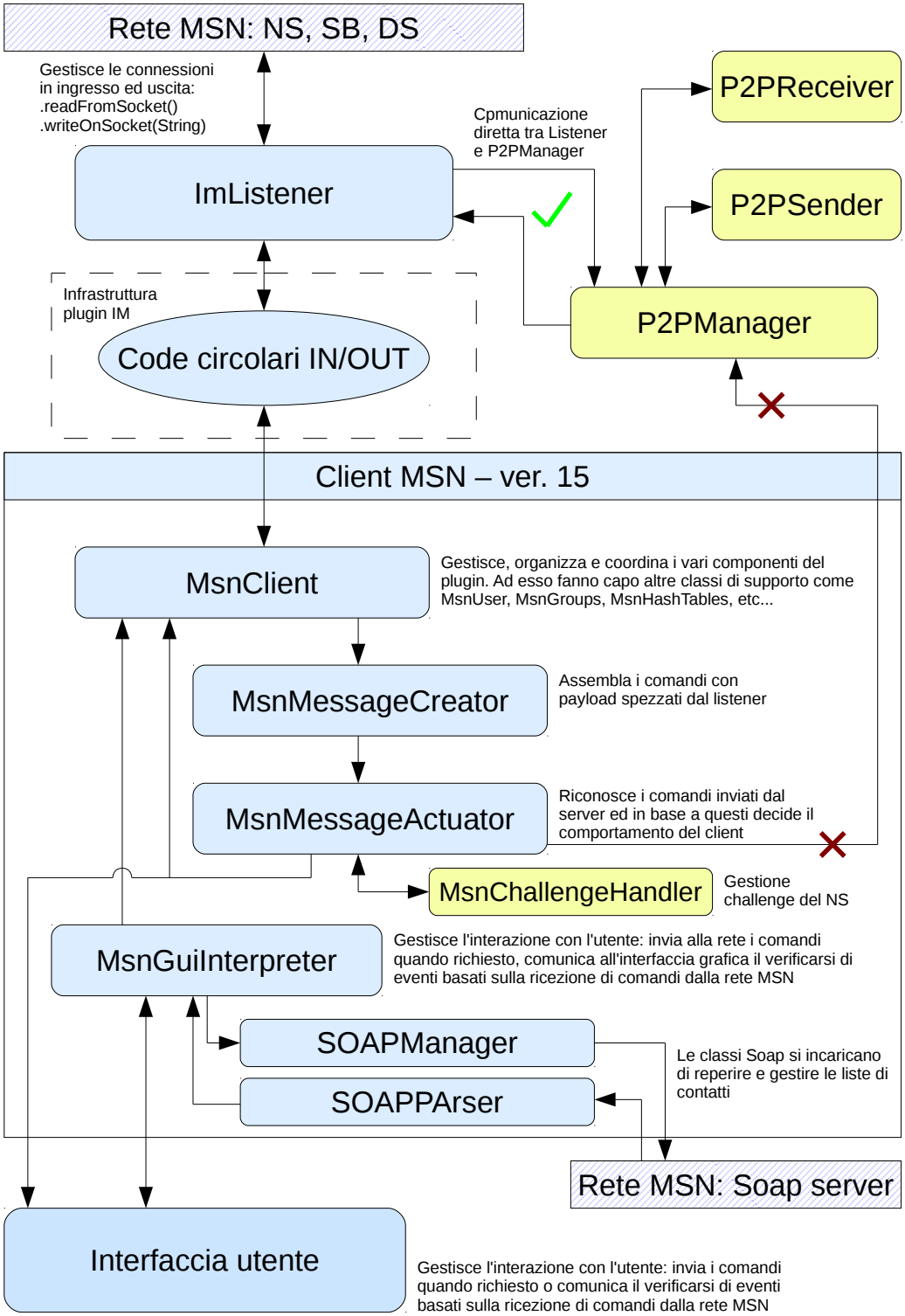

Fig. 9.2: Schema plugin MSN dopo l'inserimento del modulo di trasferimento file

lo stesso meccanismo di risposta nel P2PReceiver; il collaudo di questo codice consisteva quindi nell'inviare la medesima richiesta al nostro client da uno esterno e valutarne la correttezza del comportamento.

9.3.2 P2PTester

A questa prima parte molto dispendiosa in termini di tempo (compilazione, avvio PariPari, IM, MSN, instaurazione chat, prova invio file) si è andati ad integrare un secondo strumento di test, sottoforma di classe creata specificatamente, chiamata P2PTester.

Durante una connessione il P2PManager, il quale veicola tutti e soli i messaggi p2p, salva una copia di questi in un file su disco, in modo da avere una pseudo-sessione avviabile anche offline. Il compito della classe P2PTester è quindi quello di istanziare un P2PManager (operazione non banale in quanto gli si dovrà far credere di avere a disposizione un listener, un client e diverse altri oggetti in realtà non referenziati) leggere da detto file e ricreare una finta sessione di trasferimento file.

Questa pratica è stata di grandissima utilità per lo sviluppo del P2PReceiver perché, già sapendo come questo doveva reagire a determinati input, non era necessario provare diversi scenari: lo si è pertanto preferito testare utilizzando gli stessi comandi di una sessione già svolta a parti inverse senza accedere alla rete MSN, verificando di volta in volta il suo comportamento e perfezionandolo sino a renderlo corretto.

9.3.3 Strutture dati utilizzate

Per la corretta realizzazione del trasferimento file, è stata di fondamentale importanza la scelta delle corrette strutture dati da impiegare di volta in volta.

ByteBuffer Per la corretta gestione di tutte le parti dei messaggi p2p non riducibili a stringhe, si è scelto di wrappare gli array di byte in oggetti di tipo ByeBuffer. Questa scelta ha comportato il vantaggio di avere una gestione automatica dell'**endianness**: i metodi della classe ByteBuffer riescono ad estrarre dati dall'array sotteso rispettandone il formato di memorizzazione. Un esempio di utilizzo di questo oggetto è dato dalla classe Utilities del package: qui è contenuto il metodo `public static ByteBuffer createBinaryHeader(...)` che, ricevuti in input i 9 dati per i campi di un binary header, ne restituisce la loro corretta composizione.

Enum Sono stati definiti diversi tipi di dati `enum` all'interno del pacchetto di file transfer. Per le classi P2PReceiver e P2PSender sono stati definiti enum privati per gestire gli stati di una istanza della classe. Ad esempio per il P2PReceiver

```
1     private enum States {
2         WAITING_FIRST_INVITATION,
3         WAITING_SECOND_INVITATION,
4         HANDSHAKING_DIRECT_CONNECTION,
5         RECEIVING_DATA
6     };
```

Questi vari stati sono consultabili dall'esterno mediante apposito getter. Il P2PManager in questo modo può sempre sapere in che stato si trova uno dei comunicatori, ed in base a questo può decidere quali messaggi forwardargli, oppure addirittura può procedere con la rimozione di una istanza non più attiva.

È stata inoltre creata un'apposita classe di enum per agevolare la gestisce dei flag dei binary header (vedi pag. 38) da parte dell'intero package.

9.3.4 Gli acknowledgement da Switchboard Server

Durante lo svolgimento si una sessione MSNC mediante SB bridge, esistono due tipi di acknowledgement che il client riceve. Il primo è l'ack p2p, e viene gestito dal client sottoforma di messaggio MSNC completo, con opportuni valori nel binary header; il secondo è quello inviato dallo Switchboard server al momento della ricezione e dell'instradamento del comando MSG. A prima vista l'utilizzo del solo primo tipo è sufficiente per gestire una sessione: attendo un messaggio, lo ricompongo nel caso sia inviato mediante più chunk e dopo averlo eventualmente riassemblato invio un messaggio di acknowledgement per informare l'altro client che la ricezione è avvenuta con successo. In quest'ottica la ricezione di comandi di conferma da parte dello SB pare ridondante.

Forti di questa convinzione, si è deciso di implementare la prima versione del modulo di trasferimento file senza considerare questi ultimi ack. L'invio di un file è stato quindi realizzato inserendo un thread nella classe P2PSender che effettuava lettura ed invio dei chunk secondo una temporizzazione stabilita, onde evitare di sovraccaricare il buffer di invio del socket. Nessu tipo di temporizzazione però sembrava funzionare, dopo l'invio dei primi chunk il server rispondeva sempre con il burst di rumore descritto a pag. 48 e la connessione cadeva.

Si è pertanto deciso di provare a temporizzare l'invio dei singoli chunk secondo gli ack dello SB. L'implementazione di questa funzionalità non è stata immediata ed ha comportato l'applicazione delle seguenti modifiche al plugin:

- MsnMessageActuator: alla ricezione di un comando ACK, il metodo `realizeACK(MsnMessage mess)` ne estrae il TrID e lo invia al P2PManager invocando il metodo `notifySender(trid)`;

- P2PManager: il metodo testè citato esegue un poll tra tutti i sender, chiedendo a ciascuno qual'è il TrID dell'ultimo messaggio p2p inviato; se c'è corrispondenza, al relativo P2PSender viene permesso l'invio del prossimo chunk del messaggio, mediante chiamata al metodo `sendData()`;

- P2PSender: dal canto suo ciascun sender deve farsi carico di memorizzare il TrID; si è pertanto aggiunto tale dato come variabile d'istanza che viene aggiornata ad ogni invio di un comando, facendone richiesta al P2PManager.

Questo medoto di temporizzazione ha effettivamente eliminato tutti gli errori generati dallo Switchboard server, ed ha anzi permesso di effettuare delle osservazioni interessanti. Si è notato che gli SB gestiscono il traffico p2p applicando pesanti restrizioni sulla banda utilizzabile per questo scopo: non si è mai riusciti ad instaurare un trasferimento con velocità superiori ai 3 KB/s. Secondariamente è stato notato che l'accettazione di messaggi p2p avviene generalmente in cluster di dimensioni dell'ordine della mezza dozzina per volta con ack quasi immediato; dopo ogni cluster vi è un lag nell'invio dell'ack dell'ultimo messaggio di circa 2 secondi[18]. Si ritiene sia questo comportamento la causa degli errori precedentemente menzionati: un ritardo dell'invio dell'acknowledgement con variazione così alta porta alla necessità di sincronizzare l'invio del singolo messaggio sulla conferma di quello precedente, cosa che permette allo SB di controllare con estrema precisione il numero di pacchetti veicolati ed in ultima istanza di poter limitare la banda concessa al p2p. A supporto di quanto detto, si è verificato con test appositi che è proprio l'invio di un comando MSG avente payload p2p prima dell'ack di quello precedente che genera il comportamento anomalo dello SB.

È lecito supporre che tale tipo di controllo sulla banda intenda scoraggiare l'utilizzo del trasporto tramite SB bridge. Per questo, ora che le funzionalità di file transfer sono state correttamente implementate, sarà necessario sviluppare il supporto ai rimanenti tipi di trasporto.

[18] In prima approssimazione sono 6 messaggi in circa 3 secondi. Se ciascun messaggio veicola 1202 byte sono circa 7KB su 3 secondi, poco più di 2 KB/s di media.

9.3.5 Modifiche da apportare al codice

Durante la trattazione si sono potuti vedere alcuni modi in cui il modulo di trasferimento file è immediatamente migliorabile. Su tutti spicca il tipo di dipendenza che si instaura tra P2PManager e le due classi di comunicazione, P2PSender e P2PReceiver. In questo contesto sono da rivedere due aspetti:

- **SessionID**: identifica univocamente una sessione di comunicazione, pertanto pare corretto memorizzarlo direttamente nell0istanza di un comunicatore mediante variabile d'istanza; epperò si è visto che al momento del forward di un messaggio in arrivo da parte del P2PManager, quest'ultimo svolte un'attività di polling estensivo sui comunicatori alla ricerca del destinatario corretto;
- **TrID**: come nel caso precedente è parso conveniente memorizzare per ogni istanza il relativo TrID dell'ultimo messaggio inviato; in questo caso però ci si accorge che queso vìola i ruoli che in fase di progettazione si sono assegnati alle varie classi poiché non può e non deve essere di pertinenza di un sender sapere quale trasporto si sta usando e come lo si gestisce.

La soluzione che ci si propone di integrare quanto prima nel client consiste nel creare un tipo di dato complesso all'interno del manager, che andrà a contenere i seguenti oggetti:

1. referenza al comunicatore istanziato, sia esso sender o receiver;
2. tipo di trasporto utilizzato;
3. identificativo della sessione in corso;
4. ultimo Trid inviato nel caso in cui si sia utilizzando il trasporto mediante SB bridge.

Un veloce esempio di ciò che si andrà a creare può essere, nel caso dei P2PSender:

```
1  private enum bridgeType {
2     SB_BRIDGE,
3     DIRECT_CONNECT,
4     NAT
5  };
6
7  private class SenderRef {
8     private P2PSender sender;
9     private int sid; // SessionID
10    private bridgeType bridge;
11    private int trid; //
12
13    public SenderRef() {
14       // TODO initialize te object
15    }
16
17    // Getters and setters
18    public P2PSender getSender() { return sender; }
19    public void setSender(P2PSender sender) { this.sender = sender; }
20    public int getSid() { return sid; }
21    public void setSid(int sid) { this.sid = sid; }
22    public bridgeType getBridge() { return bridge; }
23    public void setBridge(bridgeType bridge) { this.bridge = bridge; }
24    public int getTrid() { return trid; }
25    public void setTrid(int trid) { this.trid = trid; }
26 }
```

A questo punto basterà istanziare un ArrayList di questi oggetti anziché di puntatori diretti al comunicatore:

```
1 // OLD VERSION
2 // private ArrayList<P2PSender> senders;
3
4 // NEW!
5 private ArrayList<SenderRef> senders;
```

In questo modo si crea una sorta di "cuscinetto" tra manager e sender, di proprietà e di pertinenza del manager stesso, rispettando quindi le specifiche iniziali ed, in ultima analisi, la coerenza semantica tra i dati e gli scopi specifici di ciascuna classe.

10 Conclusioni

Il lavoro svolto sul client MSN ha portato tutte le sue features implementate a rispettare la versione 15 del protocollo MSNP. Pari-MSN può ora essere considerato un client maturo e funzionale, ma al contempo ricco di potenzialità ed ancora ampiamente estendibile.

10.1 Sviluppi futuri

Tra le migliorìe che sarà possibile portare a termine si citano, in ordine di priorità:

1. **nell'immediato**: gestione del trasporto mediante connessione diretta per le sessioni p2p, possibilità di connessione alla rete MSN mediante tunnel HTTP;
2. **nel medio periodo**: gestione del trasporto p2p attraverso NAT mediante TURN, interfacciamento alla nuova GUI in via di sviluppo, espansione delle potenzialità del protocollo MSNC (gestione immagini personali, MSNObject, etc...);
3. **nel lungo periodo**: accesso alla rete MSN tramite connessione sicura SSL, gestione della messaggistica su network multipli (Facebook, gtalk, etc...).

Saranno poi da valutare ulteriori possibilità di sviluppo: upgrade versione MSNP alla 18 se non addirittura direttamente alla 21 (introdotta con Windows Live Messenger 2011), implementazioni di feature attualmente considerate non necessarie quali ink, gleam, spaces, invio personalizzazioni client tramite p2p e così via. Ci si può addirittura spingere oltre ed immaginare la creazione un insieme di API per la gestione del client: quello ufficiale ne mette a disposizione e grazie ad esse stanno nascendo progetti stimolanti quali creazione di bot, di cui un esempio è un traduttore istantaneo multilingua: [Microsoft(2011)].

10.2 Considerazioni

Quando si svolge un lavoro di analisi di un protocollo non pubblico, e soprattutto quando si ha a che fare con l'accesso a reti non di pubblico dominio, non si può trascurare il punto di vista dell'ente proprietario. La sensazione più diffusa tra chi sviluppa client non ufficiali MSN è ben riassunta dal seguente brano, tratto da `http://www.hypothetic.org/docs/msn/general/overview.php`:

> **What do Microsoft think about all of this?**
> We have no connection with Microsoft, and only very limited communication. We know that at least some Microsoft employees are aware of this site and the community in general, but corporate policy seems to be one of ignoring us. Microsoft haven't

> made any serious attempt to keep third party clients out of their network, but they've also never attempted to talk with us or give assurances about the future. They've been very receptive about reports of bugs that have security implications.
>
> Compared to AOL (which owns both AIM and ICQ), Microsoft is very nice to the third party developers of its protocol. This probably doesn't reflect any great philanthropy on their part: third party clients boost the number of users on Microsoft's network and (unlike with AOL) we don't represent a direct threat to their revenue stream.

Per quanto questo brano provenga da una pagina redatta nel 2003, la situazione pur evolvendosi è di poco cambiata. Microsoft ancora non procede attivamente con la rimozione di client non ufficiali dalla rete MSN, ma i meccanismi di controllo sull'autenticità dei client si sono raffinati pur non diventando insuperabili, come si è visto. Molto probabilmente la loro esistenza ha come unica utilità quella di validare i dati sull'utilizzo del servizio Messenger presso i partner commerciali[19].

[19] Gli stessi partner che acquistano spazi pubblicitari o contenuti personalizzati sul client ufficiale, ad esempio.

Indice

Riferimenti bibliografici

[Bertasi(2005)] Bertasi, P. (2005): *Progettazione e realizzazione in Java di una rete peer to peer anonima e multifunzionale*. Università degli studi di Padova.

[Day et al.(2000)Day, Rosenberg, and Sugano] Day, M.; J. Rosenberg; and H. Sugano (2000): A Model for Presence and Instant Messaging. *ietf.org*.

[Handley et al.(1999)Handley, Schulzrinne, Schooler, and Rosenberg] Handley, M.; H. Schulzrinne; E. Schooler; and J. Rosenberg (1999): SIP: Session Initiation Protocol. *ietf.org*.

[Hypotetic.org(2003a)] Hypotetic.org (2003a): MSN Messenger Protocol - General overview. `http://www.hypothetic.org/docs/msn/general/overview.php`.

[Hypotetic.org(2003b)] Hypotetic.org (2003b): MSN Messenger Protocol - Projects. `http://www.hypothetic.org/docs/msn/resources/projects.php`.

[Microsoft(2011)] Microsoft (2011): Machine Translation - the Bot, the project and everything in between... `http://mtbotprototype.spaces.live.com`.

[Microsoft.com(2007)] Microsoft.com (2007): Vulnerability in MSN Messenger and Windows Live Messenger Could Allow Remote Code Execution. `http://www.microsoft.com/technet/security/Bulletin/MS07-054.mspx`.

[Montini(2009)] Montini, N. (2009): *Pari Message: MSN*. Università degli studi di Padova.

[Movva and Lai(1999)] Movva, R. and W. Lai (1999): MSN Messenger Service 1.0 Protocol. *ietf.org*.

[MSNFanatic(2008)] MSNFanatic (2008): MSNC. `http://msnpiki.msnfanatic.com/index.php/MSNC`.

[MSNFanatic(2009)] MSNFanatic (2009): MSNC:Client Capabilities. `http://msnpiki.msnfanatic.com/index.php/MSNC:Client_ID`.

[MSNFanatic(2011a)] MSNFanatic (2011a): Fanatic Forum. `http://www.fanatic.net.nz`.

[MSNFanatic(2011b)] MSNFanatic (2011b): MSNPiki. `http://msnpiki.msnfanatic.com`.

[oleavr(2006)] oleavr (2006): MSNP15 authentication scheme REd. `http://www.openrce.org/blog/view/449`.

[Tolsma(2005)] Tolsma, S. (2005): The Context field of MSNSLP messages. `http://users.skynet.be/fa258499/temp/file_transfer_invite_context_field.pdf`.

[Vallini(2009)] Vallini, G. (2009): *Progettazione e realizzazione in Java di un client instant messaging multi-protocollo*. Università degli studi di Padova.

[W3C(2007)] W3C (2007): SOAP Version 1.2 Part 0: Primer (Second Edition). `http://www.w3.org/TR/soap12-part0`.

[Wikipedia(2010)] Wikipedia (2010): GUID. `http://it.wikipedia.org/wiki/GUID`.

[Wikipedia(2011a)] Wikipedia (2011a): Comparazione degli instant messenger. `http://it.wikipedia.org/wiki/Comparazione_degli_instant_messenger`.

[Wikipedia(2011b)] Wikipedia (2011b): Specials (Unicode block). `http://en.wikipedia.org/wiki/Replacement_character#Replacement_character`.

[Yergeau(2003)] Yergeau, F. (2003): UTF-8, a transformation format of ISO 10646. *ietf.org*.