Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea Triennale in Ingegneria Informatica

TESI DI LAUREA

PariPari-VoIP: Implementazione Protocollo SDP

Autore:
Marco Fabiani

Relatore:
Ch.mo Prof. Enoch Peserico Stecchini Negri De Salvi

Correlatore:
Dott. Paolo Bertasi

Anno Accademico 2010-2011

Comunicare l'un l'altro, scambiarsi informazioni è natura; tenere conto delle informazioni che ci vengono date è cultura

(Johann Wolfgang Goethe)

Indice

Capitolo 1

Introduzione

1.1 PariPari

PariPari[1] consiste in una rete peer-to-peer nata intorno alla metà degli anni 2000. È una rete serverless sviluppata in Java per permettere un'elevata portabilità, basata su una variante di Kademlia, garantisce l'anonimato dei suoi nodi, fornisce un sistema di crediti più intelligente di reti come ED2K e, soprattutto, è multifunzionale.
Le funzionalità della rete sono numerose e spaziano da un lato legato maggiormente al nucleo del sistema (DHT, Storage, Connectivity...) ad uno più vicino all'utente, come File Sharing (condivisione di file), Instant Messaging (chat, messaggistica istantanea), Torrent ed altri.
In questo frangente l'attenzione si soffermerà sulla funzionalità di chiamata vocale punto-punto o conferenza, con potenzialità future di videochiamata.
Per rendere disponibile questa opzione all'interno di PariPari, è necessario implementare la tecnologia **VoIP**, un insieme di protocolli che consentono la comunicazione tra due o più nodi della rete.

1.2 VoIP in sintesi

Voice over Internet Protocol (Voice over IP, VoIP) è un termine generico utilizzato per una famiglia di metodologie, protocolli di comunicazione e tecnologie di trasmissione per l'instaurazione di sessioni multimediali (in origine prevalentemente audio) su reti Internet Protocol (IP). Altri termini frequentemente utilizzati come sinonimo di VoIP sono telefonia IP, telefonia via Internet (tutti i servizi di comunicazione -voce, fax, SMS...- trasportati via Internet piuttosto che dalla classica rete telefonica pubblica commutata PSTN[2]), Voice over Broadband (VoBB) e telefonia a banda larga.
I passi necessari per avviare una chiamata telefonica di tipo VoIP sono:

[1] http://paripari.it/
[2] Public Switched Telephone Network

- signaling e media channel setup
- digitalizzazione del segnale analogico vocale
- compressione del segnale (opzionale)
- pacchettizzazione e trasmissione dei pacchetti su una rete switched-packet

Il ricevitore svolgerà, in maniera inversa, un procedimento simile per la ricostruzione del flusso vocale originale.
I sistemi VoIP utilizzano di norma una serie di protocolli di controllo di sessione per supervisionare processi di avvio e termine della chiamata, acquisizione e controllo dei flussi multimediali, allocazione di risorse utili ed altre funzioni necessarie a rendere stabile ed efficiente la sessione in corso.

I protocolli più diffusi che vengono implementati nei sistemi VoIP sono:

- H.323 (uno dei primi protocolli introdotti)
- IP Multimedia Subsystem (IMS)
- Media Gateway Control Protocol (MGCP)
- Real-time Transport Protocol (RTP)
- Session Initiation Protocol (SIP)
- Session Description Protocol (SDP)

Un primo significativo sviluppo dei sistemi di telefonia IP si è visto a partire dal 2004, grazie alla diffusione sempre più massiccia delle reti a banda larga al pubblico.
Oggi uno dei software più popolari ed utilizzati che sfrutta in maniera importante questa tecnologia è Skype[3] , il quale gestisce una rete esclusiva di tipo VoIP basata in parte su un'architettura peer-to-peer[4].
A differenza di applicazioni *open-source* come PariPari, Skype non fornisce al pubblico il proprio protocollo, di conseguenza tutte le applicazioni ufficiali che ne fanno uso sono *proprietarie*[5].

1.3 Il protocollo SIP

Per comprendere appieno la struttura ed il funzionamento del protocollo SDP, è necessario innanzitutto trattare almeno brevemente il SIP.
Session Initiation Protocol[6] è un protocollo di livello applicazione che gestisce in modo generale una sessione di comunicazione tra due o più entità, ovvero fornisce meccanismi per instaurare, modificare e terminare una determinata sessione. Attraverso il protocollo SIP possono essere trasferiti dati di diverso tipo

[3] `www.skype.com`
[4] struttura in cui un certo numero di nodi equivalenti (i peer) funge sia da client che da server verso il resto della rete
[5] dette anche *closed-source*, l'esatto contrario dell'open source
[6] http://www.ietf.org/rfc/rfc3261.txt

(audio, video, messaggistica testuale...), inoltre favorisce un'architettura modulare e scalabile, ovvero capace di crescere con il numero degli utilizzatori del servizio. Queste potenzialità hanno fatto sì che il SIP sia, oggi, il protocollo VoIP più diffuso nel mercato residenziale e business, sorpassando nettamente altri protocolli quali H.323 ed MGCP.
Il modello di funzionamento è basato sulla modalità richiesta/risposta simile a quella presente nel protocollo HTTP, mentre le applicazioni su cui agisce sono ovviamente piuttosto diverse. Le funzionalità messe a disposizione da SIP si possono raggruppare in cinque categorie:

1. Localizzazione dell'utente: determinare il dispositivo corretto con cui comunicare per raggiungere un determinato utente

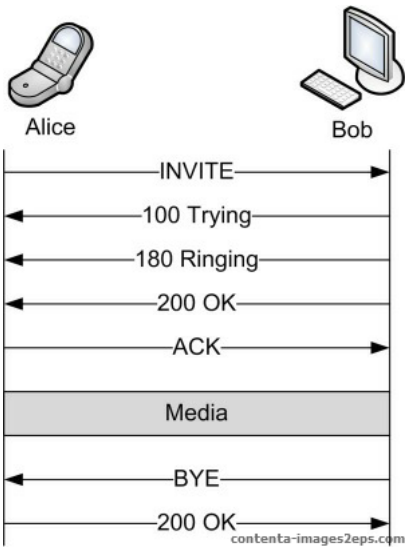
2. Disponibilità dell'utente: determinare se l'utente vuole prendere parte ad una particolare sessione di comunicazione oppure se ne ha i requisiti per sostenerla

3. Potenzialità dell'utente: determinare i *media* utilizzabili ed i relativi schemi di codifica

4. Instaurazione della sessione: stabilire i parametri della sessione (es. numeri di porta) che devono essere utilizzati dalle parti coinvolte nella comunicazione

5. Gestione della sessione: un ampio spettro di funzioni che comprendono, tra le altre, il trasferimento di sessioni e la modifica dei parametri

Nello specifico, la sessione viene instaurata e gestita, in base al RFC3261[7], come nella seguente figura:



[7] Request For Comments, documento che riporta informazioni o specifiche riguardanti nuove ricerche, innovazioni e metodologie dell'ambito informatico o, più nello specifico, di Internet.

Alice desidera chiamare Bob, ha a disposizione un telefono cellulare, mentre Bob si trova davanti al suo PC. Questa differenza di dispositivi non comporta alcuna modifica nella struttura di base della sessione in quanto si sfrutta un *proxy*, un punto di contatto con il chiamante a cui vengono inviate le richieste iniziali, le quali verranno poi inoltrate direttamente al ricevente. Questo consente di individuare dove si trova l'utente ed attraverso cosa vuole essere raggiungibile (potrebbe avere a disposizione, ad esempio, un PC quando si trova in ufficio e un dispositivo palmare quando è in viaggio), in modo da far fronte immediatamente alle peculiari caratteristiche di ciascun dispositivo fin dalle richieste iniziali, con l'ausilio di un altro terminale intermedio.
Alice invia a Bob un messaggio INVITE (cioè un messaggio, secondo il protocollo, di invito ad una sessione di comunicazione multimediale) che potrebbe essere composto, ad esempio, in questo modo:

```
INVITE sip:bob@biloxi.com SIP/2.0
Via:  SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
Max-Forwards:  70
To:  Bob <sip:bob@biloxi.com>
From:  Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq:  314159 INVITE
Contact:  <sip:alice@pc33.atlanta.com>
Content-Type:  application/sdp[8]
Content-Length:  142
```

Il primo messaggio di risposta che giunge ad Alice è il `100`[9] `trying` (Alice *sta provando* a stabilire una comunicazione con Bob), proveniente dal proxy che, nel frattempo, inoltra l'INVITE a Bob.
Una volta giunto il messaggio, il terminale di Bob risponderà con un messaggio `180 ringing`, generando così un *ring tone* nel suo dispositivo.
Nel caso in cui Bob fosse a disposizione e pronto ad avviare la conversazione con Alice, egli *alzerà la cornetta* (azione figurata). Questa azione genera un messaggio `200 ok` che viene inviato ad Alice, il cui dispositivo risponderà con un messaggio di *acknowledgment* (ACK) direttamente a Bob (non è più necessario l'utilizzo del proxy). Una volta giunto a destinazione l'ACK, il flusso di dati tra le due parti coinvolte può avere inizio.
Quando uno dei due utenti ha intenzione di concludere la sessione (nel caso della figura è Bob a farlo), invia un messaggio BYE che, in condizioni normali, provoca la risposta `200 ok` e la fine della conversazione.

1.4 SIP in PariPari

Attualmente il protocollo SIP viene implementato efficacemente in PariPari all'interno del modulo VoIP, per la gestione fondamentale di una conversazione multimediale tra due o più utenti.
Nello specifico, la struttura di base della libreria SIP prevede una serie di meto-

[8] in realtà il messaggio conterrebbe anche i campi riguardanti il SDP, ma al momento vengono trascurati e rivisti con un dettaglio decisamente superiore in seguito

[9] ogni messaggio di risposta possiede uno specifico codice

di dedicati alla realizzazione di un messaggio con tale protocollo, controllando se e quando è necessario trasmettere un messaggio di richiesta (`request`) o di risposta (`response`); ognuno di essi viene processato e trattato opportunamente una volta giunto a destinazione dal flusso di trasmissione.
Il codice sorgente è stato riscritto quasi da zero, seguendo le linee guida fornite dalla documentazione ufficiale, ma alcune classi particolarmente indicate per le esigenze attuali di PariPari sono state importate dalla libreria ufficiale *open source* riguardante il protocollo SIP.
Allo stato attuale delle cose, il modulo VoIP di PariPari consente solo la comunicazione audio sia punto-punto che in modalità multicast, grazie al protocollo SIP che permette l'handshake di base per stabilire la sessione.

Nonostante il protocollo SDP offra una gamma molto più vasta di opzioni rispetto alle effettive esigenze attuali di PariPari, si è deciso di implementarlo ugualmente e legarlo al protocollo SIP esistente. Questa scelta progettuale non deriva solamente dall'esigenza di uniformità rispetto agli altri software di questa tipologia presenti sul mercato (ormai qualsiasi programma che fa uso di una trasmissione multimediale gestita dal protocollo SIP la accompagna ad una descrizione di sessione realizzata tramite SDP), ma la presenza di una libreria che permette di fornire numerosi dettagli sulla sessione potrà consentire, in futuro, lo sviluppo di un sistema di chiamata molto più raffinato di quello attualmente disponibile (che permette una connessione e trasmissione tra utenti, ma non consente di adattare la sessione a seconda delle caratteristiche di banda, lingua, software o altro ancora).

Capitolo 2

Il protocollo SDP

2.1 Struttura di base

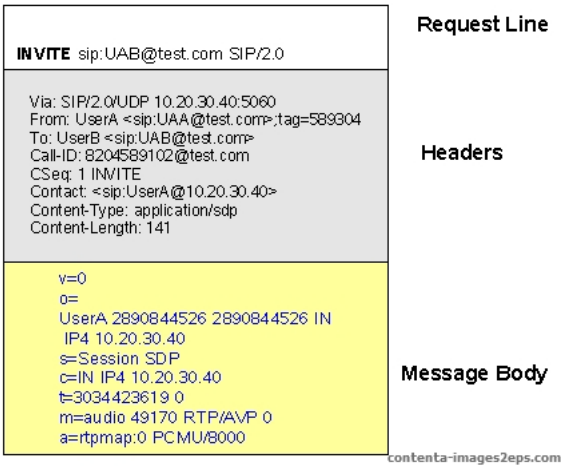
Il protocollo SDP (**Session Description Protocol**[1]) si occupa di fornire un formato per la descrizione della sessione in corso. Una prima definizione esauriente del protocollo viene pubblicata come RFC2327[2] ad aprile 1998, per poi subire un'importante rielaborazione a luglio 2006 nel RFC4566, attualmente la documentazione più completa riguardante l'argomento.
Viene utilizzato in svariati protocolli di trasporto, come SAP (Session Announcement Protocol), SIP (Session Initiation Protocol), RTSP (Real Time Streaming Protocol) ed HTTP (Hypertext Transport Protocol). Nel progetto PariPari il protocollo SDP è stato implementato all'interno del modulo VoIP insieme al protocollo SIP, in modo da ottimizzare il modello chiamata/risposta con ulteriori informazioni fornite dagli utenti coinvolti nella sessione.
Ogni messaggio (**Session Description** da qui in poi) realizzato con SDP deve essere **necessariamente** dotato di:

- versione del protocollo
- informazioni sul mittente
- nome della sessione
- informazioni sul tempo di sessione (**Time Description**)

È possibile (e consigliato per sfruttare tutte le potenzialità del protocollo) arricchire il messaggio con altri campi come larghezza di banda, attributi vari, informazioni sulla sessione e altro ancora. In particolare, è caldamente consigliato l'utilizzo di campi riguardanti informazioni sulle caratteristiche multimediali della sessione (media).
È doveroso ricordare che sono comunque campi facoltativi.

[1] da non confondere con Service Discovery Protocol, che riguarda invece la tecnologia Bluetooth
[2] da M. Handley e V. Jacobson

Un messaggio SIP completo di informazioni costruite dal protocollo SDP si presenta tipicamente nel seguente modo:



La figura rappresenta un blocco di informazioni suddivise in tre parti:

- Request Line, in cui viene riportato il tipo di richiesta effettuata dal mittente del messaggio
- Headers, contenente tutte le informazioni necessarie riguardanti il protocollo SIP
- Message Body, il vero e proprio fulcro del protocollo SDP

Il Message Body della figura non è altro che l'intestazione completa del protocollo oggetto di trattazione, in cui è piuttosto semplice riconoscere caratteristiche come la versione in corso, il nome della sessione ed il media che si sta utilizzando (altri campi sono decisamente più criptici nel loro significato, verranno discussi in seguito).
L'importanza di tenere in considerazione i protocolli SIP ed SDP contemporaneamente è, a questo punto del discorso, logicamente chiara, perché il legame tra le due tecnologie è molto saldo ed imprescindibile: il messaggio SDP è il corpo del messaggio SIP.
Senza una delle due intestazioni la sessione non può essere davvero efficiente. Se mancasse il protocollo SDP[3] sarebbe comunque possibile realizzare una conversazione multimediale con uno o più utenti in tempo reale, ma mancherebbero informazioni significative per rendere la sessione stabile, non essendoci fin dal principio la *negoziazione* delle risorse in gioco. Per esempio non si potrebbe mai sapere se il ricevente sia disponibile per una videochiamata oppure solo per una conversazione audio.
Privando invece il messaggio dell'intestazione del protocollo SIP, non si potrebbe nemmeno realizzare la sessione tra due utenti, in quanto mancherebbero le informazioni necessarie per dare avvio all'handshake descritto nel precedente capitolo.

[3] situazione di partenza in PariPari

Il funzionamento dei due protocolli in contemporanea prevede che, nel messaggio, l'intestazione riguardante il protocollo SDP sia separato da **una ed una sola** riga di testo vuota.

2.2 Session Description

Session Description costituisce la spina dorsale di tutto l'insieme di campi rappresentabili dal protocollo SDP.
Ogni campo è costituito concettualmente (e anche visivamente, mostrando a video il contenuto del pacchetto) da questo semplice formato:

```
<type>=<value>
```

In cui `<type>` rappresenta una ed una sola lettera dell'alfabeto inglese, mentre `<value>` costituisce uno o più valori possibili di quel corrispondente campo, a seconda del numero delle informazioni che richiede nello specifico.
Ovviamente `<type>` non può essere un carattere scelto a caso, ma deve far parte del ridotto set di lettere che, nel protocollo, abbiano un significato preciso. Nella versione attuale di SDP i *types* validi sono i seguenti:

```
Session description
v= (protocol version)
o= (originator and session identifier)
s= (session name)
i=* (session information)
u=* (URI of description)
e=* (email address)
p=* (phone number)
c=* (connection information - not required if included in all media)
b=* (zero or more bandwidth information lines)
One or more time descriptions (t= and r= lines; see below)
z=* (time zone adjustments)
k=* (encryption key)
a=* (zero or more session attribute lines)
Zero or more media descriptions

Time description
t= (time the session is active)
r=* (zero or more repeat times)

Media description, if present
m= (media name and transport address)
i=* (media title)
c=* (connection information - optional if included at session level)
b=* (zero or more bandwidth information lines)
k=* (encryption key)
a=* (zero or more media attribute lines)
```

(I campi contrassegnati da un asterisco sono facoltativi).
Il set di lettere è stato scelto deliberatamente piccolo e non ci sono le intenzioni di estenderlo, pertanto un *parser*[4] capace di riconoscere solo ed esclusivamente le lettere con un significato e di ignorare le altre, può tranquillamente essere utilizzabile non solo in questa versione del protocollo, ma anche in eventuali versioni successive senza doverlo aggiornare in maniera radicale.
Alcuni campi (`c=, b=, k=, a=`) vengono ripetuti nel Media Description, oltre che nel Session Description. Questo perché è possibile, concettualmente, assegnare alcuni di questi campi ad uno dei media utilizzati o a tutta la sessione. Per esempio, in un messaggio di questo tipo:

```
+Frame: Base frame properties
+ETHERNET: ETYPE = 0x0800 : Protocol = IP:  DOD Internet Protocol
+IP: ID = 0x1A11; Proto = UDP; Len: 752
+UDP: Src Port: Unknown, (1291); Dst Port: Unknown (5060); Length = 732 (0x2DC)
-SIP: INVITE Request (Length = 724)
-SIP: Request Line (Length = 35)
   SIP: INVITE sip:172.29.132.255 SIP/2.0
-SIP: Headers (Length = 327)
   SIP: Via: SIP/2.0/UDP 172.29.132.242:1241
   SIP: From: "sscarro" <sip:sscarro@RESKITCOMPUTER1>;tag=71181d00-b164-4386-a7b0-5e195bdaf98b
   SIP: To: <sip:172.29.132.255>
   SIP: Call-ID: 19862480@172.29.132.242
   SIP: CSeq: 1 INVITE
   SIP: Contact: <sip:172.29.132.242:1241>
   SIP: User-Agent: Windows Messenger/1.0
   SIP: Content-Type: application/sdp
   SIP: Content-Length: 362
-SIP: Message Body (Length = 362)
   SIP: v=0
   SIP: o=reskitcomputer1 0 0 IN IP4 172.29.132.242
   SIP: s=session
   SIP: c=IN IP4 172.29.132.242
   SIP: t=0 0
   SIP: m=audio 49548 RTP/AVP 97 6 0 8 4 5 3
   SIP: a=rtpmap:97 red/8000
   SIP: a=rtpmap:111 SIREN/16000
   SIP: a=fmtp:111 bitrate=16000
   SIP: a=rtpmap:112 G7221/16000
   SIP: a=fmtp:112 bitrate=24000
   SIP: a=rtpmap:6 DVI4/16000
   SIP: a=rtpmap:0 PCMU/8000
   SIP: a=rtpmap:8 PCMA/8000
   SIP: a=rtpmap:4 G723/8000
   SIP: a=rtpmap:5 DVI4/8000
   SIP: a=rtpmap:3 GSM/8000
   SIP: m=video 50744 RTP/AVP 34 31
   SIP: a=rtpmap:34 H263/90000
   SIP: a=rtpmap:31 H261/90000
```

il campo `c=` (Connection, come verrà enunciato in seguito) viene assegnato direttamente a tutto il Session Description, mentre i vari attributi `a=` (Attribute) si riferiscono ai due media `audio` e `video`. Ciò è intuibile semplicemente guardando l'ordine in cui appaiono i campi: `c=` si riferisce a tutta la descrizione di sessione, non essendoci alcun media definito prima di esso; al contrario invece gli ultimi due attributi si riferiscono chiaramente all'ultimo media riportato, cioè `video`.
Entrando nello specifico in ciascun campo, Session Description può contenere:

[4] Algoritmo di riconoscimento sintattico di un linguaggio

- **Protocol Version** `v=`, campo che rappresenta la versione in corso del protocollo SDP. In questo momento la versione di default è la 0, non ne esistono altre, pertanto dovrà essere utilizzato sempre e comunque `v=0`
- **Origin** `o=<username> <sess-id> <sess-version> <nettype> <addrtype> <unicast-address>`, campo che restituisce informazioni riguardanti l'utente che ha dato origine alla sessione in corso. Tali informazioni sono, nello specifico:
  - `<username>`, il nome utente di chi ha avviato la sessione. Può essere '-' se l'host non supporta il concetto di ID utente, ma NON DEVE contenere spazi
  - `<sess-id>`, una stringa numerica che identifica univocamente la sessione in corso
  - `<sess-version>`, numero di versione per la descrizione di sessione. In questi due campi si raccomanda l'utilizzo di un *timestamp* NTP (Network Time Protocol) per garantirne l'univocità
  - `<nettype>`, stringa di testo che rappresenta il tipo di rete. Attualmente è accettabile solo il valore 'IN', che sta per 'Internet', ma in futuro potrebbero essere registrati nuovi valori
  - `<addrtype>`, stringa di testo che restituisce il tipo di indirizzo unicast (rappresentato nel punto successivo). Attualmente sono definiti solo i valori 'IP4' ed 'IP6' per le versioni 4 e 6 dell'indirizzo IP, ma in futuro, come per `<nettype>`, possono essere registrati nuovi valori
  - `<unicast-address>`, indirizzo della macchina da cui è stata creata la sessione
- **Session Name** `s=`, nome della sessione. Ogni Session Description deve avere un solo Session Name, il campo NON DEVE rimanere vuoto (se non ci fosse un nome significativo da dare, è sufficiente utilizzare un singolo spazio)
- **Session Information** `i=`, informazioni testuali riguardo la sessione. Come sopra, anche Session Information può essere definito al massimo una sola volta per Session Description ed una per ogni Media Description, ma può anche non essere presente, in quanto campo facoltativo.
  Non è necessario far analizzare il valore del campo da un parser
- **URI** `u=`, contiene un link opzionale che offre ulteriori informazioni sul tema in cui verte la sessione. Se specificato, va inserito prima del primo campo media che viene definito
- **Email Address** `e=`, contiene un indirizzo E-Mail riferito al responsabile della conferenza, che non deve essere necessariamente la stessa persona ad aver avviato la sessione
- **Phone Number** `p=`, contiene il numero di telefono[5] del responsabile della conferenza in corso.
  Entrambi questi campi sono opzionali ed entrambi possono ammettere una

[5] in formato internazionale, quindi necessariamente preceduto da un '+'

stringa di testo opzionale a loro associata, a patto che essa venga rappresentata tra parentesi, per esempio:

```
e=j.doe@example.com (Jane Doe)
```

- **Time Zone** `z=<adjustment time> <offset> <adjustment time> <offset>...` , serve ad impostare eventuali modifiche al tempo di sessione dovute a differenze di fuso orario. Ogni `<adjustment time>` viene *shiftato* del suo corrispondente `<offset>`, aggiungendo o sottraendo secondi in base al segno che possiede.
  Questa modifica si ripercuote nel Time Description, che verrà trattato in seguito.

Altri campi, come `connection` o `bandwidth`, siccome possono essere associati anche a Media Description, verranno discussi in seguito, quando si tratterà appunto l'argomento in questione.

2.3 Time Description

Time Description è esso stesso un campo di Session Description, a sua volta contenente altri campi in grado di fornire informazioni su tutto ciò che riguarda il tempo di una sessione.
Gli unici due elementi (di cui solo il primo obbligatorio) gestiti dal Time Description sono:

- **Timing** `t=<start-time> <stop-time>`, specifica il tempo di inizio e fine della sessione, basandosi sulla rappresentazione decimale dei valori di tempo calcolati in secondi, a partire dal 1900, dal protocollo NTP[6].
  È possibile introdurre molteplici campi Timing, ciascuno dei quali rappresenta un ulteriore periodo di attività della sessione.
  Se il campo `<stop-time>` è settato a zero, la sessione non è limitata e può iniziare solo dopo lo `<start-time>`. Se anch'esso viene impostato a zero, la sessione è considerata permanente. Quest'ultima opzione è presente e fattibile in determinate circostanze, ma generalmente si cerca di scoraggiarne l'utilizzo in quanto non si riporta alcuna informazione sulla durata della sessione

- **Repeat Times** `r=<repeat interval> <active duration> <offsets from start-time>`, campo opzionale estremamente utile nel caso in cui una certa sessione viene abitualmente ripetuta in uno stesso periodo di tempo.
  Si prenda come esempio una sessione che sia attiva alle ore 10 del lunedì ed alle ore 11 del martedì per un'ora, ogni settimana, per tre mesi: per ottenere questo risultato si deve innanzitutto impostare nel campo Timing il tempo di partenza pari al primo giorno in cui la sessione deve avviarsi.
  Il campo Repeat Times deve essere completato nel seguente modo:

[6] per convertire questi valori in tempo UNIX è necessario sottrarre il numero decimale 2208988800

1. `<repeat interval>` indica ogni quanto tempo la sessione deve ripartire, quindi verrà impostato il tempo equivalente ad una settimana (in secondi anch'esso)
2. `<active duration>` rappresenta la durata in cui la sessione, ogni volta che inizia, deve rimanere attiva, quindi si imposta il valore corrispondente ad un'ora
3. `<offsets from start-time>` si utilizzano quando i periodi di tempo da rappresentare sono più di uno. Nel caso specifico c'è bisogno di far avviare la sessione non solo ogni lunedì alle 10, ma anche ogni martedì alle 11, 25 ore dopo la prima attività; in questo caso quindi si impostano due valori di offset, a zero ed a 25 ore (cioè la differenza di tempo tra la prima e la seconda partenza della sessione

A questo punto si completa il campo Timing con lo `<stop-time>`, che assumerà il valore equivalente al valore di `<start-time>` PIÙ i tre mesi di tempo (sempre in secondi) richiesti dall'esempio.
Nello specifico, si può realizzare nel seguente modo:

```
t=3498142400[7] 3506091213
r=604800 3600 0 90000
```

Oppure, più comodamente, il campo RepeatTime può essere rappresentato come:

```
r=7d 1h 0 25h
```

Assegnando ai valori numerici un significato molto più intuitivo per lo sviluppatore che andrà eventualmente a leggere il codice sorgente.

2.4 Media Description

Media Description, come nel caso di Time Description, è un campo di Session Description che a sua volta può contenere altre informazioni all'interno della sua struttura. È facoltativo, ma una volta definito è necessario quantomeno inserire il campo `m=`, di cui si parlerà in seguito nel dettaglio.
Anche se facoltativo, Media Description, per la sua struttura, è caldamente incoraggiato ad essere sfruttato, perché questo è il vero componente del Session Description in grado di arricchire il protocollo SDP di informazioni utili riguardanti le risorse ed i media messi a disposizione dagli utenti in una specifica sessione.
I campi disponibili per realizzare e completare un Media Description sono:

[7] il valore in secondi si riferisce alla data 7/11/2010

- **Media Description** `m=<media> <port> <proto> <fmt>`, pur contenendo, come accennato in precedenza, diversi campi al suo interno, viene associato ad una lettera a parte. Ogni Media Description inizia con un `m=` e termina solo dopo un altro `m=`, oppure quando finisce tutto il Session Description.
  L'intestazione di questo campo comprende la seguente serie di informazioni utili:

  - `<media>`, molto semplicemente è il tipo di media utilizzato nella specifica descrizione. Attualmente può assumere i valori (semplici stringhe) `audio`, `video`, `text`, `application`, `message`, ma questa lista può espandersi in una futura versione del protocollo.
  - `<port>`, la porta in cui scorre il flusso del media specificato. È un campo molto dinamico che consente di definire più porte eventualmente utilizzate per uno stesso media. Ad esempio, un Media Description definito in questo modo:

    ```
    m=video 49170/2 RTP/AVP 31
    ```

    indica che le coppie di RTP/RTCP utilizzate sono due, una sfrutta le porte 49170 e 49171, l'altra le porte 49172 e 49173.
    Si possono realizzare combinazioni molto interessanti con altri campi dell'intero Session Description, ad esempio:

    ```
    c=IN IP4 224.2.1.1/127/2
    m=video 49170/2 RTP/AVP 31
    ```

    implica che l'indirizzo 224.2.1.1 viene utilizzato con le porte 49170 e 49171, mentre l'indirizzo 224.2.1.2 viene utilizzato con le porte 49172 e 49173.
    Se invece le porte sfruttate non dovessero essere contigue, è necessario intervenire nei campi Attribute, ad esempio in `a=rtcp:`
  - `<proto>`, protocollo di trasporto utilizzato. Al momento ammette tre possibili opzioni, che potrebbero essere estese in futuro: `udp`, `RTP/AVP`[8] (RTP Profile for Audio and Video Conferences), `RTP/SAVP` (Secure RTP)
  - `<fmt>`, *media format description*, descrive il formato del media. È molto legato al campo `<proto>`: se quest'ultimo assume valore `RTP/AVP` oppure `RTP/SAVP`, il formato conterrà i numeri di RTP payload; se invece vale `udp`, il formato deve fare riferimento all'informazione contenuta nel campo `<media>`. Possono esserci più valori di `<fmt>`

  I campi che seguono possono essere definiti sia all'interno di uno specifico Media Description, sia in riferimento a tutto l'intero Session Description

- **Connection Data** `c=<nettype> <addrtype> <connection-address>`, contiene informazioni riguardanti l'indirizzo di connessione. È un campo

[8] www.ietf.org/rfc/rfc3551.txt

particolare, perché pur essendo definito *facoltativo* sia in Session Description che nei vari Media Descriptions, è necessario che sia presente almeno una volta in tutto il messaggio del protocollo. In particolare può essere presente una ed una sola volta in Session Description e tante volte quanti Media Descriptions sono definiti: in questo caso il dato presente in un `m=` prende il sopravvento su quello definito nell'intera sessione.
I sottocampi definiti sono:

- `<nettype>`, stringa che rappresenta il tipo di rete utilizzato. Attualmente, come nel campo Origin, è ammissibile solo il valore `IN` (sta per 'Internet'), ma nelle versioni successive di SDP è possibile che il concetto venga esteso.
- `<addrtype>`, indica il tipo di indirizzo. Come sopra, si possono introdurre solo i valori `IP4` ed `IP6`, ma altri valori potrebbero essere registrati in futuro
- `<connection-address>`, è il vero e proprio indirizzo di connessione. È possibile introdurre due ulteriori sotto-campi facoltativi: TTL (Time To Live, un numero compreso tra 0 e 255) e numero di indirizzi (nel caso di molteplici gruppi multicast da gestire), entrambi preceduti da "/". Per esempio, in un campo definito in questo modo:

  ```
  c=IN IP4 224.2.1.1/127/3
  ```

  il numero 127 rappresenta il TTL della connessione, mentre il 3 rappresenta il numero di indirizzi contigui da gestire, a partire da 224.2.1.1. Questa scrittura equivale a definire, in un Media Description:

  ```
  c=IN IP4 224.2.1.1/127
  c=IN IP4 224.2.1.2/127
  c=IN IP4 224.2.1.3/127
  ```

- **Bandwidth** `b=<bwype>:<bandwidth>`, indica la larghezza di banda proposta per l'utilizzo nella sessione o nello specifico media, a seconda di dove viene definito.
  Il valore `<bandwidth>` è numerico e rappresenta di default la larghezza di banda in kilobit per secondo. Per quanto riguarda `<bwtype>` sono attualmente disponibili due opzioni:

  - `CT` (sta per *conference total*), indica la banda totale utilizzata nella conferenza
  - `AS`, indica la massima larghezza di banda utilizzabile in una specifica applicazione

  È possibile definire ulteriori valori di `<bwtype>` semplicemente aggiungendo il prefisso `X-` (che sta per *experimental*), ma la documentazione ufficiale sconsiglia una simile manovra, aspettando eventualmente nuovi valori che potrebbero essere registrati per le prossime versioni.

- **Encryption Keys** `k=<method>:<encryption key>`, definisce un semplice algoritmo di scambio di chiavi per rendere sicura la sessione. Non essendo consigliato il suo utilizzo, nella attuale versione di SDP, si ritiene futile spendere ulteriori parole sull'argomento.
- **Attributes** `a=<attribute>:<value>`, rappresentano tutta una vasta serie di informazioni utili per ottimizzare al meglio la sessione. Possono essere introdotti in due forme possibili:
  - `a=<flag>`, attributi che indicano una specifica proprietà della sessione (ad esempio `a=recvonly`, indica la modalità sola ricezione)
  - `a=<attribute>:<value>`, attributi che indicano una determinata caratteristica della sessione o del media specifico, assumendo un valore legato a tale informazione definita

  Per la vastità dell'argomento, esso viene trattato nel paragrafo immediatamente successivo ad esso.

2.5 Attributi

Gli attributi, presentati nel precedente paragrafo, possono rappresentare molteplici informazioni riguardanti non solo modalità di funzionamento, ma anche dettagli sul nome, tipo della conferenza o addirittura elementi più 'estetici', come la posizione del *workspace* sullo schermo.
Nel dettaglio, l'elenco di attributi disponibile nell'attuale versione di SDP prevede:

- `a=cat:<category>`, indica la categoria della sessione, in modo che il ricevente sia in grado, tramite un apposito filtro, di ricercare o evitare sessioni di una certa categoria. Funziona solo a livello di sessione
- `a=keywds:<keywords>`, è un identificatore utile a descrivere i propositi della sessione, in modo che il ricevitore, come per il `cat`, sia in grado di scegliere se evitarlo o accedere ad essa. Anche questo attributo è a livello di sessione
- `a=tool:<name and version of tool>`, rappresenta il nome e la versione del tool utilizzato per realizzare la descrizione di sessione. È a livello di sessione
- `a=ptime:<packet time>`, associato ai media, rappresenta il tempo di durata (in millisecondi) di un pacchetto. Attualmente ha senso utilizzare questo attributo per media di tipo `audio`, ma in futuro potrebbe essere possibile sfruttare questo campo per altri tipi di media
- `a=maxptime:<maximum packet time>`, rappresenta la massima quantità di media incapsulabile in un singolo pacchetto, espressa in millisecondi. Anch'essa viene generalmente associata a media di tipo `audio`, ma non si esclude un possibile utilizzo in altri campi
- `a=rtpmap:<payload type> <encoding name>/<clock rate> [/<encoding parameters>]`, contiene parametri riguardanti la codifica di un dato media.
  Nello specifico:

- `<payload type>`, il numero di payload del media (viene preso come valore il numero contenuto in `<fmt>` del media in considerazione)
- `<encoding name>`, il tipo di codifica utilizzato
- `<clock rate>`, la frequenza di clock (in Hz) del campionamento
- `<encoding parameters>`, facoltativo, contiene il numero di canali audio utilizzati nel media di tipo `audio` (se omesso significa che viene utilizzato un solo canale). Non viene utilizzato, al momento, per altri tipi di media.

Un messaggio contenente i seguenti campi:

```
m=audio 49232 RTP/AVP 98
a=rtpmap:98 L16/16000/2
```

Rappresenta un media di tipo audio che scorre attraverso la porta 49232, utilizza il formato di trasporto RTP/AVP e contiene il media format description pari a 98. Viene codificato tramite un sistema lineare a 16 bit (L16) a due canali con frequenza di campionamento pari a 16 kHz.
È un attributo di livello media.

- `a=recvonly`, utilizzabile sia a livello di sessione che di media, indica che il flusso viene acquisito solo in ricezione.
  Questa modalità di utilizzo si riferisce solo ad il/i media, non per protocolli di controllo associati (ad esempio un sistema RTP in modalità `recvonly` deve continuare ad inviare pacchetti RTCP)
- `a=sendonly`, come il `recvonly`, ma implica una modalità di sola spedizione di pacchetti. Può essere associato alla sessione o ad un media, ma generalmente vale solo per quest'ultimo parametro
- `a=inactive`, come i due precedenti attributi, questa volta il flusso è completamente inattivo in entrambi i sensi. Può essere associato alla sessione o ad un media
- `a=sendrecv`, implica la normale modalità di funzionamento send-receive. Viene considerata la modalità di default nel caso in cui uno dei tre precedenti attributi non dovesse essere definito in alcun modo
- `a=orient:<orientation>`, normalmente utilizzato in un tool di presentazione, serve a definire la posizione del workspace sullo schermo. I valori ammessi sono `portrait`, `landscape` e `seascape` (il `landscape` capovolto). È a livello media
- `a=type:<conference type>`, a livello di sessione, specifica il tipo di conferenza. I valori attualmente registrati sono:
  - `broadcast` (che, di default, richiede una modalità di funzionamento `recvonly`)
  - `meeting` (che implica la modalità `sendrecv`)

  - `moderated`
  - `test`
  - `H332` (quando la sessione in corso fa parte di una sessione H.332)

- `a=charset:<character set>`, a livello di sessione, specifica il set di caratteri utilizzato per visualizzare il nome della sessione e le informazioni testuali che la riguardano. Il set predefinito è il ISO-10646 nella codifica UTF-8[9]

- `a=sdplang:<language tag>`, può essere specificato a livello di sessione o di media: nel primo caso indica la lingua dell'intera descrizione di sessione, nel secondo specifica la lingua per ogni campo informativo associato al media in questione.
  In generale è possibile (e sensato) aggiungere più attributi `sdplang`, nel caso in cui le informazioni rappresentate fossero appartenenti a lingue diverse, ma in generale non è un'operazione consigliata

- `a=lang:<language tag>`, leggermente diverso dal `sdplang`, rappresenta la lingua utilizzata nella sessione o media ad esso associata (dove la lingua di un media ha la priorità su quella della sessione nel caso in cui fossero diverse).
  È possibile specificare più lingue in caso di necessità: in questo caso l'ordine di tali attributi nella descrizione indica l'ordine di importanza delle lingue vigenti nella sessione

- `a=framerate:<frame rate>`, indica il frame rate video massimo, in frames/sec. È a livello media, e definito solo per quanto riguarda il `video`

- `a=quality:<quality>`, specifica la qualità video della sessione, nel caso in cui sia disponibile questo tipo di media. È un valore intero compreso tra 0 e 10, la scala che rappresenta è:

  - `10`, la miglior qualità video che lo schema di compressione può ottenere
  - `5`, il valore predefinito, non fornisce alcuna informazione significativa sulla qualità
  - `0`, la peggior qualità video che il progettista del codec pensa sia ancora utilizzabile dignitosamente!

- `a=fmtp:<format> <format specific parameters>`, a livello media, è un attributo che assegna parametri definiti in un formato particolare (`<format>`) che, normalmente, non potrebbe essere trasmesso in quanto il protocollo SDP non lo comprenderebbe
  Il campo `<format specific parameters>` consiste in un set di parametri associati al formato, utilizzabili da SDP per la trasmissione

[9] http://tools.ietf.org/html/rfc3629

Capitolo 3

jSDP

3.1 Cenni preliminari

La libreria jSDP per Java permette la possibilità di costruire una struttura dati in grado di fornire un'efficace rappresentazione di un pacchetto SDP, basandosi sulla RFC4566.
Concettualmente le classi che compongono la libreria sono piuttosto semplici, grazie al paradigma di programmazione *Object Oriented* : l'idea di base è quella di associare un oggetto per ogni campo dichiarato nel protocollo, ognuno dotato di metodi *getters* e *setters* di tutti i parametri che lo riguardano, la più classica delle norme di programmazione ad oggetti.
Generalmente ciascun parametro di ogni campo è un numero o al massimo una stringa alfanumerica, in quanto non è necessario complicare la libreria con una serie spropositata di classi a parte in grado di definire ogni singolo parametro previsto, rischiando di ritrovarsi parecchi metodi che verrebbero invocati al massimo un paio di volte in tutto il modulo.
Per preservare efficacemente la struttura gerarchica del protocollo, esistono altre classi di tipo *Description* necessarie per realizzare istanze di oggetti in grado di inglobare un certo numero di campi in corrispondenza delle tre grandi aree definite in precedenza (Sessione, Tempo, Media). Tali classi sono leggermente più complesse per via del numero di metodi da loro incorporati.

3.2 Interfacce

Per formalizzare quanto è stato detto nel paragrafo precedente riguardo i campi e le descrizioni, la libreria jSDP contiene due interfacce: *Field* (Campo) e *Description* (Descrizione), le quali rappresentano principalmente il tipo di dato che si vuole trattare.
Come verrà esplicato più nel dettaglio in seguito, queste interfacce vengono definite non per mettere a disposizione un modello strutturale della classe, ma per assegnare un tipo di dato più generico e verosimile su cui fare riferimento.
Se, ad esempio, uno sviluppatore si ritrovasse ad avere a che fare con variabili di tipo `Origin`, in un istante, osservando la documentazione, è possibile capire se il dato è un campo o una descrizione.

3.2.1 Interfaccia *Description*

L'interfaccia *Description* rappresenta tutte le classi le cui istanze sono descrizioni del protocollo (di sessione, di media o di tempo).
Il livello di complessità dell'interfaccia è pressoché nullo, in quanto non presenta altri metodi al di fuori dei classici `clone()` e `toString()`, cioè i metodi fondamentali della classe Object.
Le classi che implementano questa interfaccia sono:

- `SessionDescription`
- `TimeDescription`
- `MediaDescription`

3.2.2 Interfaccia *Field*

Come per la precedente interfaccia, *Field* viene implementata da tutte le classi che rappresentano un qualche campo definito nel protocollo. A differenza di *Description*, essa contiene anche la firma di un metodo `getType()`, che restituisce il tipo di campo dell'istanza (più nello specifico, restituisce un carattere corrispondente al tipo di campo del protocollo).
Le classi che implementano l'interfaccia sono:

- `Attribute`
- `Bandwith`[1]
- `Connection`
- `Email`
- `Information`
- `Key`
- `Media`
- `Origin`
- `Phone`
- `RepeatTime`
- `SessionName`
- `Time`
- `TimeZone`
- `Uri`
- `Version`

[1] probabilmente è un errore di scrittura dell'autore della libreria, dovrebbe essere *Bandwidth*

3.3 Modalità d'uso

Le classi elencate in precedenza sono necessarie e sufficienti per definire in maniera esauriente e corretta un messaggio SDP.
L'idea strutturale alla base della libreria suggerisce di configurare inizialmente i campi richiesti dall'applicazione, successivamente le istanze di oggetti rappresentanti tali campi dovranno essere introdotte nei Description corrispondenti (MediaDescription, TimeDescription). Questi ultimi dovranno essere inseriti nel SessionDescription (possono esserci più descrizioni di tipo media e tempo), insieme ai campi fondamentali della descrizione di sessione stessa, in modo da completare il messaggio in maniera opportuna.
Ovviamente, come descritto dalla documentazione, i campi indicati come facoltativi possono non essere inseriti nella descrizione di sessione. La libreria jSDP, nel caso limite in cui nessun campo venga in qualche modo configurato, provvederà a realizzare un messaggio minimo di questo tipo:

```
v=0
o=user 3496643150 3496643150 IN IP4 PC-user
```
(I campi `<sess-id>` e `<sess-version>` sono impostati casualmente)
```
s=-
t=0 0
```

Ogni singolo campo è dotato di una propria interfaccia di metodi che permette allo sviluppatore di configurare in poche istruzioni i parametri desiderati. Per esempio l'interfaccia di `RepeatTime` (uno dei campi più elaborati del protocollo) è costituita nel seguente modo:

```
void addOffset(long offset)

long getActiveDuration()

long[] getOffsets()

long getRepeatInterval()

boolean isTypedTime()

void setActiveDuration(long activeDuration)

void setOffset(long[] offsets)

void setRepeatInterval(long repeatInterval)

void setTypedTime(boolean typedTime)
```

I metodi elencati permettono di elaborare e restituire tutti i parametri del campo RepeatTime, definiti seguendo la logica e le istruzioni della documentazione del protocollo SDP.
Esistono altri metodi non necessariamente collegati strettamente al protocollo

SDP, ma in grado di facilitare e rendere più efficiente il compito allo sviluppatore: nell'esempio precedente si possono individuare due metodi, `isTypedTime()` e `setTypedTime(boolean typedTime)`, che servono a manipolare dati di tipo TypedTime. La classe `TypedTime` è stata introdotta nella libreria per rendere più intuitivo l'inserimento di valori temporali, infatti essa contiene alcune costanti moltiplicative corrispondenti al valore in secondi delle unità di tempo giorno, ora, minuto, secondo:

- d - days (86400 seconds)
- h - hours (3600 seconds)
- m - minutes (60 seconds)
- s - seconds (1 second)

Per evitare di fornire, agli sviluppatori, dei dati numerici apparentemente privi di senso, è possibile quindi ricorrere a tali costanti per moltiplicare una certa quantità di tempo (ad esempio `7*d` equivale ad una settimana di tempo in secondi).
Alcuni campi (come per esempio `Version` o `Origin`) contengono parametri che, allo stato attuale del protocollo, vengono dati come predefiniti (ad esempio i valori IN ed IP4 rispettivamente per i parametri `nettype` ed `addrtype`). Si sceglie di non voler modificare questo aspetto.

3.4 SDPFactory

La libreria jSDP offre la possibilità di poter creare ciascun campo definito dal protocollo non solo nel metodo tradizionale sfruttando il costruttore, ma anche *al volo* invocando una serie di metodi statici definiti nella classe `SDPFactory`. Essi sono elencati in maniera completa all'interno dell'**appendice 1** di questo documento.
Tutti i metodi di tipo *create* sono molto intuitivi ed immediati, permettono di realizzare istanze di oggetti utili per rappresentare efficacemente i campi elencati in precedenza. Alcuni di essi possono sollevare, in caso di input incorretti, un `SDPException`; in questo modo il sistema di creazione del messaggio SDP realizza subito un primo controllo dei dati rappresentati: l'eccezione infatti non è solo una banale verifica di tipologia dei parametri in ingresso, ma in certi casi si tratta già di un primo vero controllo sulla validità del dato in questione. Ad esempio, un frammento di codice del seguente tipo:

```
Media mAudio=SDPFactory.createMedia(“audio“, 16384, “RTP/AVP“, “31“);
MediaDescription md=new MediaDescription(mAudio);
```

solleverà un errore in esecuzione:

```
net.sourceforge.jsdp.SDPException:  This media description must have
a connection field
```

Significa che il MediaDescription in questione chiede necessariamente di contenere un campo Connection, supponendo che esso non sia definito nel principale SessionDescription.
Un altro esempio di controllo più dettagliato dei campi riguarda la classe Uri, se si definisce una nuova istanza di oggetto in questo modo:

```
Uri u=SDPFactory.createUri("87.4.176.146");
```

la riga di codice genererà un errore di questo tipo:

```
net.sourceforge.jsdp.SDPException:  Invalid URL: 87.4.176.146
```

perché l'indirizzo in questione non possiede un formato valido. Anche con un'istruzione del tipo:

```
Uri u=SDPFactory.createUri("www.google.it");
```

si verificherà nuovamente il sollevamento della stessa eccezione. È necessario che la stringa che rappresenta il parametro in ingresso del metodo includa, come prefisso, il formato `http://`. Solo così il controllo interno del dato restituirà un esito positivo e non verrà quindi sollevata alcuna eccezione.

I metodi `encode` permettono di prelevare una qualsiasi descrizione di sessione e di convertirla in un flusso output di tipo `OutputStream`. La differenza tra i due metodi sta nel parametro `encoding`, che definisce il tipo di codifica da utilizzare nella conversione.

I metodi `parseSessionDescription` invece permettono l'esatto contrario: viene preso in considerazione un input (può essere una stringa, una serie di campi, un file o un flusso di dati qualsiasi) ed esso viene rielaborato fino ad ottenerne automaticamente una descrizione di sessione completa. È a questo che serve l'eccezione `SDPParseException` citata in precedenza.

Capitolo 4

Da jSDP a PariPari-VoIP SDP

La libreria jSDP fornisce una semplice ma esauriente struttura dati finalizzata alla definizione di un messaggio di protocollo SDP. I metodi messi a disposizione sono stabili e quasi tutti di immediato utilizzo (e sono comunque corredati da una documentazione ben realizzata che permette di eliminare qualsiasi dubbio su alcuni parametri maggiormente criptici nella comprensione), permettendo allo sviluppatore di definire qualsiasi messaggio possibile immaginabile, nei limiti del protocollo; inoltre alcuni parametri, come visto in precedenza, vengono già controllati sintatticamente, senza essere costretti a mettere mano sul codice.
Tuttavia non è sufficiente prendere in mano la libreria e limitarsi a realizzare istanze di oggetti e richiamare metodi all'interno del codice sorgente delle classi principali del modulo VoIP. Si potrebbe anche fare, con un po' di attenzione, e si otterrebbe una struttura dati funzionante e verosimile nella definizione del protocollo SDP, ma una serie di motivi induce ad imboccare un'altra strada:

1. **Dispersività del codice**: jSDP consente di definire qualsiasi campo previsto dal protocollo SDP, ma l'associazione campo-classe, pur essendo molto intuitiva, rischia di rendere il codice molto dispersivo. In sintesi, lo sviluppatore rischia di ritrovarsi a definire campi e parametri sparsi per il sorgente, richiamare metodi completamente slegati tra di loro, per poi rischiare di non comprendere più cosa, come e quando debba essere inserito nella descrizione di sessione.
   Si sente la necessità di rendere più facilmente controllabile il codice, cercando di accorpare campi e funzioni comuni in poche classi specifiche, più complesse nella loro struttura interna, ma ugualmente semplici ed immediate nel loro utilizzo

2. **Prolissità delle classi preesistenti**: il punto descritto in precedenza è anche causa di un fenomeno abbastanza prevedibile: l'eccessivo numero di righe di codice aggiuntivo.
   La creazione di un messaggio SDP anche piuttosto semplice, direttamente all'interno delle classi adibite al protocollo SIP, provocherebbe un appesantimento notevole dei loro metodi e della leggibilità. Si cerca quindi di

realizzare una libreria che possa permettere allo sviluppatore di costruire, con l'ausilio di poche istruzioni, un messaggio SDP anche molto complesso

3. **Inconsistenza di dati**: forse il motivo più importante che spinge a realizzare PariPari-VoIP SDP (la libreria di PariPari che si occupa della gestione del protocollo SDP nella maniera più robusta e consistente possibile). Si supponga di scrivere il seguente frammento di codice:

   ```
   Attribute a=new Attribute
   (“AnswerToTheUltimateQuestionOfLifeUniverseAndEverything“, “42“)
   ```

   Il compilatore non batterà ciglio, non verrà sollevata alcuna eccezione ed il risultato sarà un messaggio contenente il campo:

   ```
   a=AnswerToTheUltimateQuestionOfLifeUniverseAndEverything:42
   ```

   Questo tipo di attributo non fa decisamente parte dell'insieme previsto nella documentazione del protocollo. In generale, pur essendoci dei controlli sull'immissione dei parametri (ci sono anche per il campo Attribute, ma riguardano solo la presenza di caratteri non alfabetici nella stringa che rappresenta il tipo di attributo), mancano molti altri paletti necessari ad impedire l'inserimento di dati scorretti, come quelli riportati nell'esempio. PariPari-VoIP SDP punta a risolvere questa situazione, fornendo metodi ad-hoc per l'inserimento di attributi o altri campi specifici che rispettano realmente i requisiti del protocollo SDP.

Per far fronte a queste motivazioni, si intende realizzare delle nuove classi in grado di sfruttare quelle già fornite da jSDP, ma che consentano lo sviluppatore di seguire le regole del protocollo senza rischiare in alcun modo (o quasi) di introdurre in circolo un messaggio dai parametri errati o inesistenti. Inoltre grazie ai nuovi metodi, il codice sorgente sarà più snello e leggibile anche una volta introdotto nei moduli principali del VoIP.
L'idea di funzionamento della nuova libreria rimarrà gerarchica come in jSDP, ma questa volta lo sviluppatore avrà a che fare con tre grosse classi (una per ogni Description) dotate di tutti i metodi necessari per manipolare qualsiasi campo previsto dal protocollo senza mai venire direttamente a contatto con uno solo di essi.
Le classi realizzate prendono il nome di SdpFactory, MediaFactory e TimeFactory; il loro funzionamento ed implementazione verranno trattati nel dettaglio nei prossimi paragrafi, iniziando prima dalle descrizioni *ausiliari* (media e tempo) per poi concludere con la vera e propria descrizione di sessione.

4.1 TimeFactory

`TimeFactory` è la classe riservata alla creazione di una o più descrizioni di tempo, a seconda dell'esigenza dello sviluppatore. È la più semplice delle tre e tratta solamente la configurazione dei parametri Time e RepeatTime di ciascun

TimeDescription, come previsto dalla documentazione ufficiale del protocollo.
L'idea principale alla base della classe è di sfruttare un `ArrayList` contenente le varie descrizioni di tempo come fosse una sorta di pila: un indice di tipo intero `nTimeDescription` tiene il conto di quante descrizioni sono state incluse nella struttura dati; tutti i metodi di manipolazione si riferiscono all'oggetto che punta all'indice dell'`ArrayList` corrispondente ad esso.
Qualora non si desiderasse più continuare con la configurazione dell'ultimo TimeDescription inserito, ma si volesse crearne una nuova istanza, il metodo adibito al compito effettua una operazione analoga al *push* di una pila. Da quel momento in poi, tutti i metodi di manipolazione della descrizione di tempo si riferiranno al TimeDescription appena inserito.
In caso di necessità si può eliminare quest'ultimo, con un altro metodo che svolge una sorta di *pop* (con tanto di restituzione dell'elemento in uscita) secondo la logica LIFO di una comune pila.

4.1.1 Metodi

`void timeSetting(long timeStart, long timeStop)`, permette di modificare determinati valori di partenza e di arresto di un periodo di tempo `Time`. Di default la classe, nel `TimeDescription`, assumerà i valori (0, 0).

`void addRepeatTime(long repeatInterval, long activeDuration, long offset)`, funziona come il metodo precedente e riguarda il campo RepeatTime. La differenza sta nel fatto che, essendoci la possibilità di inserire più RepeatTime in un TimeDescription, il metodo ne aggiunge uno in coda agli altri, senza toccarli.

`void addRepeatTime(long repeatInterval, long activeDuration, long[] offsets)`, come sopra, però si da la possibilità di aggiungere molteplici valori di offset, com'è lecito fare secondo la documentazione.

```
void addNewTimeDescription()
TimeDescription deleteLastTime()
```

Sono i metodi che permettono il funzionamento LIFO della struttura dati. Il primo aggiunge una nuova descrizione di tempo, pertanto lo sviluppatore chiude con quella precedente e si occupa di configurarne un'altra, il secondo permette di tornare indietro di un passo e di intervenire nuovamente sulla descrizione precedente, cancellando però l'ultima inserita (e restituendola in uscita, come nelle più comuni regole di una pila).

```
Time getTime(int nTime)
RepeatTime[] getRepeatTimes(int nTime)
```

I tipici metodi *getters* che consentono di usufruire in uscita dei dati contenuti in una classe senza violare il principio di *information hiding*.
Il parametro in ingresso `nTime` consente allo sviluppatore di scegliere in quale

descrizione di tempo prelevare i dati di interesse del metodo.

`TimeDescription[] getTimeDescriptions()`, può sembrare un comune metodo *getter* (e lo è), ma al suo interno nasconde un'importanza più rilevante: questo è l'unico metodo che permette di interfacciare il TimeFactory con il SdpFactory, in quanto vengono restituite le descrizioni di tempo inserite ed elaborate nella struttura dati.
Lo sviluppatore potrà quindi realizzare un'intera serie di descrizioni di tempo invocando solamente i metodi di una singola istanza in maniera opportuna, infine tramite questa istruzione può permettere l'inserimento delle descrizioni appena costruite all'interno del messaggio SDP principale.

4.2 MediaFactory

La seconda classe include tutti i metodi necessari per generare, inserire e manipolare dati riguardanti una o più descrizioni di media.
La struttura interna prevede un altro `ArrayList` di tipo `MediaDescription` che, ad ogni istanza di oggetto, viene caricato con uno o due MediaDescription, a seconda del tipo che viene inserito nel costruttore: se esso è `video`, verrà definita una descrizione di media audio ed una di tipo video, altrimenti sarà disponibile solo il MediaDescription di tipo audio. Ovviamente l'insieme dei tipi di media può essere estesa a proprio piacimento, rimanendo sempre nei limiti del protocollo.
All'interno della classe sono presenti dei metodi privati in grado di snellire notevolmente il codice sorgente dei vari metodi pubblici definiti:

- `boolean isValidMedia(String media)`, individua se un media inserito in ingresso sia a norma con la documentazione del protocollo
- `int findMedia(String media)`, cerca un media, se presente, e restituisce l'indice corrispondente. Se non viene individuato, restituisce -1
- `boolean isContained(String attribute, String media)`, rivela se un tipo di attributo è presente in un dato media. Il metodo è indispensabile per la definizione di metodi riguardanti gli attributi, in quanto si possono filtrare istruzioni in cui un attributo, già inserito, non può essere nuovamente immesso

4.2.1 Metodi

```
void setConnection(String resource, String media)
void setConnection(String resource, int nAddress, int ttl, String media)
```

Immettono in un dato media un indirizzo Connection diverso (non necessariamente) da quello di default.
Il secondo metodo serve per inserire un gruppo multicast di indirizzi contigui, tanti quanti ne indica il parametro `nAddress` (maggiore di 1).

`void addBandwidth(String bandwidth, int value, String media)`, aggiunge un parametro Bandwidth in un dato media. All'interno del metodo non vengono effettuati controlli perché vengono già realizzati dalla libreria jSDP.

```
void addPacketTimeAttributes(String ptime, String maxptime)
void addOrientationAttribute(String orientation, String media)
void addLanguageAttributes (String sdplang, String lang, String media)
void addQualityAttributes (String framerate, String quality)
```

Tali metodi riguardano l'inserimento di attributi media, enunciati nella sezione riguardante il protocollo SDP, in un dato media.
I vantaggi di questa realizzazione sono:

- mancanza di inconsistenza (lo sviluppatore è vincolato ad inserire solamente degli attributi di tipo specifico)
- accorpamento delle istruzioni (alcuni attributi vengono trattati, per associazione di idee, in uno stesso unico metodo)

Nello specifico, il metodo `addPacketTimeAttributes` definisce il tempo di durata specifica (`ptime`) e massima (`maxptime`) di un pacchetto media, `addOrientationAttribute` indica la posizione del workspace sullo schermo, `addLanguageAttributes` tratta gli attributi riguardanti la lingua di un dato media, infine `addQualityAttributes` definisce il frame rate ed il grado di qualità del media video utilizzato nella sessione in corso.
I metodi `addPacketTimeAttributes` e `addQualityAttributes` non necessitano del parametro `media` in ingresso: il primo lavora solo su media di tipo audio, il secondo su video, pertanto la classe valuterà automaticamente la situazione.

`void setModeAttribute(String mode, String media)`, serve ad impostare, in un dato media, l'attributo `mode` in una delle modalità descritte nella documentazione del protocollo.
Il valore predefinito è settato a `sendrecv`.

`void addMediaDescription(String media, int port, String proto, String fmt)`, inserisce una nuova descrizione di media nella struttura dati.

```
Connection getConnection(String media) throws SDPException
Bandwith[] getBandwidth(String media)
Attribute[] getAttributes(String media) throws SDPException
```

Metodi *getters* che restituiscono, rispettivamente, i campi Connection, Bandwidth e Attribute in un dato media in ingresso.
Nel caso di Bandwidth e Attribute, che sono degli array, restituiscono **tutti** i campi di quel tipo aggiunti all'interno del dato media.

`MediaDescription[] getMediaDescriptions() throws SDPException`, restituisce un array contenente tutte le descrizioni di media incluse nella struttura dati.
Come nel TimeFactory, questo metodo è fondamentale per interfacciarsi alla descrizione di sessione.

4.3 SdpFactory

Il vero corpo del PariPari-VoIP SDP, la classe fondamentale per la realizzazione del messaggio finale da immettere nei pacchetti, realizza una descrizione di sessione.
Lo sviluppatore può scegliere, a seconda delle sue esigenze, se far lavorare in parallelo le tre classi oppure se è sufficiente soltanto configurare quest'ultima, in quanto solo questa è la classe che fornisce i dati fondamentali e minimi per poter parlare di messaggio SDP.
Anche se non definita alcuna descrizione di tempo (obbligatoria, come enunciato nella documentazione), `SdpFactory` fornisce una descrizione minima che rappresenta una sessione perpetua, cioè il tempo di partenza e di arresto sono pari a zero e non esiste alcun parametro RepeatTime. Sta allo sviluppatore, se necessario, configurare in maniera corretta il proprio oggetto `TimeFactory` ed aggiungere tutte le descrizioni di tempo realizzate nell'oggetto `SdpFactory`.
Il funzionamento di base riguarda anche le descrizioni di media, che dovranno prima essere realizzate con `MediaFactory`.
Può sembrare una manovra complessa, ma in realtà un elementare esempio chiarificatore al termine della sezione permetterà di comprendere quanto sia immediato realizzare un messaggio SDP anche molto ricco di parametri in poche righe di codice.

4.3.1 Metodi

`void setOrigin(TransactionMessage message) throws SDPException`, configura i parametri del campo Origin sfruttando in ingresso un parametro di tipo `TransactionMessage`.
All'interno del metodo viene richiamata una funzione che, dal campo `username`, elimina tutti gli eventuali spazi, in modo da non violare la definizione del protocollo.
L'inserimento dei parametri `sessionID` e `sessionVersion` viene lasciato alla libreria jSDP.
Come per il metodo `setConnection` della classe `MediaFactory`, `setOrigin` sfrutta al suo interno un parametro `message` di tipo `TransactionMessage`, che consiste in un messaggio SIP già costruito (può essere una richiesta o una risposta), da inviare attraverso il flusso di trasmissione. Al suo interno contiene informazioni utili che possono essere sfruttate anche per completare alcuni parametri specifici di determinati campi SDP: in particolare si può individuare l'indirizzo IP di chi invia il messaggio e chi lo riceve.

`void setSessionNameInformation(String sessionName, String information) throws SDPException`, imposta, molto semplicemente, nome e informazioni sul-

la descrizione di sessione.

```
void setConnection(String resource) throws SDPException
void setConnection(String resource, int nAddress, int ttl) throws SDPException
void setURI(String URI) throws SDPException
```

Impostano i campi Connection ed URI della descrizione di sessione. Si noti che il campo Connection contenuto in questa classe non ha alcun legame con quello impostato in `MediaFactory` (anche se, molto spesso, i due valori coincidono).

```
void addEmail(String email) throws SDPException
void addPhone(String phone) throws SDPException
```

Aggiungono alla descrizione di sessione un indirizzo E-Mail o un numero di telefono.

`void addBandwidth(String bandwidth, int value) throws SDPException`, aggiunge alla descrizione di sessione una larghezza di banda.

```
void addCategoryAttributes(String cat, String keywds, String tool)
throws SDPException
void addTypeConferenceAttribute(String conferenceType) throws SDPException
void addCharSetAttribute(String characterSet) throws SDPException
void addLanguageAttributes(String sdplang, String lang) throws SDPException
```

Metodi che permettono l'inserimento di attributi previsti dalla documentazione riguardanti la descrizione di sessione. Anche in questo caso non esiste alcun legame con i metodi di configurazione di attributi elencati in `MediaFactory`.
Nello specifico, `addCategoryAttributes` definisce informazioni riguardanti la categoria ed il tool utilizzato per realizzare la descrizione di sessione, `addTypeConferenceAttribute` specifica il tipo di conferenza, `addCharSetAttribute` specifica il set di caratteri utilizzato per visualizzare il nome della sessione e le informazioni testuali che la riguardano, mentre `addLanguageAttributes`, analogamente alla classe `MediaFactory`, tratta gli attributi riguardanti la lingua della sessione in corso.

`void setModeAttribute(String mode) throws SDPException`, imposta l'attributo `mode` della descrizione di sessione (e quindi anche in questa circostanza non viene toccata alcuna descrizione di media).
Anche se nella documentazione vengono fatte alcune piccole distinzioni sulle modalità di funzionamento a seconda che siano collocate nella sessione o nel media, si sceglie di non porre altri ulteriori vincoli e di lasciare il controllo allo sviluppatore.

```
void setMediaDescription(MediaDescription[] mediaDescriptions) throws
IllegalArgumentException, SDPException
void setTimeDescription(TimeDescription[] timeDescriptions)
```

Metodi che permettono l'inserimento di tutte le descrizioni di tempo e di media realizzate e configurate nelle altre classi precedentemente descritte.

```
Origin getOrigin() throws SDPException
SessionName getSessionName() throws SDPException
Information getSessionInformation() throws SDPException
Uri getUri()
Connection getConnection()
Email[] getEmails()
Phone[] getPhones()
Bandwith[] getBandwidths()
Attribute[] getAttributes()
```

Metodi *getters* che restituiscono tutti i parametri utili definiti nella sessione di descrizione.

`SessionDescription getSessionDescription() throws SDPException`, metodo che restituisce l'intera descrizione di sessione realizzata, corredata con i vari campi, facoltativi e non, e descrizioni di tempo e media.
Ogni applicazione che sfrutta PariPari-VoIP SDP per realizzare un messaggio SDP dovrebbe terminare sempre con questa istruzione.

4.4 Esempio d'uso

Si vuole realizzare il seguente messaggio SDP:

```
v=0
o=howard 3496726834 3496726834 IN IP4 66.249.71.227
s=VoIP call
i=prova di sessione
b=CT:1024
t=3498720300 3498723964
a=sendrecv
m=audio 12200 RTP/AVP 0
c=IN IP4 66.249.71.227
a=sendrecv
a=sdplang:english
a=lang:english
m=video 12200 RTP/AVP 0
c=IN IP4 66.249.71.227
```

```
a=sendonly
```

Esso rappresenta una situazione di prova di sessione effettuata dall'utente howard dotato di indirizzo IPv4 66.249.71.227 che desidera realizzare una videochiamata (presenza dei media `audio` e `video`) il giorno 14 novembre 2010 alle ore 10.45, fino alle ore 11.45 circa, in lingua inglese, sulla porta 12200 e sfruttando il protocollo di trasporto RTP/AVP. Egli può sostenere una larghezza di banda totale di 1024 kbps, può ricevere ed inviare la sua voce ma non può ricevere il flusso video inviato dal destinatario (invia solamente).

Supponendo che si faccia riferimento ad un parametro `message` di tipo `TransactionMessage` (di cui non viene riportata la definizione), il frammento di codice sorgente in grado di realizzare questo corposo messaggio si traduce in quanto segue:

```
TimeFactory timeFactory=new TimeFactory();
MediaFactory mediaFactory=new MediaFactory(message, “video“, “RTP/AVP“,
“0“);
SdpFactory sdpFactory=new SdpFactory();
timeFactory.timeSetting(3498720300, 3498723964);
mediaFactory.setModeAttribute(“sendonly“, “video“);
mediaFactory.addLanguageAttributes(“english“, “english“, “audio“);
sdpFactory.setOrigin(message);
sdpFactory.setSessionNameInformation(“VoIP call“, “prova di sessione“);
sdpFactory.addBandwidth(“CT“, 1024);
sdpFactory.setTimeDescription(timeFactory.getTimeDescriptions());
sdpFactory.setMediaDescription(mediaFactory.getMediaDescriptions());
```

Da notare come un messaggio SDP di 16 righe sia stato realizzato in 12 righe di codice.
Il grande punto di forza di questo sistema sta nella semplicità di utilizzo: con tre oggetti è possibile controllare efficacemente (e con possibilità minime di immettere parametri errati) tutti i numerosi campi di cui uno sviluppatore può avere bisogno per realizzare un messaggio a norma con le regole dettate dal protocollo.

Capitolo 5

Applicazione della libreria

Lo sviluppo di PariPari-VoIP SDP consente all'utente di realizzare un messaggio anche piuttosto complesso rispettando le regole del protocollo SDP; in particolare i dati in ingresso nei vari campi sono interamente controllati a monte, preservando completamente il rischio di immettere nel flusso di trasmissione parametri inconsistenti. Inoltre i metodi implementati possono manipolare in maniera specifica tutti i campi e le descrizioni previste dal protocollo (escludendo il campo Encryption, ignorato completamente a causa delle raccomandazioni della documentazione a non utilizzare, per il momento, tale parametro), con tutte le possibilità e limitazioni previste: è stato spiegato, per esempio, come il campo Attribute non sia definito in maniera fumosa e generica come in jSDP (dove era possibile introdurre qualsiasi tipo di attributo, anche inesistente, lasciando forse eccessiva libertà allo sviluppatore), ma esistono metodi in grado di settare tipologie specifiche di attributi, implementando anche particolari opzioni in caso di necessità. Con questa scelta progettuale si ottiene anche, come diretta conseguenza, un notevole risparmio di codice, come esplicato nel precedente capitolo (e nell'appendice 2, contenente altri esempi d'uso della libreria), in quanto molti metodi *automatizzano* i processi più comuni e ripetitivi, come la creazione di un campo e la sua immissione nella descrizione presa in considerazione.
In conclusione quindi, lo sviluppatore che utilizza PariPari-VoIP SDP si ritrova di fronte ad una struttura molto più guidata di jSDP, limitando la consultazione della documentazione del protocollo al minimo indispensabile.
Nonostante i molteplici punti di forza che la libreria presenta, ci sono ancora dei dettagli da sistemare riguardanti principalmente la leggibilità del sorgente all'interno delle classi del modulo VoIP, oltre a voler gettare le basi in un ambizioso progetto di *standardizzazione* dei messaggi SDP secondo le esigenze attuali e dell'immediato futuro per quanto riguarda il VoIP di PariPari.

5.1 Problema della portabilità

Un esempio chiarificatore consentirà di spiegare più facilmente il problema in questione. Si desideri realizzare un semplice messaggio SDP:

```
v=0
o=howard 3496726834 3496726834 IN IP4 87.0.178.252
s=VoIP call
i=prova di sessione numero 2
t=0 0
a=sendrecv
c=IN IP4 87.0.178.252
m=audio 15000 RTP/AVP 31
a=sendrecv
```

che rappresenta il solito utente howard che vuole stabilire una comunicazione audio perpetua (il campo Time include entrambi i parametri a zero) in sessione e ricezione, su porta 15000, sfruttando il protocollo di trasporto RTP/AVP.
Il codice sorgente da compilare per realizzare questo messaggio è il seguente:

```
MediaFactory mediaFactory=new MediaFactory(message, ‘‘audio‘‘, ‘‘RTP/AVP‘‘,
‘‘31‘‘);
SdpFactory sdpFactory=new SdpFactory();
sdpFactory.setOrigin(message);
sdpFactory.setSessionNameInformation(‘‘VoIP call‘‘, ‘‘prova di sessione
numero 2‘‘);
sdpFactory.setTimeDescription(timeFactory.getTimeDescriptions());
sdpFactory.setMediaDescription(mediaFactory.getMediaDescriptions());
```

in cui `message` consiste nel parametro di tipo `TransactionMessage`, enunciato nel capitolo 4.
Così com'è il codice è funzionante e compatto, oltre ad essere molto leggibile e di intuitiva comprensione. Si suppone però di voler introdurre, finalmente, questo messaggio SDP all'interno di un messaggio SIP di richiesta nel modulo VoIP di PariPari dove, nello specifico, il metodo che si occupa di costruirlo (all'interno della classe `SipManager`) è costituito dalla seguente firma:

```
public void sendRequest(TransactionRequest request) throws IOException,
SDPException
```

e dal seguente codice:

```
SipRequest sipRequest = factory.createRequest(request.method, request.toUser.getContactURI(),
request.number,new SipUser(request.toUser), new SipUser(request.fromUser),
new SipContact(request.fromUser), request.callid,null);
factory.sendMessage(sipRequest, request.toUser.getSipIp());
```

Dove il metodo `createRequest`, contenuto nella classe `SipFactory`, realizza una richiesta SIP che verrà successivamente inviata al destinatario tramite il metodo `sendMessage`.
È importante sapere che, tra i parametri immessi in ingresso nel metodo `createRequest`, l'ultimo riguarda il campo `body` del messaggio SIP. Si ricordi che il corpo di un messaggio SIP corrisponde ad una descrizione di sessione realizzata secondo il protocollo SDP, pertanto è sufficiente immettere (in formato array `byte`) tale messaggio come parametro `body` del metodo.
Nello specifico, in base all'esempio preso in considerazione, il codice corrispondente è:

```
MediaFactory mediaFactory=new MediaFactory(message, "audio", "RTP/AVP", "31");
SdpFactory sdpFactory=new SdpFactory();
sdpFactory.setOrigin(message);
sdpFactory.setSessionNameInformation("VoIP call", "prova di sessione numero 2");
sdpFactory.setMediaDescription(mediaFactory.getMediaDescriptions());
SessionDescription description=sdpFactory.getSessionDescription();
SipRequest sipRequest = factory.createRequest(request.method, request.toUser.getContactURI(),
request.number, new SipUser(request.toUser), new SipUser(request.fromUser),
new SipContact(request.fromUser), request.callid, description.toString().getBytes());
factory.sendMessage(sipRequest, request.toUser.getSipIp());
```

Già con una descrizione di sessione molto essenziale il codice sorgente del metodo inizia a diventare più prolisso e meno facilmente leggibile, con messaggi più complessi si rischia di appesantire notevolmente la classe. Inoltre si dovrebbero realizzare descrizioni *ad-hoc*: in sintesi ogni qualvolta che si desidera introdurre un nuovo campo o attributo all'interno della sessione, lo sviluppatore è obbligato a mettere mano continuamente al sorgente per effettuare anche piccole ma significative modifiche all'interno della classe.
Si dice che il prodotto non è *portabile*, significa che è poco adatto ad estendersi o adattarsi ai cambiamenti di esigenza degli utenti.

5.2 SdpManager

L'ultimo passo verso l'implementazione del protocollo SDP all'interno del modulo VoIP di PariPari riguarda la realizzazione di una classe contenente dei modelli di costruzione di messaggi standard; l'obiettivo è di avere una serie di metodi al cui interno si svolge la creazione di un particolare tipo di messaggio: se per esempio si fosse interessati a costruire un messaggio video con determinati parametri, è sufficiente invocare solamente uno specifico metodo della classe in questione.
Considerato che, allo stato attuale dello sviluppo, il modulo VoIP di PariPari consente solamente una trasmissione audio punto-punto, o anche multicast, i campi di SDP effettivamente utili da implementare non sono numerosi. Al momento quindi esiste un unico metodo:

```
public SessionDescription sdpAudioCreatorStandard() throws SDPException
```

il cui risultato finale consiste in una descrizione di sessione di questo tipo:

```
v=0
o=howard 3496726834 3496726834 IN IP4 87.0.178.252
s=VoIP Call
i=PariPari audio calling
t=0 0
u=http://www.paripari.it
a=sendrecv
a=charset:ISO-8859-1
m=audio 5060 RTP/AVP 0
c=IN IP4 87.0.178.252
a=sendrecv
a=sdplang:English
a=lang:undefined
```

realizzato dal seguente sorgente:

```
mediaFactory.addLanguageAttributes(“English“, “undefined“, “audio“);
sdpFactory.setSessionNameInformation(“VoIP Call“, “PariPari audio calling“);
sdpFactory.setURI(“http://www.paripari.it“);
sdpFactory.addCharSetAttribute(“ISO-8859-1“);
sdpFactory.setTimeDescription(timeFactory.getTimeDescriptions());
sdpFactory.setMediaDescription(mediaFactory.getMediaDescriptions());
return sdpFactory.getSessionDescription();
```

inoltre sono presenti tre metodi *getters*:

```
public SdpFactory<S> getSdpFactory() throws SDPException
public TimeFactory<S> getTimeFactory() throws SDPException
public MediaFactory<S> getMediaFactory() throws SDPException
```

che restituiscono in uscita le tre principali descrizioni trattate nel capitolo 4. La scelta è motivata dal fatto che, per quanto si desideri ottenere una libreria in grado di costruire messaggi predefiniti, in certe particolari applicazioni può essere utile poter accedere direttamente alle descrizioni di sessione, media e tempo per modellarle personalmente in base alle proprie esigenze.
Si ottiene dunque uno strumento che consente di realizzare al volo un messaggio SDP senza troppe complicazioni, mentre gli sviluppatori più smaliziati possono intervenire tranquillamente su singoli aspetti della descrizione di sessione nel momento in cui i metodi contenuti nella classe non rispettassero le loro esigen-

ze.
Nello specifico, il codice del metodo `sendRequest` trattato precedentemente si riduce a queste semplici istruzioni:

```
SdpManager<S> manager=new SdpManager<S>(request);
SipRequest sipRequest = factory.createRequest(request.method, request.toUser.getContactURI(),
request.number, new SipUser(request.toUser), new SipUser(request.fromUser),
new SipContact(request.fromUser), request.callid,
manager.sdpAudioCreatorStandard().toString().getBytes());
factory.sendMessage(sipRequest, request.toUser.getSipIp());
```

Capitolo 6

Sviluppi futuri

Il protocollo SDP è piuttosto recente e in fase di sviluppo continuo, ma nonostante questo viene già tranquillamente sfruttato in numerose applicazioni multimediali, principalmente in parallelo con il protocollo SIP precedentemente citato.
Il campo principale a cui gli sviluppatori preme maggiormente realizzare un aggiornamento valido è indubbiamente il campo Key (`k=`), creato funzionante, ma sconsigliato dalla documentazione stessa del protocollo: i metodi di criptaggio infatti sono discretamente efficaci solo nel caso in cui il messaggio SDP venga trasportato in un canale sicuro e protetto, cosa che non è sempre possibile garantire in una comunicazione di tipo VoIP.
In ogni caso gli sviluppatori del protocollo sono al lavoro su un sistema più efficiente e robusto di scambio delle chiavi, in modo che le nuove applicazioni che lo sfruttano possano utilizzare serenamente il campo Key per garantire un trasporto sicuro dei dati, senza alcun timore di attacchi crittografici.

Per quanto riguarda una visione più generale, attualmente il protocollo SDP viene utilizzato non solo per la descrizione di sessioni multimediali, ma anche per quanto riguarda la descrizione di capacità di un sistema.
Non essendo assolutamente pensato per questo tipo di applicazioni, il protocollo SDP viene sottoposto a vari adattamenti, tagli ed approssimazioni, fino a realizzare dei veri e propri protocolli a parte, che però complicano non poco la struttura di un messaggio.
Questo *trend* può essere abbattuto in futuro con la definizione di un nuovo protocollo: **Session Description and Capability Negotiation** (SDPng). Le intenzioni degli sviluppatori di questo formato riguardano la risoluzione di questo problema di *bricolage* nella struttura di un messaggio SDP, offrendo la possibilità di definire le informazioni a riguardo mediante dei tag XML, permettendo una maggiore estendibilità dei dati. In questa maniera è possibile sfruttare lo stesso protocollo sia per sessioni multimediali vere e proprie, sia per la definizione di altri parametri necessari per un determinato tipo di comunicazione non necessariamente multimediale.
Il protocollo è ancora in fase di definizione e, pur essendo già disponibili delle prime corpose *Memo* in rete[1], non esiste ancora una RFC su cui fare riferimento

[1] http://www-rn.informatik.uni-bremen.de/ietf/mmusic/sdp-ng/drafts/draft-ietf-mmusic-sdpng-req-01.txt

per una eventuale applicazione significativa del protocollo.

PariPari-VoIP SDP può essere considerata una libreria piuttosto completa per quanto riguarda la realizzazione di un messaggio SDP, manca solamente la possibilità di introdurre i campi Key e TimeZone: il primo è sconsigliato dalla documentazione stessa, il secondo è stato al momento trascurato per scelta progettuale (non si ritiene importante, al momento, pensare alla gestione di una sessione multimediale con una differenza di fuso orario). Inoltre mancano metodi riguardanti gli attributi `a=rtpmap` e `a=fmtp`, in quanto la struttura attuale di jSDP non ne permette la creazione in maniera agevole (si dovrebbe realizzare una classe a parte solo per questo obiettivo, ma per il momento non sono attributi significativi per le attuali potenzialità del modulo VoIP).
L'interfacciamento con il modulo è invece ancora piuttosto limitato per le possibilità offerte dalla libreria: la classe `SdpManager` contiene solamente un metodo che, di fatto, realizza un unico messaggio vero e proprio che andrà in circolo nel flusso di pacchetti scambiati durante la sessione. Il punto di forza di questa libreria risiede nella sua espandibilità: tale classe, una volta implementate nuove funzionalità ed opzioni all'interno del VoIP di PariPari (la videochiamata, su tutte), potrà essere ampliata con altri modelli di creazione di messaggi, in modo da sfruttare in maniera più completa un numero decisamente superiore di campi e configurazioni del messaggio da inviare.

In conclusione, il progetto in questione è indubbiamente rilevante per rendere completo e moderno l'attuale sistema di chiamata VoIP, rimanendo al passo con i software a disposizione sul mercato che, ormai, utilizzano tutti comunemente i due protocolli SIP+SDP (nonostante quest'ultimo sia ancora incompleto e "giovane", tutte le opzioni implementate e collaudate sono funzionanti e tranquillamente sfruttabili). Questa libreria sarà però importante soprattutto nel prossimo futuro di PariPari, quando nuove opzioni motiveranno la realizzazione di descrizioni di sessione più complessi, quindi uno sfruttamento maggiormente consistente di tutte le potenzialità offerte.

Appendice 1: Interfaccia SDPFactory

```
Attribute createAttribute(String name)

Attribute createAttribute(String name, String value)

Bandwith createBandwith(String modifier, int value)

Connection createConnection(String resource)

Email createEmail(String info)

Information createInformation(String value)

Key createKey(String method, String key)

Media createMedia(String media, int port, int portsCount, String protocol,
String format)

Media createMedia(String media, int port, String protocol, String format)

Origin createOrigin()

Origin createOrigin(long sessionVersion, String address)

Origin createOrigin(String address)

Origin createOrigin(String user, long sessionID, long sessionVersion,
String address)

Origin createOrigin(String user, long sessionVersion, String address)

Phone createPhone(String number)

RepeatTime createRepeatTime(long repeatInterval, long activeDuration,
long offset)

RepeatTime createRepeatTime(long repeatInterval, long activeDuration,
```

```
long[] offsets)

SessionDescription createSessionDescription()

SessionName createSessionName()

SessionName createSessionName(String value)

Time createTime()

Time createTime(Date start, Date stop)

Time createTime(long start, long stop)

TimeZone createTimeZone(Date time, long offset)

Uri createUri(String url)

Uri createUri(URL url)

Version createVersion()

void encode(SessionDescription sd, OutputStream stream)

void encode(SessionDescription sd, String encoding, OutputStream stream)

SessionDescription parseSessionDescription(File file)

SessionDescription parseSessionDescription(InputStream stream)

SessionDescription parseSessionDescription(String input)

SessionDescription parseSessionDescription(String[] fields)
```

Appendice 2: RFC4566

Il testo che segue è un estratto della documentazione originale RFC4566, capitoli 5 e 6, riguardante la descrizione dei vari campi definiti dal protocollo SDP.

Requirements and Recommendations

The purpose of SDP is to convey information about media streams in multimedia sessions to allow the recipients of a session description to participate in the session. SDP is primarily intended for use in an internetwork, although it is sufficiently general that it can describe conferences in other network environments. Media streams can be many-to-many. Sessions need not be continually active.

Thus far, multicast-based sessions on the Internet have differed from many other forms of conferencing in that anyone receiving the traffic can join the session (unless the session traffic is encrypted). In such an environment, SDP serves two primary purposes. It is a means to communicate the existence of a session, and it is a means to convey sufficient information to enable joining and participating in the session. In a unicast environment, only the latter purpose is likely to be relevant.

An SDP session description includes the following:

- Session name and purpose
- Time(s) the session is active
- The media comprising the session
- Information needed to receive those media (addresses, ports, formats, etc.)

As resources necessary to participate in a session may be limited, some additional information may also be desirable:

- Information about the bandwidth to be used by the session
- Contact information for the person responsible for the session

In general, SDP must convey sufficient information to enable applications to join a session (with the possible exception of encryption keys) and to announce the resources to be used to any non-participants that may need to know. (This latter feature is primarily useful when SDP is used with a multicast session announcement protocol.)

Media and Transport Information

An SDP session description includes the following media information:

- The type of media (video, audio, etc.)
- The transport protocol (RTP/UDP/IP, H.320, etc.)
- The format of the media (H.261 video, MPEG video, etc.)

In addition to media format and transport protocol, SDP conveys address and port details. For an IP multicast session, these comprise:

- The multicast group address for media
- The transport port for media

This address and port are the destination address and destination port of the multicast stream, whether being sent, received, or both.

For unicast IP sessions, the following are conveyed:

- The remote address for media
- The remote transport port for media

The semantics of this address and port depend on the media and transport protocol defined. By default, this SHOULD be the remote address and remote port to which data is sent. Some media types may redefine this behaviour, but this is NOT RECOMMENDED since it complicates implementations (including middleboxes that must parse the addresses to open Network Address Translation (NAT) or firewall pinholes).

Timing Information

Sessions may be either bounded or unbounded in time. Whether or not they are bounded, they may be only active at specific times. SDP can convey:

- An arbitrary list of start and stop times bounding the session
- For each bound, repeat times such as every Wednesday at 10am for one hour

This timing information is globally consistent, irrespective of local time zone or daylight saving time.

Private Sessions

It is possible to create both public sessions and private sessions. SDP itself does not distinguish between these; private sessions are typically conveyed by encrypting the session description during distribution. The details of how encryption is performed are dependent on the mechanism used to convey SDP; mechanisms are currently defined for SDP transported using SAP [14] and SIP

[15], and others may be defined in the future.

If a session announcement is private, it is possible to use that private announcement to convey encryption keys necessary to decode each of the media in a conference, including enough information to know which encryption scheme is used for each media.

Obtaining Further Information about a Session

A session description should convey enough information to decide whether or not to participate in a session. SDP may include additional pointers in the form of Uniform Resource Identifiers (URIs) for more information about the session.

Categorisation

When many session descriptions are being distributed by SAP, or any other advertisement mechanism, it may be desirable to filter session announcements that are of interest from those that are not. SDP supports a categorisation mechanism for sessions that is capable of being automated (the `a=cat:` attribute).

Internationalisation

The SDP specification recommends the use of the ISO 10646 character sets in the UTF-8 encoding [5] to allow many different languages to be represented. However, to assist in compact representations, SDP also allows other character sets such as ISO 8859-1 to be used when desired. Internationalisation only applies to free-text fields (session name and background information), and not to SDP as a whole.

SDP Specification

An SDP session description is denoted by the media type application/sdp.

An SDP session description is entirely textual using the ISO 10646 character set in UTF-8 encoding. SDP field names and attribute names use only the US-ASCII subset of UTF-8, but textual fields and attribute values MAY use the full ISO 10646 character set. Field and attribute values that use the full UTF-8 character set are never directly compared, hence there is no requirement for UTF-8 normalisation. The textual form, as opposed to a binary encoding such as ASN.1 or XDR, was chosen to enhance portability, to enable a variety of transports to be used, and to allow flexible, text-based toolkits to be used to generate and process session descriptions. However, since SDP may be used in environments where the maximum permissible size of a session description is limited, the encoding is deliberately compact. Also, since announcements may be transported via very unreliable means or damaged by an intermediate caching server, the encoding was designed with strict order and formatting rules so that most errors would result in malformed session announcements that could be detected easily and discarded. This also allows rapid discarding of encrypted

session announcements for which a receiver does not have the correct key.

An SDP session description consists of a number of lines of text of the form:

```
<type>=<value>
```

where <type> MUST be exactly one case-significant character and <value> is structured text whose format depends on <type>. In general, <value> is either a number of fields delimited by a single space character or a free format string, and is case-significant unless a specific field defines otherwise. Whitespace MUST NOT be used on either side of the = sign.

An SDP session description consists of a session-level section followed by zero or more media-level sections. The session-level part starts with a v= line and continues to the first media-level section. Each media-level section starts with an m= line and continues to the next media-level section or end of the whole session description. In general, session-level values are the default for all media unless overridden by an equivalent media-level value.

Some lines in each description are REQUIRED and some are OPTIONAL, but all MUST appear in exactly the order given here (the fixed order greatly enhances error detection and allows for a simple parser). OPTIONAL items are marked with a *.

Session description:
v= (protocol version)
o= (originator and session identifier)
s= (session name)
i=* (session information)
u=* (URI of description)
e=* (email address)
p=* (phone number)
c=* (connection information – not required if included in all media)
b=* (zero or more bandwidth information lines)
One or more time descriptions (t= and r= lines; see below)
z=* (time zone adjustments)
k=* (encryption key)
a=* (zero or more session attribute lines)
Zero or more media descriptions

Time description:
t= (time the session is active)
r=* (zero or more repeat times)

Media description, if present:
m= (media name and transport address)
i=* (media title)
c=* (connection information – optional if included at session level)
b=* (zero or more bandwidth information lines)

k=* (encryption key)
a=* (zero or more media attribute lines)

The set of type letters is deliberately small and not intended to be extensible – an SDP parser MUST completely ignore any session description that contains a type letter that it does not understand. The attribute mechanism (a= described below) is the primary means for extending SDP and tailoring it to particular applications or media. Some attributes have a defined meaning, but others may be added on an application-, media-, or session-specific basis. An SDP parser MUST ignore any attribute it doesn't understand.

An SDP session description may contain URIs that reference external content in the u=, k=, and a= lines. These URIs may be dereferenced in some cases, making the session description non-self- contained.
The connection (c=) and attribute (a=) information in the session-level section applies to all the media of that session unless overridden by connection information or an attribute of the same name in the media description. For instance, in the example below, each media behaves as if it were given a recvonly attribute.

An example SDP description is:

```
v=0
o=jdoe 2890844526 2890842807 IN IP4 10.47.16.5
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.example.com/seminars/sdp.pdf
e=j.doe@example.com (Jane Doe)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
m=audio 49170 RTP/AVP 0
m=video 51372 RTP/AVP 99
a=rtpmap:99 h263-1998/90000
```

Text fields such as the session name and information are octet strings that may contain any octet with the exceptions of 0x00 (Nul), 0x0a (ASCII newline), and 0x0d (ASCII carriage return). The sequence CRLF (0x0d0a) is used to end a record, although parsers SHOULD be tolerant and also accept records terminated with a single newline character. If the `a=charset` attribute is not present, these octet strings MUST be interpreted as containing ISO-10646 characters in UTF-8 encoding (the presence of the `a=charset` attribute may force some fields to be interpreted differently).

A session description can contain domain names in the o=, u=, e=, c=, and a= lines. Any domain name used in SDP MUST comply with [1], [2]. Internationalised domain names (IDNs) MUST be represented using the ASCII Compatible Encoding (ACE) form defined in [11] and MUST NOT be directly represented in UTF-8 or any other encoding (this requirement is for compatibility with RFC 2327 and other SDP-related standards, which predate the development of

internationalised domain names).

Protocol Version (v=)

```
v=0
```

The v= field gives the version of the Session Description Protocol. This memo defines version 0. There is no minor version number.

Origin (o=)

```
o=<username> <sess-id> <sess-version> <nettype> <addrtype> <unicast-address>
```

The o= field gives the originator of the session (her username and the address of the user's host) plus a session identifier and version number:

`<username>` is the user's login on the originating host, or it is - if the originating host does not support the concept of user IDs. The `<username>` MUST NOT contain spaces.

`<sess-id>` is a numeric string such that the tuple of `<username>`, `<sess-id>`, `<nettype>`, `<addrtype>`, and `<unicast-address>` forms a globally unique identifier for the session. The method of `<sess-id>` allocation is up to the creating tool, but it has been suggested that a Network Time Protocol (NTP) format timestamp be used to ensure uniqueness [13].

`<sess-version>` is a version number for this session description. Its usage is up to the creating tool, so long as `<sess-version>` is increased when a modification is made to the session data. Again, it is RECOMMENDED that an NTP format timestamp is used.

`<nettype>` is a text string giving the type of network. Initially IN is defined to have the meaning Internet, but other values MAY be registered in the future.

`<addrtype>` is a text string giving the type of the address that follows. Initially IP4 and IP6 are defined, but other values MAY be registered in the future.

`<unicast-address>` is the address of the machine from which the session was created. For an address type of IP4, this is either the fully qualified domain name of the machine or the dotted- decimal representation of the IP version 4 address of the machine. For an address type of IP6, this is either the fully qualified domain name of the machine or the compressed textual representation of the IP version 6 address of the machine. For both IP4 and IP6, the fully qualified domain name is the form that SHOULD be given unless this is unavailable, in which case the globally unique address MAY be substituted. A local IP address MUST NOT be used in any context where the SDP description might leave the scope in which the address is meaningful (for example, a local address MUST NOT be included in an application-level referral that might leave the scope).

In general, the o= field serves as a globally unique identifier for this version of this session description, and the subfields excepting the version taken together identify the session irrespective of any modifications.

For privacy reasons, it is sometimes desirable to obfuscate the username and IP address of the session originator. If this is a concern, an arbitrary `<username>` and private `<unicast-address>` MAY be chosen to populate the o= field, provided that these are selected in a manner that does not affect the global uniqueness of the field.

Session Name (s=)

`s=<session name>`

The s= field is the textual session name. There MUST be one and only one s= field per session description. The s= field MUST NOT be empty and SHOULD contain ISO 10646 characters (but see also the `a=charset` attribute). If a session has no meaningful name, the value s= SHOULD be used (i.e., a single space as the session name).

Session Information (i=)

`i=<session description>`

The i= field provides textual information about the session. There MUST be at most one session-level i= field per session description, and at most one i= field per media. If the `a=charset` attribute is present, it specifies the character set used in the i= field. If the `a=charset` attribute is not present, the i= field MUST contain ISO 10646 characters in UTF-8 encoding.

A single i= field MAY also be used for each media definition. In media definitions, i= fields are primarily intended for labelling media streams. As such, they are most likely to be useful when a single session has more than one distinct media stream of the same media type. An example would be two different whiteboards, one for slides and one for feedback and questions.

The i= field is intended to provide a free-form human-readable description of the session or the purpose of a media stream. It is not suitable for parsing by automata.

URI (u=)

`u=<uri>`

A URI is a Uniform Resource Identifier as used by WWW clients [7]. The URI should be a pointer to additional information about the session. This field is OPTIONAL, but if it is present it MUST be specified before the first media field. No more than one URI field is allowed per session description.

Email Address and Phone Number (e= and p=)

```
e=<email-address>
p=<phone-number>
```

The e= and p= lines specify contact information for the person responsible for the conference. This is not necessarily the same person that created the conference announcement.

Inclusion of an email address or phone number is OPTIONAL. Note that the previous version of SDP specified that either an email field or a phone field MUST be specified, but this was widely ignored. The change brings the specification into line with common usage.

If an email address or phone number is present, it MUST be specified before the first media field. More than one email or phone field can be given for a session description.

Phone numbers SHOULD be given in the form of an international public telecommunication number (see ITU-T Recommendation E.164) preceded by a +. Spaces and hyphens may be used to split up a phone field to aid readability if desired. For example:

```
p=+1 617 555-6011
```

Both email addresses and phone numbers can have an OPTIONAL free text string associated with them, normally giving the name of the person who may be contacted. This MUST be enclosed in parentheses if it is present. For example:

```
e=j.doe@example.com (Jane Doe)
```

The alternative RFC 2822 [29] name quoting convention is also allowed for both email addresses and phone numbers. For example:

```
e=Jane Doe <j.doe@example.com>
```

The free text string SHOULD be in the ISO-10646 character set with UTF-8 encoding, or alternatively in ISO-8859-1 or other encodings if the appropriate session-level a=charset attribute is set.

Connection Data (c=)

```
c=<nettype> <addrtype> <connection-address>
```

The c= field contains connection data.

A session description MUST contain either at least one c= field in each media description or a single c= field at the session level. It MAY contain a single session-level c= field and additional c= field(s) per media description, in which

case the per-media values override the session-level settings for the respective media.

The first sub-field (<nettype>) is the network type, which is a text string giving the type of network. Initially, IN is defined to have the meaning Internet, but other values MAY be registered in the future.

The second sub-field (<addrtype>) is the address type. This allows SDP to be used for sessions that are not IP based. This memo only defines IP4 and IP6, but other values MAY be registered in the future.

The third sub-field (<connection-address>) is the connection address. OPTIONAL sub-fields MAY be added after the connection address depending on the value of the <addrtype> field.

When the <addrtype> is IP4 and IP6, the connection address is defined as follows:

- If the session is multicast, the connection address will be an IP multicast group address. If the session is not multicast, then the connection address contains the unicast IP address of the expected data source or data relay or data sink as determined by additional attribute fields. It is not expected that unicast addresses will be given in a session description that is communicated by a multicast announcement, though this is not prohibited.
- Sessions using an IPv4 multicast connection address MUST also have a time to live (TTL) value present in addition to the multicast address. The TTL and the address together define the scope with which multicast packets sent in this conference will be sent. TTL values MUST be in the range 0-255. Although the TTL MUST be specified, its use to scope multicast traffic is deprecated; applications SHOULD use an administratively scoped address instead.

The TTL for the session is appended to the address using a slash as a separator. An example is:

```
c=IN IP4 224.2.36.42/127
```

IPv6 multicast does not use TTL scoping, and hence the TTL value MUST NOT be present for IPv6 multicast. It is expected that IPv6 scoped addresses will be used to limit the scope of conferences.

Hierarchical or layered encoding schemes are data streams where the encoding from a single media source is split into a number of layers. The receiver can choose the desired quality (and hence bandwidth) by only subscribing to a subset of these layers. Such layered encodings are normally transmitted in multiple multicast groups to allow multicast pruning. This technique keeps unwanted traffic from sites only requiring certain levels of the hierarchy. For applications requiring multiple multicast groups, we allow the following notation to be used

for the connection address:

```
<base multicast address>[/<ttl>]/<number of addresses>
```

If the number of addresses is not given, it is assumed to be one. Multicast addresses so assigned are contiguously allocated above the base address, so that, for example:

```
c=IN IP4 224.2.1.1/127/3
```

would state that addresses 224.2.1.1, 224.2.1.2, and 224.2.1.3 are to be used at a TTL of 127. This is semantically identical to including multiple c= lines in a media description:

```
c=IN IP4 224.2.1.1/127
c=IN IP4 224.2.1.2/127
c=IN IP4 224.2.1.3/127
```

Similarly, an IPv6 example would be:

```
c=IN IP6 FF15::101/3
```

which is semantically equivalent to:

```
c=IN IP6 FF15::101
c=IN IP6 FF15::102
c=IN IP6 FF15::103
```

(remembering that the TTL field is not present in IPv6 multicast).

Multiple addresses or c= lines MAY be specified on a per-media basis only if they provide multicast addresses for different layers in a hierarchical or layered encoding scheme. They MUST NOT be specified for a session-level c= field.

The slash notation for multiple addresses described above MUST NOT be used for IP unicast addresses.

Bandwidth (b=)

```
b=<bwtype>:<bandwidth>
```

This OPTIONAL field denotes the proposed bandwidth to be used by the session or media. The `<bwtype>` is an alphanumeric modifier giving the meaning of the `<bandwidth>` figure. Two values are defined in this specification, but other values MAY be registered in the future:

- `CT`, If the bandwidth of a session or media in a session is different from the bandwidth implicit from the scope, a `b=CT:...` line SHOULD be supplied for the session giving the proposed upper limit to the bandwidth used (the conference total bandwidth). The primary purpose of this is to

give an approximate idea as to whether two or more sessions can coexist simultaneously. When using the CT modifier with RTP, if several RTP sessions are part of the conference, the conference total refers to total bandwidth of all RTP sessions.

- `AS`, The bandwidth is interpreted to be application specific (it will be the application's concept of maximum bandwidth). Normally, this will coincide with what is set on the application's maximum bandwidth control if applicable. For RTP-based applications, AS gives the RTP session bandwidth

Note that CT gives a total bandwidth figure for all the media at all sites. AS gives a bandwidth figure for a single media at a single site, although there may be many sites sending simultaneously.

A prefix X- is defined for `<bwtype>` names. This is intended for experimental purposes only. For example:

```
b=X-YZ:128
```

Use of the X- prefix is NOT RECOMMENDED: instead new modifiers SHOULD be registered with IANA in the standard namespace. SDP parsers MUST ignore bandwidth fields with unknown modifiers. Modifiers MUST be alphanumeric and, although no length limit is given, it is recommended that they be short.

The `<bandwidth>` is interpreted as kilobits per second by default. The definition of a new `<bwtype>` modifier MAY specify that the bandwidth is to be interpreted in some alternative unit (the CT and AS modifiers defined in this memo use the default units).

Timing (t=)

```
t=<start-time> <stop-time>
```

The t= lines specify the start and stop times for a session. Multiple t= lines MAY be used if a session is active at multiple irregularly spaced times; each additional t= line specifies an additional period of time for which the session will be active. If the session is active at regular times, an r= line (see below) should be used in addition to, and following, a t= line – in which case the t= line specifies the start and stop times of the repeat sequence.

The first and second sub-fields give the start and stop times, respectively, for the session. These values are the decimal representation of Network Time Protocol (NTP) time values in seconds since 1900 [13]. To convert these values to UNIX time, subtract decimal 2208988800.

NTP timestamps are elsewhere represented by 64-bit values, which wrap sometime in the year 2036. Since SDP uses an arbitrary length decimal representation, this should not cause an issue (SDP timestamps MUST continue counting

seconds since 1900, NTP will use the value modulo the 64-bit limit).

If the `<stop-time>` is set to zero, then the session is not bounded, though it will not become active until after the `<start-time>`. If the `<start-time>` is also zero, the session is regarded as permanent.

User interfaces SHOULD strongly discourage the creation of unbounded and permanent sessions as they give no information about when the session is actually going to terminate, and so make scheduling difficult.

The general assumption may be made, when displaying unbounded sessions that have not timed out to the user, that an unbounded session will only be active until half an hour from the current time or the session start time, whichever is the later. If behaviour other than this is required, an end-time SHOULD be given and modified as appropriate when new information becomes available about when the session should really end.

Permanent sessions may be shown to the user as never being active unless there are associated repeat times that state precisely when the session will be active.

Repeat Times (r=)

`r=<repeat interval> <active duration> <offsets from start-time>`

r= fields specify repeat times for a session. For example, if a session is active at 10am on Monday and 11am on Tuesday for one hour each week for three months, then the `<start-time>` in the corresponding t= field would be the NTP representation of 10am on the first Monday, the `<repeat interval>` would be 1 week, the `<active duration>` would be 1 hour, and the offsets would be zero and 25 hours. The corresponding t= field stop time would be the NTP representation of the end of the last session three months later. By default, all fields are in seconds, so the r= and t= fields might be the following:

```
t=3034423619 3042462419
r=604800 3600 0 90000
```

To make description more compact, times may also be given in units of days, hours, or minutes. The syntax for these is a number immediately followed by a single case-sensitive character. Fractional units are not allowed – a smaller unit should be used instead. The following unit specification characters are allowed:

d - days (86400 seconds)
h - hours (3600 seconds)
m - minutes (60 seconds)
s - seconds (allowed for completeness)

Thus, the above session announcement could also have been written:

```
r=7d 1h 0 25h
```

Monthly and yearly repeats cannot be directly specified with a single SDP repeat time; instead, separate t= fields should be used to explicitly list the session times.

Time Zones (z=)

```
z=<adjustment time> <offset> <adjustment time> <offset> ....
```

To schedule a repeated session that spans a change from daylight saving time to standard time or vice versa, it is necessary to specify offsets from the base time. This is required because different time zones change time at different times of day, different countries change to or from daylight saving time on different dates, and some countries do not have daylight saving time at all.

Thus, in order to schedule a session that is at the same time winter and summer, it must be possible to specify unambiguously by whose time zone a session is scheduled. To simplify this task for receivers, we allow the sender to specify the NTP time that a time zone adjustment happens and the offset from the time when the session was first scheduled. The z= field allows the sender to specify a list of these adjustment times and offsets from the base time.

An example might be the following:

```
z=2882844526 -1h 2898848070 0
```

This specifies that at time 2882844526, the time base by which the session's repeat times are calculated is shifted back by 1 hour, and that at time 2898848070, the session's original time base is restored. Adjustments are always relative to the specified start time – they are not cumulative. Adjustments apply to all t= and r= lines in a session description.

If a session is likely to last several years, it is expected that the session announcement will be modified periodically rather than transmit several years' worth of adjustments in one session announcement.

Encryption Keys (k=)

```
k=<method>
k=<method>:<encryption key>
```

If transported over a secure and trusted channel, the Session Description Protocol MAY be used to convey encryption keys. A simple mechanism for key exchange is provided by the key field (k=), although this is primarily supported for compatibility with older implementations and its use is NOT RECOMMENDED. Work is in progress to define new key exchange mechanisms for use with SDP [27] [28], and it is expected that new applications will use those mechanisms.

A key field is permitted before the first media entry (in which case it applies

to all media in the session), or for each media entry as required. The format of keys and their usage are outside the scope of this document, and the key field provides no way to indicate the encryption algorithm to be used, key type, or other information about the key: this is assumed to be provided by the higher-level protocol using SDP. If there is a need to convey this information within SDP, the extensions mentioned previously SHOULD be used. Many security protocols require two keys: one for confidentiality, another for integrity. This specification does not support transfer of two keys.

The method indicates the mechanism to be used to obtain a usable key by external means, or from the encoded encryption key given. The following methods are defined:

- `k=clear:<encryption key>`, The encryption key is included untransformed in this key field. This method MUST NOT be used unless it can be guaranteed that the SDP is conveyed over a secure channel. The encryption key is interpreted as text according to the charset attribute; use the `k=base64:` method to convey characters that are otherwise prohibited in SDP.

- `k=base64:<encoded encryption key>`, The encryption key is included in this key field but has been base64 encoded [12] because it includes characters that are prohibited in SDP. This method MUST NOT be used unless it can be guaranteed that the SDP is conveyed over a secure channel.

- `k=uri:<URI to obtain key>`, A Uniform Resource Identifier is included in the key field. The URI refers to the data containing the key, and may require additional authentication before the key can be returned. When a request is made to the given URI, the reply should specify the encoding for the key. The URI is often an Secure Socket Layer/Transport Layer Security (SSL/TLS)-protected HTTP URI (https:), although this is not required.

- `k=prompt`, No key is included in this SDP description, but the session or media stream referred to by this key field is encrypted. The user should be prompted for the key when attempting to join the session, and this user-supplied key should then be used to decrypt the media streams. The use of user-specified keys is NOT RECOMMENDED, since such keys tend to have weak security properties.

The key field MUST NOT be used unless it can be guaranteed that the SDP is conveyed over a secure and trusted channel. An example of such a channel might be SDP embedded inside an S/MIME message or a TLS-protected HTTP session. It is important to ensure that the secure channel is with the party that is authorised to join the session, not an intermediary: if a caching proxy server is used, it is important to ensure that the proxy is either trusted or unable to access the SDP.

Attributes (a=)

```
a=<attribute>
a=<attribute>:<value>
```

Attributes are the primary means for extending SDP. Attributes may be defined to be used as session-level attributes, media-level attributes, or both.

A media description may have any number of attributes (a= fields) that are media specific. These are referred to as media-level attributes and add information about the media stream. Attribute fields can also be added before the first media field; these session-level attributes convey additional information that applies to the conference as a whole rather than to individual media.

Attribute fields may be of two forms:

- A property attribute is simply of the form `a=<flag>`. These are binary attributes, and the presence of the attribute conveys that the attribute is a property of the session. An example might be `a=recvonly`.
- A value attribute is of the form `a=<attribute>:<value>`. For example, a whiteboard could have the value attribute `a=orient: landscape`

Attribute interpretation depends on the media tool being invoked. Thus receivers of session descriptions should be configurable in their interpretation of session descriptions in general and of attributes in particular.

Attribute names MUST use the US-ASCII subset of ISO-10646/UTF-8. Attribute values are octet strings, and MAY use any octet value except 0x00 (Nul), 0x0A (LF), and 0x0D (CR). By default, attribute values are to be interpreted as in ISO-10646 character set with UTF-8 encoding. Unlike other text fields, attribute values are NOT normally affected by the charset attribute as this would make comparisons against known values problematic. However, when an attribute is defined, it can be defined to be charset dependent, in which case its value should be interpreted in the session charset rather than in ISO-10646.

Attributes MUST be registered with IANA. If an attribute is received that is not understood, it MUST be ignored by the receiver.

Media Descriptions (m=)

`m=<media> <port> <proto> <fmt> ...`

A session description may contain a number of media descriptions. Each media description starts with an m= field and is terminated by either the next m= field or by the end of the session description. A media field has several sub-fields:

`<media>` is the media type. Currently defined media are audio, video, text, application, and message, although this list may be extended in the future.

`<port>` is the transport port to which the media stream is sent. The meaning of the transport port depends on the network being used as specified in the relevant c= field, and on the transport protocol defined in the `<proto>` subfield of the media field. Other ports used by the media application (such as the RTP Control Protocol (RTCP) port [19]) MAY be derived algorithmically from

the base media port or MAY be specified in a separate attribute (for example, `a=rtcp:` as defined in [22]).

If non-contiguous ports are used or if they don't follow the parity rule of even RTP ports and odd RTCP ports, the `a=rtcp:` attribute MUST be used. Applications that are requested to send media to a `<port>` that is odd and where the `a=rtcp:` is present MUST NOT subtract 1 from the RTP port: that is, they MUST send the RTP to the port indicated in `<port>` and send the RTCP to the port indicated in the `a=rtcp` attribute.

For applications where hierarchically encoded streams are being sent to a unicast address, it may be necessary to specify multiple transport ports. This is done using a similar notation to that used for IP multicast addresses in the c= field:

```
m=<media> <port>/<number of ports> <proto> <fmt> ...
```

In such a case, the ports used depend on the transport protocol. For RTP, the default is that only the even-numbered ports are used for data with the corresponding one-higher odd ports used for the RTCP belonging to the RTP session, and the `<number of ports>` denoting the number of RTP sessions. For example:

```
m=video 49170/2 RTP/AVP 31
```

would specify that ports 49170 and 49171 form one RTP/RTCP pair and 49172 and 49173 form the second RTP/RTCP pair. RTP/AVP is the transport protocol and 31 is the format (see below). If non- contiguous ports are required, they must be signalled using a separate attribute (for example, `a=rtcp:` as defined in [22]).

If multiple addresses are specified in the c= field and multiple ports are specified in the m= field, a one-to-one mapping from port to the corresponding address is implied. For example:

```
c=IN IP4 224.2.1.1/127/2
m=video 49170/2 RTP/AVP 31
```

would imply that address 224.2.1.1 is used with ports 49170 and 49171, and address 224.2.1.2 is used with ports 49172 and 49173.

The semantics of multiple m= lines using the same transport address are undefined. This implies that, unlike limited past practice, there is no implicit grouping defined by such means and an explicit grouping framework (for example, [18]) should instead be used to express the intended semantics.

`<proto>` is the transport protocol. The meaning of the transport protocol is dependent on the address type field in the relevant c= field. Thus a c= field of IP4 indicates that the transport protocol runs over IP4. The following transport protocols are defined, but may be extended through registration of new

protocols with IANA:

- udp: denotes an unspecified protocol running over UDP.
- RTP/AVP: denotes RTP [19] used under the RTP Profile for Audio and Video Conferences with Minimal Control [20] running over UDP.
- RTP/SAVP: denotes the Secure Real-time Transport Protocol [23] running over UDP.

The main reason to specify the transport protocol in addition to the media format is that the same standard media formats may be carried over different transport protocols even when the network protocol is the same – a historical example is vat Pulse Code Modulation (PCM) audio and RTP PCM audio; another might be TCP/RTP PCM audio. In addition, relays and monitoring tools that are transport-protocol-specific but format-independent are possible.

`<fmt>` is a media format description. The fourth and any subsequent sub-fields describe the format of the media. The interpretation of the media format depends on the value of the `<proto>` sub-field.

If the `<proto>` sub-field is RTP/AVP or RTP/SAVP the `<fmt>` sub-fields contain RTP payload type numbers. When a list of payload type numbers is given, this implies that all of these payload formats MAY be used in the session, but the first of these formats SHOULD be used as the default format for the session. For dynamic payload type assignments the `a=rtpmap:` attribute SHOULD be used to map from an RTP payload type number to a media encoding name that identifies the payload format. The `a=fmtp:` attribute MAY be used to specify format parameters.

If the `<proto>` sub-field is udp the `<fmt>` sub-fields MUST reference a media type describing the format under the audio, video, text, application, or message top-level media types. The media type registration SHOULD define the packet format for use with UDP transport.

For media using other transport protocols, the `<fmt>` field is protocol specific. Rules for interpretation of the `<fmt>` sub- field MUST be defined when registering new protocols.

SDP Attributes

The following attributes are defined. Since application writers may add new attributes as they are required, this list is not exhaustive.

`a=cat:<category>`

This attribute gives the dot-separated hierarchical category of the session. This is to enable a receiver to filter unwanted sessions by category. There is no central registry of categories. It is a session-level attribute, and it is not dependent on charset.

`a=keywds:<keywords>`

Like the cat attribute, this is to assist identifying wanted sessions at the receiver. This allows a receiver to select interesting session based on keywords describing the purpose of the session; there is no central registry of keywords. It is a session-level attribute. It is a charset-dependent attribute, meaning that its value should be interpreted in the charset specified for the session description if one is specified, or by default in ISO 10646/UTF-8.

`a=tool:<name and version of tool>`

This gives the name and version number of the tool used to create the session description. It is a session-level attribute, and it is not dependent on charset.

`a=ptime:<packet time>`

This gives the length of time in milliseconds represented by the media in a packet. This is probably only meaningful for audio data, but may be used with other media types if it makes sense. It should not be necessary to know ptime to decode RTP or vat audio, and it is intended as a recommendation for the encoding/packetisation of audio. It is a media-level attribute, and it is not dependent on charset.

`a=maxptime:<maximum packet time>`

This gives the maximum amount of media that can be encapsulated in each packet, expressed as time in milliseconds. The time SHALL be calculated as the sum of the time the media present in the packet represents. For frame-based codecs, the time SHOULD be an integer multiple of the frame size. This attribute is probably only meaningful for audio data, but may be used with other media types if it makes sense. It is a media-level attribute, and it is not dependent on charset. Note that this attribute was introduced after RFC 2327, and non-updated implementations will ignore this attribute.

`a=rtpmap:<payload type> <encoding name>/<clock rate> [/<encoding parameters>]`

This attribute maps from an RTP payload type number (as used in an m= line) to an encoding name denoting the payload format to be used. It also provides information on the clock rate and encoding parameters. It is a media-level attribute that is not dependent on charset.

Although an RTP profile may make static assignments of payload type numbers to payload formats, it is more common for that assignment to be done dynamically using `a=rtpmap:` attributes. As an example of a static payload type, consider u-law PCM coded single-channel audio sampled at 8 kHz. This is completely defined in the RTP Audio/Video profile as payload type 0, so there is no need for an `a=rtpmap:` attribute, and the media for such a stream sent to UDP port 49232 can be specified as:

`m=audio 49232 RTP/AVP 0`
An example of a dynamic payload type is 16-bit linear encoded stereo audio sampled at 16 kHz. If we wish to use the dynamic RTP/AVP payload type 98 for this stream, additional information is required to decode it:

```
m=audio 49232 RTP/AVP 98
a=rtpmap:98 L16/16000/2
```

Up to one rtpmap attribute can be defined for each media format specified. Thus, we might have the following:

```
m=audio 49230 RTP/AVP 96 97 98
a=rtpmap:96 L8/8000
a=rtpmap:97 L16/8000
a=rtpmap:98 L16/11025/2
```

RTP profiles that specify the use of dynamic payload types MUST define the set of valid encoding names and/or a means to register encoding names if that profile is to be used with SDP. The RTP/AVP and RTP/SAVP profiles use media subtypes for encoding names, under the top-level media type denoted in the m= line. In the example above, the media types are audio/l8 and audio/l16.

For audio streams, `<encoding parameters>` indicates the number of audio channels. This parameter is OPTIONAL and may be omitted if the number of channels is one, provided that no additional parameters are needed.

For video streams, no encoding parameters are currently specified.

Additional encoding parameters MAY be defined in the future, but codec-specific parameters SHOULD NOT be added. Parameters added to an `a=rtpmap:` attribute SHOULD only be those required for a session directory to make the choice of appropriate media to participate in a session. Codec-specific parameters should be added in other attributes (for example, `a=fmtp:`).

Note: RTP audio formats typically do not include information about the number of samples per packet. If a non-default (as defined in the RTP Audio/Video Profile) packetisation is required, the ptime attribute is used as given above.

`a=recvonly`

This specifies that the tools should be started in receive-only mode where applicable. It can be either a session- or media- level attribute, and it is not dependent on charset. Note that recvonly applies to the media only, not to any associated control protocol (e.g., an RTP-based system in recvonly mode SHOULD still send RTCP packets).

`a=sendrecv`

This specifies that the tools should be started in send and receive mode. This is necessary for interactive conferences with tools that default to receive-only

mode. It can be either a session or media-level attribute, and it is not dependent on charset.

If none of the attributes sendonly, recvonly, inactive, and sendrecv is present, sendrecv SHOULD be assumed as the default for sessions that are not of the conference type broadcast or H332 (see below).

`a=sendonly`

This specifies that the tools should be started in send-only mode. An example may be where a different unicast address is to be used for a traffic destination than for a traffic source. In such a case, two media descriptions may be used, one sendonly and one recvonly. It can be either a session- or media-level attribute, but would normally only be used as a media attribute. It is not dependent on charset. Note that sendonly applies only to the media, and any associated control protocol (e.g., RTCP) SHOULD still be received and processed as normal.

`a=inactive`

This specifies that the tools should be started in inactive mode. This is necessary for interactive conferences where users can put other users on hold. No media is sent over an inactive media stream. Note that an RTP-based system SHOULD still send RTCP, even if started inactive. It can be either a session or media-level attribute, and it is not dependent on charset.

`a=orient:<orientation>`

Normally this is only used for a whiteboard or presentation tool. It specifies the orientation of a the workspace on the screen. It is a media-level attribute. Permitted values are `portrait`, `landscape`, and `seascape` (upside-down landscape). It is not dependent on charset.

`a=type:<conference type>`

This specifies the type of the conference. Suggested values are `broadcast`, `meeting`, `moderated`, `test`, and `H332`. `recvonly` should be the default for `type:broadcast` sessions, `type:meeting` should imply `sendrecv`, and `type:moderated` should indicate the use of a floor control tool and that the media tools are started so as to mute new sites joining the conference.

Specifying the attribute `type:H332` indicates that this loosely coupled session is part of an H.332 session as defined in the ITU H.332 specification [26]. Media tools should be started `recvonly`.

Specifying the attribute `type:test` is suggested as a hint that, unless explicitly requested otherwise, receivers can safely avoid displaying this session description to users.

The type attribute is a session-level attribute, and it is not dependent on charset.

`a=charset:<character set>`

This specifies the character set to be used to display the session name and information data. By default, the ISO-10646 character set in UTF-8 encoding is used. If a more compact representation is required, other character sets may be used. For example, the ISO 8859-1 is specified with the following SDP attribute:

`a=charset:ISO-8859-1`

This is a session-level attribute and is not dependent on charset. The charset specified MUST be one of those registered with IANA, such as ISO-8859-1. The character set identifier is a US-ASCII string and MUST be compared against the IANA identifiers using a case-insensitive comparison. If the identifier is not recognised or not supported, all strings that are affected by it SHOULD be regarded as octet strings.

Note that a character set specified MUST still prohibit the use of bytes 0x00 (Nul), 0x0A (LF), and 0x0d (CR). Character sets requiring the use of these characters MUST define a quoting mechanism that prevents these bytes from appearing within text fields.

`a=sdplang:<language tag>`

This can be a session-level attribute or a media-level attribute. As a session-level attribute, it specifies the language for the session description. As a media-level attribute, it specifies the language for any media-level SDP information field associated with that media. Multiple sdplang attributes can be provided either at session or media level if multiple languages in the session description or media use multiple languages, in which case the order of the attributes indicates the order of importance of the various languages in the session or media from most important to least important.

In general, sending session descriptions consisting of multiple languages is discouraged. Instead, multiple descriptions SHOULD be sent describing the session, one in each language. However, this is not possible with all transport mechanisms, and so multiple sdplang attributes are allowed although NOT RECOMMENDED.

The sdplang attribute value must be a single RFC 3066 language tag in US-ASCII [9]. It is not dependent on the charset attribute. An sdplang attribute SHOULD be specified when a session is of sufficient scope to cross geographic boundaries where the language of recipients cannot be assumed, or where the session is in a different language from the locally assumed norm.

`a=lang:<language tag>`

This can be a session-level attribute or a media-level attribute. As a session-level attribute, it specifies the default language for the session being described. As a media- level attribute, it specifies the language for that media, overriding any session-level language specified. Multiple lang attributes can be provided

either at session or media level if the session description or media use multiple languages, in which case the order of the attributes indicates the order of importance of the various languages in the session or media from most important to least important.

The lang attribute value must be a single RFC 3066 language tag in US-ASCII [9]. It is not dependent on the charset attribute. A lang attribute SHOULD be specified when a session is of sufficient scope to cross geographic boundaries where the language of recipients cannot be assumed, or where the session is in a different language from the locally assumed norm.

`a=framerate:<frame rate>`

This gives the maximum video frame rate in frames/sec. It is intended as a recommendation for the encoding of video data. Decimal representations of fractional values using the notation `<integer>.<fraction>` are allowed. It is a media-level attribute, defined only for video media, and it is not dependent on charset.

`a=quality:<quality>`

This gives a suggestion for the quality of the encoding as an integer value. The intention of the quality attribute for video is to specify a non-default trade-off between frame-rate and still-image quality. For video, the value is in the range 0 to 10, with the following suggested meaning:

- 10 - the best still-image quality the compression scheme can give.
- 5 - the default behaviour given no quality suggestion.
- 0 - the worst still-image quality the codec designer thinks is still usable.

It is a media-level attribute, and it is not dependent on charset.

`a=fmtp:<format> <format specific parameters>`

This attribute allows parameters that are specific to a particular format to be conveyed in a way that SDP does not have to understand them. The format must be one of the formats specified for the media. Format-specific parameters may be any set of parameters required to be conveyed by SDP and given unchanged to the media tool that will use this format. At most one instance of this attribute is allowed for each format.

It is a media-level attribute, and it is not dependent on charset.

Appendice 3: Ulteriori esempi d'uso della libreria

1) Si vuole realizzare un messaggio SDP, legato ad un messaggio SIP di tipo INVITE, in cui l'utente TizioIncognito (indirizzo IP 66.249.66.134) desidera stabilire una videochiamata punto-punto con l'utente Homer (IP 87.2.179.201), per raccontargli una divertente barzelletta appena ascoltata in un canale televisivo. Egli può ricevere ed inviare entrambi i segnali audio e video in maniera perpetua, mettendo a disposizione per tutta la conferenza una banda di 4096 kbps massimi. I due media agiscono sulle porte 16384 e 16385, sfruttando un protocollo di trasporto RTP/AVP, fmt 31.
La qualità video della sessione è quantificabile, con un intero da 0 a 10, da un 7 (e frame rate di 40 frame/sec), mentre la lingua utilizzata dal mittente è l'inglese nella descrizione di sessione, l'italiano nel parlato.
Dato che il destinatario è americano e non comprende la lingua italiana, la sessione sarà costituita anche da sottotitoli in lingua inglese (sfruttando la porta 20461 ed il protocollo di trasporto RTP/AVP) che traducono quanto viene detto dal mittente.

Innanzitutto è necessario creare un'istanza della richiesta (`message`) che si vuole realizzare. Essa deve contenere, tra le altre cose, il nickname dell'utente, le porte dove agiscono i flussi di dati (16384 e 16385 nel caso specifico) ed il suo indirizzo IP, fondamentali per costruire il campo Origin. In seguito è sufficiente ottenere la descrizione di sessione con il seguente frammento di codice:

```
MediaFactory<S> mediaFactory=new MediaFactory<S>(message, "video",
"RTP/AVP", "31");
SDPFactory<S> sdpFactory=new SdpFactory<S>();
sdpFactory.setOrigin(message);
sdpFactory.setSessionNameInformation("Video Calling", "I would like
to tell you a funny joke¡');
sdpFactory.addBandwidth("CT", 4096);
mediaFactory.addMediaDescription("text", 20461, "RTP/AVP", "31");
mediaFactory.addLanguageAttributes("english", "italian", "audio");
mediaFactory.addQualityAttributes("40", "7");
mediaFactory.setModeAttribute("sendonly", "text");
mediaFactory.addLanguageAttributes("english", "english", "text");
```

```
sdpFactory.setMediaDescriptions(mediaFactory.getMediaDescriptions());
sdpFactory.getSessionDescription();
```

Il risultato finale è costituito dal seguente messaggio:

```
v=0
o=3276982320 3276982320 IN IP4 66.249.66.134
s=Video Calling
i=I would like to tell you a funny joke!
b=CT:4096
t=0 0
a=sendrecv
m=audio 16384 RTP/AVP 31
c=IN IP4 66.249.66.134
a=sendrecv
a=sdplang:english
a=lang:italian
m=video 16385 RTP/AVP 31
c=IN IP4 66.249.66.134
a=sendrecv
a=framerate:40
a=quality:7
m=text 20461 RTP/AVP 31
c=IN IP4 66.249.66.134
a=sendonly
a=sdplang:english
a=lang:english
```

2) L'utente Homer riceve il messaggio, lo analizza e si prepara ad inviare la sua risposta. Egli può stabilire la comunicazione, a patto che vengano rispettate alcune determinate condizioni:

- la banda disponibile per tutta la conferenza è di 1024 kbps, non gli è consentito trasmettere video, quindi riceverà solamente le immagini del mittente,
- il mittente conosce l'inglese, quindi può comprendere la risposta di Homer, pertanto non gli è necessario trasmettere sottotitoli,
- la qualità delle immagini è peggiore, all'incirca 20 frame/sec e voto stabilito pari a 5

Il messaggio che invia insieme alla risposta SIP è realizzato nel modo seguente:

```
MediaFactory<S> mediaFactory=new MediaFactory<S>(message, “video“,
“RTP/AVP“, “31“);
SDPFactory<S> sdpFactory=new SdpFactory<S>();
```

```
sdpFactory.setOrigin(message);
sdpFactory.setSessionNameInformation(“Video Calling“, “I would like
to tell you a funny joke¡‘);
sdpFactory.addBandwidth(“CT“, 1024);
mediaFactory.addMediaDescription(“text“, 20461, “RTP/AVP“, “31“);
mediaFactory.addLanguageAttributes(“english“, “english“, “audio“);
mediaFactory.addQualityAttributes(“20“, “5“);
mediaFactory.setModeAttribute(“recvonly“, “text“);
mediaFactory.setModeAttribute(“recvonly“, “video“);
sdpFactory.setMediaDescriptions(mediaFactory.getMediaDescriptions());
sdpFactory.getSessionDescription();
```

Il risultato finale è costituito dal seguente messaggio:

```
v=0
o=3276982320 3276982320 IN IP4 66.249.66.134
s=Video Calling
i=I would like to tell you a funny joke!
b=CT:1024
t=0 0
a=sendrecv
m=audio 16384 RTP/AVP 31
c=IN IP4 87.2.179.201
a=sendrecv
a=sdplang:english
a=lang:english
m=video 16385 RTP/AVP 31
c=IN IP4 87.2.179.201
a=recvonly
a=framerate:20
a=quality:5
m=text 20461 RTP/AVP 31
c=IN IP4 87.2.179.201
a=recvonly
```

3) Si vuole realizzare un messaggio SDP in cui l'utente Simone Pepe desidera organizzare delle audioconferenze periodiche dove, puntata dopo puntata, enuncia ai suoi ascoltatori alcune regole e trucchi fondamentali per il gioco del calcio. In particolare, la cosa importante da segnalare nella descrizione del messaggio (oltre ai campi standard generati autonomamente dalla libreria) riguarda la gestione della tempistica della conferenza: le lezioni inizieranno mercoledì 1 dicembre 2010 e proseguiranno a cadenza settimanale, dalle 19.00 alle 20.00 fino a gennaio.

Innanzitutto è necessario, come di consueto, definire un'istanza delle principali classi Factory (questa volta verrà definito anche un oggetto `TimeFactory`):

```
TimeFactory<S> timeFactory=new TimeFactory<S>();
MediaFactory<S> mediaFactory=new MediaFactory<S>(message, “audio“,
“RTP/AVP“, “0“);
SDPFactory<S> sdpFactory=new SdpFactory<S>();
sdpFactory.setOrigin(message);
sdpFactory.setSessionNameInformation(“Lezione di calcio“, “ “);
```

La consegna non specifica altri campi o caratteristiche da segnalare, pertanto è possibile ora concentrarsi esclusivamente sulla definizione dei campi riguardanti il tempo.
Prima di tutto va analizzato il range di tempo in cui la sessione andrà in onda: proseguendo di mercoledì in mercoledì, si arriverà al 29 dicembre; la richiesta però viene interpretata come “il periodo di tempo andrà avanti fino a gennaio **compreso**“, pertanto si considera il mercoledì successivo al 29 dicembre, cioè il 5 gennaio 2011.
L'istruzione che traduce questo range è la seguente:

```
timeFactory.timeSetting(3500218800, 3503246400);
```

Rappresentando in secondi (a partire dal 1/1/1900) prima la data 1 dicembre 2010 ore 19.00, poi il 5 gennaio 2011 ore 20.00, cioè la fine del periodo delle sessioni.
Per rappresentare il periodo di tempo in cui si ripete la conferenza, è necessario ricorrere al campo RepeatTime: il primo parametro (`<repeat interval>`) indica ogni quanto tempo la sessione deve ripartire (nel caso specifico, l'equivalente di una settimana di tempo); il secondo parametro (`<active duration>`) rappresenta la durata in cui la sessione, ogni volta che comincia, deve rimanere attiva (quindi un'ora, in questo caso).
Il parametro `<offsets from start-time>` è fissato a 0, in quanto esiste un solo periodo di tempo da rappresentare.
Il risultato, in codice sorgente, è il seguente:

```
timeFactory.addRepeatTime(604800, 3600, 0);
```

Uno sviluppatore esterno, osservando questa istruzione, potrebbe storcere il naso nel ritrovarsi questi numeri apparentemente senza senso. Per rendere più elegante l'inserimento dei parametri, si può ricorrere alla classe `TypedTime` (importata direttamente dalla libreria jSDP), che consente di restituire staticamente il numero di secondi corrispondenti ad un giorno, un'ora, un minuto ed un secondo (quest'ultimo è più un “vezzo“).
A livello di codice, si può semplificare notevolmente la lettura in questo modo:

```
String repeatInterval="7"+TypedTime.DAYS;
String activeDuration=TypedTime.HOURS;
timeFactory.addRepeatTime(TypedTime.getTime(repeatInterval), TypedTime.getTime(activeDuration),
0);
```

Le prime due righe rappresentano, in maniera molto più intuitiva, periodi di tempo pari, rispettivamente, a 7 giorni (7d) e 1 ora (h). Nell'ultima riga, il metodo statico `getTime` consente di convertire in numero la stringa corrispondente che indica un certo arco di tempo, come nel modo visto in precedenza.

Nel caso in cui la conferenza ottenesse un discreto successo, la volontà dell'utente è di organizzare una nuova edizione della trasmissione, con diversi orari e cadenze periodiche.
Nel codice sorgente è sufficiente questa istruzione:

```
timeFactory.addNewTimeDescription();
```

per definire una nuova descrizione di tempo in riferimento alle nuove sessioni in programma dopo quelle indicate dalla descrizione precedente.

Bibliografia

1. M. Handley, V. Jacobson. *SDP: Session Description Protocol*, `http://tools.ietf.org/html/rfc4566` (2006).

2. Claudio Di Vita. *Progetto e realizzazione di una libreria per la gestione di un protocollo per le trasmissioni multicast*, `http://www.scribd.com/doc/14698/Progetto-e-realizzazione-di-una-libreria-per-la-gestione-di-un-protocollo-per-le-trasmissioni-multicast` (2004)

3. Documentazione di jSDP. `http://jsdp.sourceforge.net/api/net/sourceforge/jsdp/package-summary.html`

4. Elapsed Time Calculator. `http://www.mathcats.com/explore/elapsedtime.html`. Calcolatore in grado di riportare il numero di secondi trascorsi da una determinata data ad un'altra.