UNIVERSITÀ DEGLI STUDI DI MILANO
Facoltà di Scienze Matematiche, Fisiche e Naturali
Dipartimento di Informatica
Corso di Laurea Magistrale in Informatica per la Comunicazione

INTERFACCIAMENTO DI PERIFERICHE MIDI CON DISPOSITIVI MOBILI

Relatore: Dott. Luca A. LUDOVICO

Correlatore: Dott. Stefano BALDAN

Tesi di Laurea di:

Federico MARIN

Matricola: 786052

Anno Accademico 2012-2013

Indice

Elenco delle figure

Capitolo 1

Introduzione

Negli ultimi 50 anni lo sviluppo degli strumenti musicali è sempre più dipeso dalle tecnologie prima elettroniche e in seguito digitali.
In particolare è stato lo sviluppo dei primi strumenti elettronici a costruire le fondamenta per lo sviluppo di tecnologie musicali legate al mondo del digitale.
Se da un lato i primi sintetizzatori elettronici come Telharmonium, Theremin, Ondes Martenot, Trautonium ed Hammond, sono apparsi già nella prima metà del XX secolo, lo sviluppo dei primi strumenti digitali accessibili sul mercato è cominciato intorno agli anni 80.
La diffusione dei circuiti integrati nell'industria degli strumenti musicali, ha permesso di incorporare sofisticati dispositivi in grado di riprodurre suoni generati o registrati digitalmente, la cui unità base è costituita dal campione audio.
È stato questo il momento in cui la sintesi digitale ha soppiantato di fatto quella analogica; i sintetizzatori sono diventati macchine in grado di riprodurre le sonorità di tutti gli strumenti musicali, e non solo.
Il grande vantaggio delle tecnologie digitali utilizzate per le produzioni musicali è consistito nella gestione accurata dell'accordatura e dell'utilizzo contemporaneo di più oscillatori. La sincronizzazione di più oscillatori e la loro stabilità nel tempo sono stati infatti alcuni dei problemi principali nelle tecnologie analogiche per la produzione sonora.
Parallelamente alle migliorie appena descritte, i dispositivi digitali per la produzione sonora sintetica hanno anche permesso a questo tipo di tecnologie di diminuire le proprie dimensioni e di entrare così nel mercato *prosumer*.
Tuttavia, almeno inizialmente, queste conquiste digitali rendevano le sonorità di tali strumenti non paragonabili a quelle degli strumenti analogici, che, seppur privi della precisione matematica digitale risultavano all'orecchio umano più corposi nel suono per via della non perfetta sincronia dei componenti, come accade nei suoni prodotti in natura.
Oggigiorno le recenti tecniche di elaborazione del suono hanno sviluppato modelli capaci di emulare le leggi fisiche, rendendo di fatto anche la sintesi digitale capace di produrre sonorità altrettanto naturali e corpose all'orecchio umano; tuttavia esistono ancora in

commercio e nel mercato del vintage, sintetizzatori analogici molto apprezzati e richiesti, come ad esempio quelli della casa di produzione Moog.
Parallelamente allo sviluppo delle tecniche di sintesi del suono, le case produttrici hanno dovuto lavorare all'implementazione di metodologie per la trasmissione delle informazioni musicali per pilotare i sintetizzatori.
Il protocollo MIDI è diventato, di fatto, lo standard per la trasmissione di informazioni musicali per far dialogare tra loro differenti strumenti elettronici digitali.
Sebbene questo protocollo sia nato da più di 30 anni, continua ad esistere e ad essere implementato anche nei dispositivi musicali di fascia più alta come sintetizzatori, schede audio, *Digital Audio Workstation*[1] e software.
In particolare questo elaborato si concentra sull'analisi di come il protocollo MIDI sia oggi implementato nei moderni dispositivi mobili come smartphone e tablet e di come sia possibile sfruttarlo e adattarlo per l'utilizzo combinato con dispositivi MIDI.

Il lavoro è stato articolato in diverse fasi: nella prima fase è stato studiato lo stato dell'arte scientifico, descritto nel Capitolo 2, circa le metodologie e le soluzioni riguardanti l'utilizzo del protocollo MIDI sui dispositivi mobili, presenti sul mercato o proposte come prototipi; in seguito nel Capitolo 3 è presente una analisi delle tecnologie coinvolte nello sviluppo del prototipo software sperimentale per piattaforma Android per il collegamento di dispositivi MIDI tramite interfaccia USB, descritto nel Capitolo 4.
La motivazione principale che ha portato alla scelta di sperimentazione su questa piattaforma consiste nel fatto che al tempo di scrittura di questo elaborato sono già presenti diverse soluzioni di collegamento per apparecchiature MIDI con dispositivi mobili basati su iOS, grazie a componenti integrati al sistema operativo mobile, mentre non esistono ancora soluzioni native analoghe per sistemi Android.
In seguito allo sviluppo del *layer* di collegamento sono state implementate alcune metodologie di collegamento wireless per far dialogare il prototipo con più istanze dello stesso software o verso dispositivi come, per esempio, computer utilizzati per registrare e creare musica su *Digital Audio Workstation*.
In particolare si è cercato di fornire soluzioni che potessero rendere l'indirizzamento wireless dei messaggi MIDI compatibile con il più vasto numero di differenti tipologie di sistemi.

[1] Applicativo software progettato per la registrazione, l'editing e la riproduzione dell'audio digitale.

Nel Capitolo 5 sono presentati i test svolti e i risultati per la valutazione del prototipo proposto. In particolare, considerate le criticità dei collegamenti wireless in un contesto applicativo musicale, legate alla latenza intrinseca del mezzo trasmissivo, è stata svolta un'analisi, quantitativa e qualitativa, del tempo necessario per l'invio dei dati dal prototipo all'host ricevente, comparando la soluzione proposta con alcune tecnologie per lo scambio di messaggi MIDI via wireless presenti sul mercato.

Nel Capitolo 6 sono infine proposte alcune considerazioni finali sulla possibilità di estensione del prototipo implementato, per l'apporto di migliorie funzionali e per l'utilizzo delle tecnologie coinvolte in altri ambiti.

Capitolo 2

Stato dell'arte

Il protocollo MIDI, Musical Instrument Digital Interface, oggetto di analisi in questo lavoro, costituisce un linguaggio e una specifica che permette a più componenti hardware, software, strumenti musicali elettronici e computer di comunicare l'uno con l'altro attraverso una rete connessa. Il MIDI è utilizzato per tradurre parametri di performance, come quelli generati da uno strumento a tastiera o di diverso tipo, in equivalenti messaggi digitali che possono essere inviati ad altri dispositivi MIDI in grado di tradurli e utilizzarli per produzione sonora o ulteriore elaborazione. Le tecnologie software permettono oggigiorno anche di registrare un flusso di messaggi MIDI e di poterlo modificare in seguito, diventando così uno strumento di supporto per la creatività e la scrittura musicale correlata alle moderne evoluzioni della composizione.

Il protocollo MIDI nasce come esigenza per il controllo di più sintetizzatori in contemporanea. Le prime sincronizzazioni tra sintetizzatori erano ottenute grazie alla capacità di alcuni moduli di creare una corrente continua (DC) in grado di pilotare altri strumenti; in genere un'ottava musicale corrispondeva nell'incremento di 1 volt nella corrente e ogni semitono corrispondeva in 1/12 di volt. Questa soluzione si è rivelata sufficientemente stabile per pilotare in contemporanea più sintetizzatori monofonici; era inoltre necessario intervallare il segnale DC con un apposito segnale per rappresentare la fine di una nota, in modo sequenziale. Il successivo tentativo di sincronizzare più sintetizzatori polifonici evidenziò le carenze tecniche della modalità di controllo appena citata, fu così necessario studiare soluzioni differenti.

Dave Smith e Chet Wood, della casa di produzione "Sequential Circuits", oggi non più presente sul mercato, svilupparono un primo protocollo digitale per il controllo di strumenti elettronici, chiamato USI (Universal Synthesizer Interface). Una delle caratteristiche fondamentali di questo protocollo era la possibilità di essere implementato su prodotti di diversi produttori, rendendone possibile la comunicazione [12].

Il protocollo USI fu proposto alla fine del 1981 alla "Audio Engineering Society" durante la "70th AES Convention" [21]. Nei due anni successivi lo standard USI fu modificato e

adottato da AES con il nome di MIDI e con la prima specifica del protocollo, in seguito rivista e corretta fino all'ultima versione [32].

Sebbene il protocollo abbia compiuto recentemente 30 anni di vita, grazie al mantenimento delle caratteristiche principali e alla vasta diffusione nelle tecnologie musicali, il MIDI è diventato lo standard di comunicazione *de facto*, in ambito musicale, che difficilmente potrà essere sostituito in breve tempo, anche per la necessità di rendere compatibili le tecnologie sviluppate fino ad ora. Basti pensare che, grazie all'utilizzo del protocollo, è possibile mantenere nei nuovi prodotti la compatibilità con le prime soluzioni sviluppate con il supporto MIDI, come mostrato in figura 2.1.

Figura 2.1: Commodore 64 e iPad collegati via MIDI durante la fiera NAMM 2013 in celebrazione dei 30 anni del protocollo MIDI

2.1 Alternative al protocollo MIDI

Nonostante il protocollo MIDI sia stato adottato dalla totalità dei produttori di tecnologie audio, diverse critiche sono state mosse contro lo standard. Alcuni dei protocolli che si sono proposti in alternativa [20] al MIDI nel corso degli anni sono stati *mLAN*, *ZIPI*, *MaGIC* e *OSC*.

mLAN

mLan [42] è un protocollo open source sviluppato intorno all'anno 2000, con l'intenzione di portare il MIDI su connessione FireWire. Sebbene le prestazioni in termini di velocità siano maggiori, nell'utilizzo di questo protocollo è necessario interfacciamento con una porta in grado di moltiplicare le uscite FireWire per potere collegare altri dispositivi che utilizzano la stessa tecnologia. Nel 2008 molti produttori smisero di produrre dispositivi *mLAN* compatibili; soltanto qualche nuova macchina Yamaha presenta ancora questa funzionalità.

ZIPI

ZIPI [44], Zeta Instrument Processor Interface, fu introdotto dal "Department of Music" dell'università di Berkley in California. Introdotto come nuovo protocollo per il trasporto di informazioni musicali, il principale vantaggio rispetto al MIDI consisteva nella diversa tipologia di connessione tra differenti dispositivi. Anziché creare una catena seriale di dispositivi, come accade nella modalità MIDI *daisy chain*, in questo caso è utilizzato un *hub* centrale verso il quale ogni dispositivo che vuole partecipare al network deve essere collegato. Oltre ad una migliore gestione dei collegamenti, anche le specifiche del protocollo sono strutturate in modo da fornire prestazioni superiori al MIDI. La gestione dei canali, per esempio, avviene con 63 famiglie separate, ognuna delle quali presenta 127 strumenti in grado di riprodurre 127 differenti note, mentre nel protocollo MIDI i canali disponibili per ogni strumento sono soltanto 16, in grado di pilotare uno strumento da 127 note per ogni canale. Sebbene alcuni messaggi derivino direttamente dal protocollo MIDI, e in generale ZIPI apporti diverse migliorie, esso non è mai divenuto uno standard diffuso per la comunicazione dei metadati per le maggiori case produttrici di hardware musicale.

Le cause del mancato successo del protocollo *ZIPI* sono da ricondursi alla complessa, e quindi costosa, implementazione hardware che gli apparati necessitavano per interfacciarsi con tali specifiche e alla mancata compatibilità con il protocollo MIDI.

MaGIC

MaGIC [14], Media-Accelerated Global Information Carrier, è un protocollo basato sul trasporto bidirezionale su cavo ethernet di dati audio multicanale, informazioni di performance e corrente necessaria ad alimentare strumenti, sviluppato dalla azienda produttrice di chitarre Gibson. Questo protocollo non è nato nel tentativo di soppiantare il MIDI ma bensì per essere utilizzato in modo complementare, soprattutto con dispositivi come chitarre MIDI. L'ambito ristretto di applicazione e la necessità di *hub box* aggiuntive per la connessione con PC non hanno permesso a questo protocollo di essere implementato su dispositivi di altre case produttrici, se non su alcuni modelli di chitarre Gibson.

HD – MIDI

Il protocollo *HD-MIDI* [33], sviluppato a partire dall'anno 2006 dalla stessa MIDI Manufacturer Association, è stato presentato in forma privata per la prima volta presso la conferenza musicale NAMM nel 2013.
Essendo sviluppato dallo stesso gruppo di lavoro del protocollo MIDI, le specifiche HD-MIDI tendono a proporre soluzioni per la gran parte dei problemi riscontrati negli anni dagli utenti MIDI 1.0 mantenendo la retrocompatibilità con la prima versione. Alcune delle migliorie proposte riguardano la presenza di un supporto alla gestione delle sessioni tra più dispositivi via cavo o wireless, il supporto per più canali trasmissivi e *controller*, una risoluzione superiore nella specifica dei parametri, nuovi messaggi considerati mancanti nel MIDI 1.0, una nuova gestione del parametro di *pitch* per le note non associato ad un numero, in grado di dialogare anche con differenti modalità di accordatura, ed infine la possibilità di modificare in una nota parametri come *attack*, *decay*, *sustain* e *release* (ADSR) comuni in abito di sintesi sonora. Tuttavia non esistono ancora implementazioni commerciali per questo nuovo protocollo al di fuori dei prototipi, non ancora disponibili al pubblico e agli sviluppatori hardware, di proprietà di MIDI Manufacturer Association.

2.2 Protocollo *OSC*

Il protocollo *OSC* [25], acronimo per Open Sound Control, nasce nel 1997 dagli stessi sviluppatori Matt Wright e Adrian Freed che hanno lavorato al protocollo *ZIPI* descritto in precedenza. Tuttora questo protocollo rappresenta una delle soluzioni più in voga e tra le più promettenti per le implementazioni alternative al protocollo MIDI, promettendo di migliorare gli aspetti più critici rispetto all'utilizzo delle moderne tecnologie di controllo e sintesi sonora.

OSC è definito da Wright come un "protocollo per la comunicazione tra computer, sintetizzatori e altri dispositivi multimediali, ottimizzato per le moderne tecnologie di networking" [45]. Il protocollo definisce una struttura binaria precisa per i messaggi, ma è indipendente dal livello di trasporto, essendo utilizzabile con UDP e TCP, via ethernet o attraverso tecnologie wireless.

Oltre alla specifica riguardante le tipologie di dato ammesse, il protocollo *OSC* definisce una serie di metodologie per il trasporto dei dati tra computer, dispositivi e applicativi in esecuzione sulla stessa macchina. Pensato per applicativi musicali, il protocollo *OSC*, avendo una struttura libera tramite la quale è compito dello sviluppatore definire il set di messaggi necessari per ogni applicativo, anche in questo caso, come nel MIDI, il protocollo può essere utilizzato in altri contesti, dove è necessaria la definizione di una specifica per un *layer* di trasporto e la definizione di messaggi specifici.

La struttura binaria dei messaggi e l'implementazione del protocollo di trasporto saranno descritte in maggior dettaglio nel Capitolo 3, poiché OSC è stato scelto per il trasporto delle informazioni di performance musicali del prototipo sviluppato per il lavoro in esame. Le motivazioni della scelta di questo protocollo sono da ritrovarsi nella sua frequente adozione all'interno dei più recenti sistemi di produzione musicale e nella sua relativa semplicità ed efficacia di implementazione.

Sono in seguito considerate alcune problematiche legate all'utilizzo di *OSC* in contesti di live performance, dove l'invio su reti IP necessita di una analisi per quanto riguarda la latenza trasmissiva.

2.2.1 Latenza

Wright focalizza l'attenzione sul fatto che la latenza trasmissiva sia inevitabile; anche con reti dedicate e pacchetti che viaggiano alla velocità della luce, la latenza non è di fatto eliminabile.
Diverse ricerche sono state effettuate per la valutazione della latenza accettabile in una performance dal vivo; in tabella 2.1 sottostante sono mostrati i valori accettabili di latenza definiti da Wright.

Performance	**Latenza accettabile (ms)**	**Distanza percorsa dal suono (m)**	**Distanza percorsa dalla luce (km)**
Limite "real-time" per musica interattiva	10	3,43	2,998
Sessione di musica d'insieme	20	6,86	5,996
Sessione di musica da camera	50	17,15	14,990

Tabella 2.1: valori di latenza per live performance definiti da Wright

Molti altri studi hanno cercato di quantificare i valori di latenza accettabili; in [26] e [27] si evince come una latenza di 25ms sia il massimo accettabile per performance interattive. In alcuni casi la latenza viene considerata come un artefatto positivo, in contesti nei quali la performance risulterebbe troppo diretta, come in situazioni che richiedono l'utilizzo di cuffie.
In [26] sono proposti anche approcci svincolati da una precisa misurazione della latenza nei quali viene accettata anche quando presente in modo consistente. Alcune delle modalità di accettazione della latenza sono proposte nella registrazione di contenuti audio via rete e in particolari *performance* live asincrone (*Latency Accepting Approach* – LAA).
Un parametro importante correlato alla latenza è quello del *jitter*; questo parametro misura la differenza nel tempo del valore della latenza. Quanto più è elevato, tanto più crea problemi nella performance, anche in casi di bassa latenza generale. *OSC* propone una serie di *timetag* temporali per l'ordinamento dei pacchetti e l'eventuale aggiunta di ritardo generale nell'elaborazione per minimizzare gli artefatti provocati dal *jitter*.

Sulla base dei dati appena citati, nel Capitolo 5 è descritto il protocollo di *testing* tramite il quale è stato valutato il prototipo proposto in Capitolo 4, cercando di verificarne la possibilità di utilizzo concreto per live performance in network locali.

2.2.2 OSC vs MIDI

Nell'analisi di un protocollo che vuole porsi come strumento innovativo e alternativo ad una tecnologia che da più di 30 anni è standard nel campo delle tecnologie musicali, è necessario focalizzare l'attenzione su quali siano le sue caratteristiche di forza e quali siano invece gli eventuali limiti. Considerata la larga diffusione del MIDI, una nuova tecnologia dovrebbe quantomeno proporre soluzioni per interfacciarsi con il vecchio protocollo o essere direttamente retrocompatibile. Per questo motivo la tecnologia che appare più adeguata oggigiorno, sebbene non ancora pubblica e in fase prototipale, è quella dell'*HD-MIDI*, direttamente sviluppata da MIDI Manufacturer Association e nativamente retrocompatibile con il protocollo MIDI 1.0.

Il documento [34] si propone di rispondere con attenzione ad alcune delle affermazioni più comuni attribuite al protocollo OSC rispetto al MIDI in documenti rilasciati ufficialmente sul portale *opensoundcontrol.org* [6].

A differenza di quanto espresso nel documento di confronto OSC, anche il MIDI è una specifica hardware indipendente dal protocollo di trasporto, infatti, sebbene la specifica MIDI 1.0 tratti del trasporto via cavo a 5 poli, come descritto nel Capitolo 3 di questo lavoro, è possibile che i messaggi MIDI vengano spediti anche tramite USB e altre tecnologie.

L'affermazione che il protocollo OSC sia più veloce del MIDI è da considerarsi sotto specifiche condizioni: nel caso sia utilizzato il trasporto via ethernet, sia i messaggi MIDI sia i messaggi OSC hanno la stessa velocità di trasferimento e, grazie alla quantità minore di byte occupati da i primi, in realtà il valore di *throughput* [2] è maggiore nel primo caso.

Un'altra obiezione all'affermazione del fatto che OSC sia una tecnologia in grado di scambiare informazioni più velocemente rispetto al MIDI è definita in [5] dove è evidenziato il fatto che risulti improprio paragonare OSC e MIDI in termini di velocità

[2] Indice dell'effettivo utilizzo della capacità del mezzo trasmissivo e quantità di dati trasmessi per unità di tempo.

poiché il primo non prevede nelle specifiche un dato temporale di trasmissione dei dati (*data transmission rate*).

La pluralità delle tipologie di dati ammessi nei messaggi OSC appare in prima analisi a favore rispetto al MIDI; tuttavia grazie ai messaggi *system exclusive*, gli sviluppatori MIDI possono strutturare i dati secondo le proprie necessità, integrando tipologie di dato differenti.

Se da un lato OSC offre il vantaggio agli sviluppatori di definire i propri messaggi, grazie a un sistema di indirizzamento aperto, dall'altro la mancanza di messaggi standardizzati frena la diffusione di uno standard basato sul protocollo OSC.

La possibilità di utilizzare OSC per interfacciarsi in ambiti non legati al controllo di strumenti musicali, si ritrova anche nel protocollo MIDI. Molti sistemi di controllo luci, macchinari audio, video e persino interi show, come per esempio lo spettacolo della fontana presso il Bellagio Hotel[3] e "Pirate Battle at Treasure Island[4]" a Las Vegas, vengono controllati tramite il protocollo MIDI.

Sebbene OSC fornisca agli sviluppatori modalità aperte di definizione dei messaggi, grazie a stringhe di indirizzamento leggibili dall'utente, è necessario che vengano sviluppate soluzioni per renderlo compatibile con il protocollo MIDI. In questo modo diverrebbe uno strumento di supporto in grado di fornire funzionalità aggiuntive mantenendo retrocompatibilità con lo standard. Nel Capitolo 4 viene descritto come il protocollo OSC sia utilizzato per questo scopo, fornendo un'interfaccia di comunicazione per il protocollo MIDI, creando, di fatto, uno strumento in grado di completare lo standard con le funzionalità di networking del protocollo OSC.

2.3 MIDI e dispositivi mobili

Sempre più applicazioni dedicate alla musica vengono pubblicate sui market dei dispositivi mobili. Considerando i sistemi operativi mobili più diffusi, iOS e Android, il primo fornisce un maggiore supporto in quanto a compatibilità [20] con il protocollo MIDI. Infatti, dalla versione 4.2 del sistema operativo mobile di Apple, è stato integrato il supporto nativo al protocollo MIDI tramite il framework *Core MIDI*, già implementato sulle versioni di OS X e descritto nelle sue specifiche nel Capitolo 4.5 di questo lavoro.

[3] https://www.bellagio.com/attractions/fountains-of-bellagio.aspx
[4] http://vegasclick.com/vegas/attractions.html

Grazie a *Core MIDI*, è possibile accedere a interfacce di connessione hardware di controller MIDI esterni e utilizzare classi che già rappresentano le strutture base del protocollo. Tuttavia gli sviluppatori hardware devono adattare i connettori in modo da interfacciarsi con le differenti tipologie di porte proprietarie che la casa di produzione Apple monta sui propri dispostivi mobili. Il primo connettore con la nuova porta Apple *lightning* sembra essere quello proposto dalla casa di produzione IK Multimedia, azienda già attiva nello sviluppo di applicazioni per sintesi sonora in ambiente iOS [47]. Il sistema operativo Android resta tutt'oggi senza un supporto ufficiale per il protocollo MIDI e per la connessione di dispositivi MIDI. Tuttavia in questo caso, la presenza di connettori micro USB standardizzati sui device Android avvantaggia la possibilità di collegare dispositivi MIDI tramite USB, con adattatori micro USB / USB facilmente reperibili in commercio.
Una panoramica sul protocollo USB OTG in Capitolo 3 mostra come sia possibile sfruttare questa modalità di connessione abbinata ad un *layer* software di interfacciamento per connettere i dispositivi MIDI ad Android.
Molte applicazioni presenti sugli store digitali presentano la possibilità di utilizzare le superfici *touch* come controller per strumenti musicali. Le applicazioni per la sintesi sonora in iOS possono essere divise in due categorie: quelle compatibili con i controller MIDI e quelle invece gestite esclusivamente tramite interfaccia *touch*.
Spesso la dimensione ridotta dei controlli rende impossibile l'utilizzo di tali applicativi in ambiti professionali, dove i controller fisici MIDI provvisti di tasti, manopole, slider e bottoni restano la soluzione migliore per veicolare l'espressività del musicista.
Molte applicazioni mobili sono in grado di operare come controller a distanza tramite interfaccia *touch*. Sul market Android nessuna di queste nasce esplicitamente per essere utilizzata come controller a distanza pilotato da un dispositivo MIDI esterno. La soluzione più diffusa in questi casi è quella dell'utilizzo del protocollo OSC per l'invio delle informazioni di performance musicali, generate da controller *touch* sull'interfaccia utente.
Alcune applicazioni per il controllo a distanza di software musicali citate in [13] sono: *TouchOSC*, *ITM MidiLab*, *ITM Pad*, *c74*, *iOSC*, *iTM Tilt*, *Remokon for OSC*, *OSCemote*, *Griid Pro*, *Runxt Life*, *Breath OSC Interface*, *Hex OSC S*, *Control*, *eyoControl*, *OSC Physics*, *Griid Pro*, *SonicLife*, *Ardumote HD*, *Remote for iPad*, *GyrOSC*, *touchAble*, *expressionPad*, *Kapture Pad*, *DrawJong*, *Live Music Coder M^2 OSC*.

Qualora un'applicazione voglia utilizzare il mezzo trasmissivo wireless per il controllo a distanza di software musicali, deve tenere in considerazione la latenza introdotta e quantificarla in modo da valutarne la possibilità di utilizzo in ambito pseudo *real-time*.
Il framework *Core MIDI* permette di gestire canali MIDI via wireless in modo nativo tra gli applicativi iOS e le macchine OS X, anch'esse *Core MIDI* compatibili. In aggiunta, il grande vantaggio rispetto ad Android è rappresentato dalla possibilità di poter interfacciare il dispositivo mobile attraverso una connessione *ad-hoc*[5], generata dalla macchina host. Questo risolve in parte alcuni problemi di latenza per via del routing diretto. Google non permette la connessione verso reti *ad-hoc* sui dispositivi Android, per questioni di sicurezza, a meno di modifiche che prevedono l'ottenimento sul dispositivo dei permessi di root. Il prototipo proposto in Capitolo 4 è stato volutamente sviluppato senza modifiche di questo tipo, per permetterne il suo utilizzo sul più ampio numero possibile di dispositivi compatibili con il protocollo USB OTG. Non essendo basata su tecnologie *Core MIDI* proprietarie, il prototipo proposto è compatibile con un gran numero di piattaforme.

[5] Una rete *ad-hoc* rende possibile collegare in modo indipendente più postazioni wireless tra loro senza nessun dispositivo centrale, router o hub, che funga da tramite.

Capitolo 3

Tecnologie coinvolte

In questo capitolo sono descritte le principali tecnologie utilizzate nello sviluppo dell'applicativo descritto nel Capitolo 4.
In particolare sono analizzate le tecnologie per la connessione dei device MIDI alla piattaforma Android e i protocolli per l'invio e la ricezione dei dati. È stato necessario implementare il protocollo MIDI per poter riconoscere i messaggi in arrivo dal device collegato al dispositivo Android, mentre il protocollo OSC, descritto in seguito, è stato utilizzato per scambiare i dati via rete, essendo uno dei protocolli più in voga negli ultimi anni per la comunicazione delle informazioni di performance verso software e dispositivi musicali. Il *server* OSC *cross-platform* per la ricezione dei messaggi su personal computer proposto è stato sviluppato con il linguaggio di programmazione Pure.
Infine una seconda modalità di indirizzamento dei messaggi MIDI su rete è stata implementata tramite il protocollo RTP MIDI.

3.1 Protocollo MIDI

MIDI (Musical Instrument Digital Interface) è un protocollo di comunicazione digitale. È costituito da un linguaggio standardizzato e da specifiche hardware [32] che rendono possibile a strumenti musicali, controller, processori e altri tipi di dispositivi la comunicazione di parametri musicali e informazioni in tempo reale.
In questa sezione sono descritte le principali caratteristiche di tale protocollo.

3.1.1 Specifiche MIDI

L'unità base di comunicazione nel protocollo MIDI è il messaggio MIDI. Ogni messaggio attraversa il cavo preposto per la comunicazione in modalità seriale[6] ad una velocità di

[6] La trasmissione seriale è una modalità di comunicazione tra dispositivi digitali nella quale le informazioni sono comunicate una di seguito all'altra e giungono sequenzialmente al ricevente nello stesso ordine in cui le ha trasmesse il mittente.

31250 bit/sec (su cavo standard MIDI). Il flusso di messaggi può inoltre percorrere soltanto un verso del cavo.
Le strutture atomiche che compongono i messaggi MIDI sono "parole", costituite da 8 bit ciascuna.
Il protocollo definisce soltanto due tipologie di parole da 8 bit, che risultano sufficienti per la descrizione di tutti i messaggi, lo *status byte* e il *data byte*.

- ***status byte***: identifica quale funzione deve essere eseguita dal ricevente e in particolare a quale dei 16 canali di comunicazione, definiti dal protocollo, il messaggio è destinato.

- ***data byte***: definisce il valore da associare all'evento descritto nello status byte.

La distinzione tra le due tipologie appena citate consiste nell'analisi del bit più a sinistra nella "parola", detto anche *most significant bit*; se uguale a valore zero allora il byte sarà di tipo *data byte*, viceversa sarà di tipo *status byte*.
Il protocollo MIDI descrive anche diverse modalità di interazione tra dispositivo ricevente e messaggi in ingresso.
In particolare esistono 4 modalità differenti in seguito descritte:

- **Modo 1 - Omni on / Poly**: un dispositivo riceve messaggi MIDI da ogni canale e sono indirizzati al canale dello strumento predefinito. Lo strumento inoltre ha la possibilità di suonare più di una nota in contemporanea, in modo polifonico.

- **Modo 2 - Omni on / Mono**: un dispositivo riceve messaggi MIDI da ogni canale, come nella prima modalità, ma può riprodurre soltanto una nota alla volta. Ogni nuova nota in arrivo viene eseguita e la nota precedente attiva viene interrotta.

- **Modo 3 - Omni off / Poly**: in questa modalità il dispositivo considera soltanto i messaggi relativi al canale impostato sullo strumento predefinito. In aggiunta, può eseguire più di una nota in contemporanea. Questa è la modalità solitamente più utilizzata ed è consentito eseguire più strumenti in contemporanea su diversi canali,

come per esempio accade nell'utilizzo di software di sintesi e simulazione di strumenti musicali.

- **Modo 4 – Omni off / Mono**: come in modalità 3, il dispositivo risponde solo ai messaggi relativi al canale impostato sullo strumento predefinito, ma esegue soltanto una nota per volta, come in modalità 2.

I messaggi riguardanti la performance risiedono nella categoria definita come *Channel Voice*. Tutti i messaggi presenti in questa categoria sono stati implementati nel prototipo descritto nel Capitolo 4.
Sono in seguito descritti i messaggi presenti nella categoria *Channel Voice*.

- **Messaggio Note On**: rappresenta l'inizio di una nota MIDI, generata dalla pressione su un tasto di una tastiera o su un altro controller come per esempio un controller pad per batteria MIDI. Il messaggio relativo è composto da 3 byte di informazioni: uno status byte comprendente il canale relativo all'evento, un byte dati per indicare quale nota è associata ed un altro byte dati per indicare il valore della pressione associato. Disponendo di 7 bit per rappresentare il numero di note, poiché il primo è destinato alla specifica del byte di dati, sono rappresentabili 128 note nel protocollo MIDI. Per controparte, anche i valori di pressione possibili, relativi a che valore di *loudness* deve essere associata la nota, sono 128.

- **Messaggio Note Off**: questo messaggio è affine nelle specifiche al messaggio di Note On; tuttavia rappresenta la fine, o spegnimento, di una nota precedentemente attivata. Non ha effetto su una nota che non abbia ancora ricevuto il messaggio di Note On. Il terzo byte sta ad indicare quanto velocemente è stato rilasciato il controller associato alla nota da spegnere. È compito dell'hardware, o software, interpretare o meno questo dato.

- **Messaggio Polyfonic Aftertouch**: alcuni dispositivi sono in grado di rilevare il variare della pressione di un controller una volta che è stato premuto la prima volta; la misurazione del cambiamento nella pressione con cui è premuto tale tasto o controller, genera questo tipo di messaggi. È compito del software o hardware

ricevente interpretare o meno questo tipo di messaggi. Tipicamente viene associato a variazioni nei valori di effetti come *vibrato*, *loudness*, inserimento di filtri o incremento e diminuzione del *pitch*. I 3 byte che costituiscono il messaggio sono rispettivamente: uno status byte con indicazione del tipo di messaggio e canale associato, un byte dati per la rappresentazione di quale nota ha generato il messaggio ed infine un byte dati per il valore della pressione associata.

- **Messaggio Channel Pressure**: questo messaggio è trasmesso da uno strumento che risponde a un'unica variazione della pressione indipendentemente dal tasto premuto. L'effetto viene replicato per tutte le note attive al momento della ricezione. I 3 byte che costituiscono il messaggio sono rispettivamente: status byte con indicazione del tipo di messaggio e canale associato, un byte dati per la rappresentazione di quale nota ha generato il messaggio ed infine un byte dati per il valore della pressione associata.

- **Messaggio Program Change**: questo messaggio è utilizzato per modificare il programma sonoro attualmente in utilizzo sul dispositivo ricevente. Per esempio può servire per modificare il preset associato ad uno strumento. Il messaggio è formato da un byte di status indicante il tipo di messaggio e il canale associato ed un byte dati per il valore di program change associato. Poiché il *most significant bit* per il byte di dati è destinato alla specifica del byte dati, con i restanti 7 bit è possibile rappresentare 128 diversi valori di program change.

- **Messaggio Pitch Bend**: questo messaggio è generato da una particolare rotella presente in alcuni strumenti MIDI come tastiere. È in genere utilizzata per modificare il *pitch* di una nota in esecuzione. Poiché il valore associato è rappresentato da 2 byte dati, tolti i *most significant bit,* con i restanti 14 bit sono rappresentabili 16384 valori per questo controller. Essendo un controller fisico di tipo continuo, genera molto traffico sul cavo midi, se paragonato al traffico generato dai tasti di una tastiera MIDI.

- **Messaggio Control Change**: questo tipo di messaggi comunica al dispositivo ricevente una modifica di un controllo per l'esecuzione in tempo reale. Esistono

molti parametri associati a questo messaggio, suddivisi in due categorie: controller continui e controller di tipo switch. Alcuni controller associati a questo messaggio sono per esempio pedali, leve, breath controller per la modulazione tramite flusso d'aria e rotelle. Il messaggio è formato da 3 byte: il primo status byte descrive il tipo di messaggio e il relativo canale MIDI, il secondo byte di dati descrive quale tipo di controller va a modificare ed infine il terzo byte dati con quale valore effettuare la modifica.

- **Messaggio All sound Off (categoria Control Change)**: sebbene questo messaggio risieda nella categoria Control Change, è qui descritto perché di grande importanza nel controllare e correggere messaggi MIDI eventualmente persi. Nell'eventualità di un blocco del sistema di performance live, o nel caso in cui un messaggio di Note Off venga perso, questo messaggio spegne in una volta sola tutte le note. Essendo un messaggio con un forte impatto sulla performance, è inteso per essere utilizzato in casi particolari e spesso associato a un tasto detto "panic button". Nel prototipo implementato nel Capitolo 4, questa funzione è stata associata ad un bottone sullo schermo del device Android, proprio perché nel contesto di networking potrebbe capitare che alcuni messaggi vengano persi, soprattutto con connessioni tramite protocollo UDP, risultando in una errata gestione dei messaggi Note Off, con conseguente artefatto nella performance real time.

Esiste una seconda categoria di messaggi MIDI detti *Sytem Messages*, che vengono trasmessi globalmente a tutti i dispositivi presenti nella catena di collegamento, poiché i canali MIDI non sono specificati in questa tipologia di messaggi. Esistono 3 sotto categorie di messaggi: messaggi *System-common*, messaggi *System Real-Time* e *System Exclusive*.

- **Messaggi *System Common***: messaggi destinati al trasporto di informazioni come posizionamento e sincronizzazione del timing associato a un brano MIDI.

- **Messaggi *System Real-Time***: messaggi formati da un unico byte il cui scopo è quello di sincronizzare diversi device collegati tra loro. Poiché importanti per la

sincronizzazione Real-Time, essi hanno priorità su tutti gli altri tipi di messaggi e possono essere inseriti in un qualsiasi punto del flusso dati.

- **Messaggi *System Exclusive***: questo tipo di messaggi permette ai produttori di hardware MIDI, a sviluppatori e designer di prevedere modalità di comunicazione personalizzata non definibile attraverso gli altri messaggi presenti nel protocollo MIDI. In particolare, data la personalizzazione possibile, non è definita una lunghezza standard per questi messaggi, ma esistono di fatto 2 messaggi che indicano l'inizio e la fine di uno *stream* di un messaggio *system exclusive*.

3.1.2 Hardware MIDI

Il cavo principale attraverso il quale sono trasmesse le informazioni tra un dispositivo MIDI e il relativo ricevente è stato studiato appositamente per lo standard.

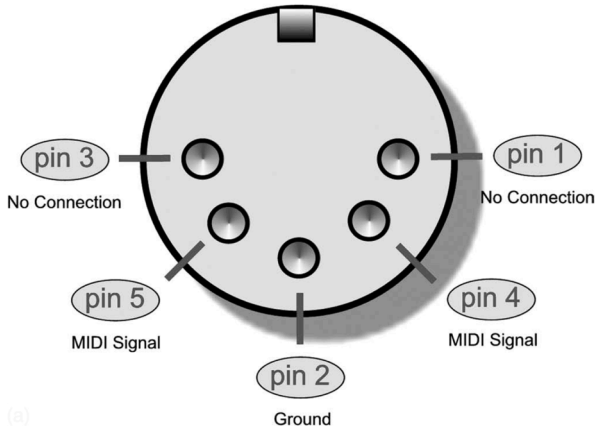


Figura 3.1: Connettore cavo MIDI

Il connettore maschio possiede 5 pin alla sua estremità; tuttavia sono utilizzati soltanto 3 pin. Come mostrato in figura 3.1, i pin dedicati alla trasmissione delle informazioni sono il 4 e il 5, mentre il pin numero 2 collega la messa a terra elettrica.
Alcune implementazioni parallele allo standard MIDI hanno portato all'utilizzo dei pin 1 e 3 per il trasporto di corrente elettrica per alimentare il dispositivo ricevente, procedimento definito come MIDI *phantom power*.

Un'interessante evoluzione del trasporto delle informazioni MIDI si è avuta con l'introduzione dei dispositivi in grado di trasportare il MIDI via wireless. Dispositivi di questo tipo permettono l'invio e la ricezione dei messaggi con un piccolo overhead temporale di latenza introdotto per via del mezzo trasmissivo e sono in grado di inviare segnali fino a circa 150 metri di distanza.
Uno degli obiettivi primari collegati allo sviluppo del prototipo descritto in Capitolo 4, è stato proprio quello di sviluppare una applicazione per smartphone e tablet Android in grado di emulare questo tipo di dispositivi, fornendo quindi le medesime funzionalità ma con un costo modesto considerato il prezzo ancora elevato di questo tipo di dispositvi.
Le connessioni in genere presenti sui dispositvi MIDI sono solitamente 3: MIDI IN, MIDI OUT e MIDI THRU. Il connettore MIDI IN serve per collegare il cavo per i messaggi in ingresso. Il connettore OUT manda i messaggi prodotti dal dispositivo all'esterno attraverso questa porta. Attraverso la porta THRU è invece inviata in uscita una copia dei messaggi in ingresso fornendo la possibilità di creare catene di dispositivi MIDI collegati in serie; questo tipo di collegamento è chiamato *daisy chain*.

3.2 USB On-The-Go

La possibilità di utilizzare il collegamento di dispositivi USB su piattaforma Android è una feature relativamente recente; è dalla versione 3.1 del sistema operativo (api level 12) [10] che gli sviluppatori hanno la possibilità di accedere ad alto livello alle funzionalità di interazione con tali dispositivi.
Prima di analizzare in che modo è possibile interfacciarsi attraverso il codice nativo java, sono approfondite le funzionalità principali del protocollo USB On-The-Go, implementato dai dispositivi Android che permettono il collegamento di device USB.

USB On-The-Go spesso abbreviato come USB OTG, è un protocollo che permette a dispositivi USB come player audio digitali, telefoni cellulari o tablet di interagire come *host*, permettendo il collegamento ad essi di altri dispositivi USB come memorie flash, camere digitali, dispositivi di puntamento o tastiere.
Come descritto in [35], le prime implementazioni del protocollo USB OTG sono apparse sul mercato nel 2004.

I dispositivi che implementano questo protocollo sono infatti in grado non solo di essere collegati a un *host* come per esempio un computer, ma possono loro stessi agire come *host* una volta collegati ad un dispositivo USB.
In [35] è sottolineato il fatto che non sia necessario che entrambi i dispositivi implementino il protocollo, è sufficiente infatti che tale tecnologia sia supportata dal device che deve assumere la funzionalità di *host*.
Se collegati due dispositivi che supportano entrambi USB On-The-Go, grazie al protocollo Host Negotiation Protocol (HNP) [36] è selezionato il dispositivo principale. Se l'applicativo richiede un'inversione dei ruoli, questo avviene in modo trasparente senza che l'utente ne abbia percezione.

3.2.1 Architettura USB OTG

Il protocollo USB utilizza un'architettura di tipo *master*/*slave*. Per tutta la durata del collegamento, un dispositivo funge da *master*, l'altro da *slave*.
In genere ogni dispositivo nasce con un compito prestabilito; per esempio un PC ricopre il ruolo di *master* mentre una stampante ha quello di *slave*.
Quando un dispositivo è collegato a un bus USB, il *master device*, o *host*, ha il compito di istanziare il collegamento. L'*host* è quindi responsabile di tutti i trasferimenti di dati attraverso il bus.
Per esempio, per trasferire dati tra un PC e una stampante, l'*host* legge inizialmente i dati da spedire e poi avviene il trasferimento al secondo device.
Questo permette ai device di tipologia *slave*, come per esempio un mouse, di contenere una logica molto semplice rispetto ai compiti che deve eseguire il device *master*.
Mentre la logica di suddivisione dei ruoli è inequivocabile per l'esempio sopracitato, nel caso di una stampate, lo scenario è più complesso. Nel caso in cui la stampante è collegata ad un PC, è quest'ultimo che deve interagire come *master*. Nel caso invece in cui venga collegata per esempio una memoria flash alla stampante, è quest'ultima che deve assumere il ruolo di *master* e occuparsi dello scambio di dati necessari per stampare i files presenti sulla memoria.

USB On-The-Go introduce il concetto che un device possa interagire con un altro sia nel ruolo di *master*, sia nel ruolo di *slave*; in questo modo può essere considerato come *host* nel caso sia collegato come *master*, oppure considerato come periferica nel caso di collegamento di tipo *slave*.
La scelta di quale dispositivo assumerà ruolo master o slave è unicamente gestita dal modo in cui sono collegati i due device: il dispositivo collegato nell'ingresso dell'altro fungerà da *slave* mentre il secondo come *master*.

In seguito a una fase iniziale di setup, le operazioni eseguite sono le stesse del collegamento USB standard, dove il device *master* gestisce le comunicazioni con il dispositivo *slave*.
L'inversione di ruolo è possibile grazie al protocollo Host Negotiation Protocol, nel caso in cui entrambi i dispositivi implementino le specifiche OTG. Il processo di inversione dei ruoli è possibile soltanto attraverso un collegamento diretto; non è possibile infatti interporre una periferica di tipo USB hub, per la replica di porte.

3.2.2 Specifiche e protocollo USB OTG

USB On-The-Go è parte del supplemento al documento ufficiale dello standard USB pubblicato nel 2001.
L'ultima versione di questo supplemento definisce alcune specifiche come SuperSpeed OTG devices pubblicate attraverso il documento [40].
Il documento [36], introduce tre protocolli di comunicazione, Attach Decection Protocol (ADP), Session Request Protocol (SRP) e Host Negotiation Protocol (HNP) descritto in precedenza.
ADP permette a un device OTG di determinare lo stato di collegamento in assenza di un bus USB alimentato. Questo avviene grazie a una periodica misurazione della capacità elettrica sulla porta USB; quando la misurazione varia oltre una certa soglia è in grado di ottenere informazioni circa il collegamento di un dispositivo, oppure determinare la presenza di un cavo senza dispositivo dall'altro capo. Una volta notificato del collegamento, il device *master* provvede a fornire corrente elettrica per alimentare il dispositivo che è stato collegato.

Il protocollo SRP è in grado di controllare quando è in funzione l'erogazione della corrente elettrica. Nei collegamenti USB standard soltanto l'*host* è in grado di eseguire questa operazione.
Questo controllo permette di bilanciare il consumo di batteria, molto importante quando il protocollo OTG è implementato da dispositivi mobili, dove la gestione delle politiche energetiche deve essere opportunamente sviluppata in modo da prevenire ogni tipo di consumo non necessario.
Il dispositivo *master* OTG può interrompere l'erogazione di energia su collegamento USB finché la periferica USB non ne richieda l'utilizzo.

Il protocollo HNP permette a due dispositivi OTG collegati tra loro di invertire i ruoli di *host* e periferica.
Uno dei motivi pratici di utilizzo di tale protocollo è quando l'ordine di collegamento è errato: una stampante e una camera possono automaticamente invertire i ruoli per ripristinare la corretta suddivisione dei ruoli.
Il documento [40] introduce un protocollo addizionale, il Role Swap Protocol (RSP), grazie al quale la funzionalità ottenuta con HNP è estesa anche allo standard USB 3.0.
Le periferiche che adottano le specifiche descritte in [36] devono necessariamente attenersi anche a quelle descritte nel supplemento della versione 2.0 per mantenere retro compatibilità.

I dispositivi SuperSpeed OTG devono supportare il protocollo RSP, mentre quelli di tipo SuperSpeed Peripheral Capable OTG non devono supportare RSP poiché possono solo operare come periferiche e quindi verrebbe meno il concetto di inversione dei ruoli.

3.2.3 Ruoli USB OTG e Targeted Peripheral List

USB OTG definisce due ruoli per i dispositivi: il primo OTG A-device e il secondo OTG B-device.
Questa terminologia vuole spiegare il verso della distribuzione di corrente che è fornita al collegamento e in particolare quale dei due dispositivi deve assumere ruolo di *host* iniziale.
L'OTG A-device è il dispositivo che fornisce la corrente elettrica necessaria al funzionamento del dispositivo OTG B-device.

La configurazione standard descrive quindi A-device come USB *host* e B-device come periferica *slave* USB.
Un device che può assumere entrambi i ruoli non deve essere necessariamente compatibile con tutti i tipi di periferica USB. Il costruttore deve fornire una lista di compatibilità nella quale sono inseriti i dispositivi che possono funzionare come periferica con quel determinato prodotto.
La versione base di questa lista, chiamata *Targeted Peripheral List* deve contenere per ogni device supportato informazioni come tipologia, numero di modello e nome del produttore.

3.2.4 Connettori USB OTG

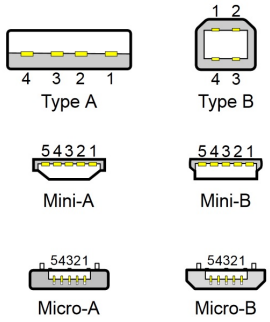


Figura 3.2: Connettori USB OTG

Nell'immagine 3.2 è possibile osservare le differenti tipologie di connettori OTG.
In particolare sulla piattaforma Android, come è descritto in seguito, la porta più diffusa è quella di tipo micro USB, la stessa utilizzata per ricaricare i dispositivi e per collegare verso un *host* USB standard.
Adattatori da micro USB a standard USB sono facilmente reperibili in commercio e rendono il prototipo proposto nel Capitolo 4 particolarmente semplice da utilizzare e predisporre.
Il cavo OTG aggiunge un pin in più, detto ID-pin, rispetto al connettore USB standard che viene collegato alla terra dal lato del connettore A, mentre non è collegato sul lato B.
Il dispositivo collegato al connettore A assume il ruolo di OTG-A, mentre quello collegato all'altro lato assume ruolo di OTG B-device.

La presenza di un nuovo dispositivo collegato è notificata al dispositivo A grazie allo stato del quinto pin aggiuntivo ID-pin.
Le porte USB OTG hanno anche la possibilità di essere collegate a un caricatore per dispositivi accessori USB.
Questo permette a dispositivi come smartphone e tablet di fornire corrente sufficiente ad alimentare i dispositivi collegati. Come descritto in seguito il prototipo sviluppato, laddove utilizzato su dispositivi in grado di fornire tensione elettrica, fornisce energia per alimentare i dispositivi MIDI a esso connessi.
Per collegare un dispositivo OTG a un PC è necessario un cavo che abbia da un lato un connettore USB standard di tipo micro A (figura 3.2) e dall'altro lato un connettore di tipo B (figura 3.2). Per collegare invece una periferica ad un dispositivo OTG è necessario che questa abbia un connettore di tipo micro A maschio.
In commercio sono facilmente reperibili adattatori che convertono connettori USB standard in connettori USB micro OTG; oggigiorno sono considerati accessori standard dai possessori di tablet o smartphone OTG compatibili.
Per collegare tra loro due dispositivi OTG è necessario un cavo con connettori mini/micro entrambi a 5 pin, in modo che possano scambiare le informazioni riguardanti ruolo di *host* e di periferica, oppure con una combinazione di cavo USB standard e rispettivi adattatori micro A e B alle estremità.

3.2.5 Retrocompatibilità

I dispositivi USB OTG sono retrocompatibili con lo standard USB 2.0; quelli SuperSpeed OTG retrocompatibili invece con lo standard 3.0.
Tramite il protocollo OTG è garantita l'alimentazione per la periferica collegata; tuttavia tale energia fornita potrebbe non essere sempre sufficiente per alimentare tutti i device presenti sulla *Targeted Peripheral List*.
È possibile che tramite un hub USB alimentato, interposto tra il dispositivo e la periferica, si vada a sopperire la quantità di corrente mancante.

3.3 USB On-The-Go e Android

In questo paragrafo sono descritte le principali caratteristiche dell'implementazione del protocollo USB OTG su piattaforma Android, la cui analisi ha permesso di implementare il prototipo proposto in Capitolo 4.

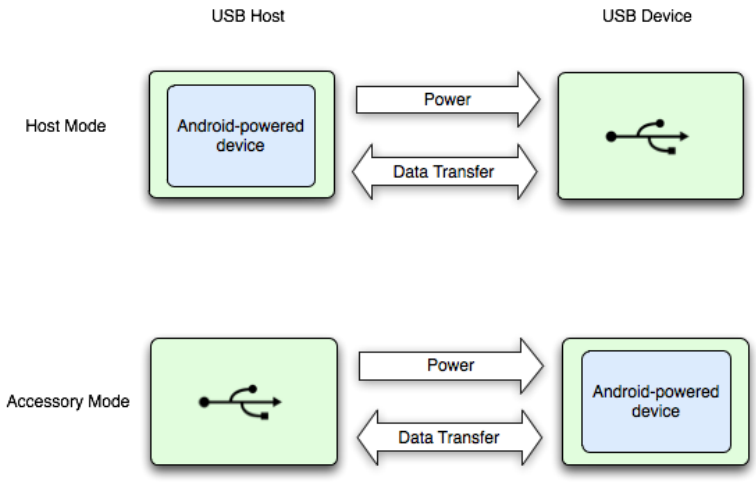


Figura 3.3: modalità USB OTG su piattaforma Android

Esistono due modalità di collegamento che la piattaforma Android offre, la prima detta *accessory mode*, mentre la seconda *host mode*. Tuttavia il supporto nativo per queste modalità di connessione si ha solo con la versione 3.1 del sistema operativo (api livello 12) e successive.

Esiste una libreria di compatibilità per abilitare la *accessory mode* fino alla versione 2.3.4 [10].

L'utilizzo software delle api rilasciate da Google è strettamente legato alle caratteristiche hardware di ciascun dispositivo; non è raro ritrovare tali caratteristiche su tablet smartphone di case produttrici come Samsung e Google stessa.

Sono ora presentate le due modalità di connessione USB fornite dalla piattaforma Android.

Nella modalità *accessory* i dispositivi esterni USB, come per esempio docking station, card reader e strumenti diagnostici, funzionano come *host* USB. Questo permette anche ai quei device privi di capacità *host*, di interagire con tali periferiche, tuttavia esse devono necessariamente essere progettate per aderire al protocollo di comunicazione *accessory*. Il protocollo *accessory* è stato sviluppato a partire dalla piattaforma elettronica open source

Arduino, estesa con opportune patch software e hardware per poter comunicare con Android, come presentate nella guida per sviluppatori hardware Android "The Android Accessory Development Kit (ADK) for 2012". Tuttavia, poiché il prototipo sviluppato per questo elaborato e descritto nel Capitolo 4 non si serve di quest'ultima modalità di connessione, non sarà approfondita ulteriormente.
Nella modalità USB *host*, utilizzata per l'implementazione del prototipo proposto, il dispositivo Android interagisce con il ruolo di *host*. Alcuni esempi di periferiche compatibili in questo caso sono: camere digitali, tastiere per l'immissione di testo, periferiche HID (human interface device) come mouse e controller per il gioco.
In figura 3.3 è possibile osservare lo schema di collegamento per le due modalità appena descritte. Quando un dispositivo Android assume ruolo di USB *host* ha anche il compito di fornire la corrente elettrica sul link di collegamento necessaria ad alimentare la periferica collegata.

3.3.1 Android USB host API overview

Una serie di classi presenti nel package *Android.hardware.usb* forniscono il *layer* per l'interfacciamento ad alto livello in codice nativo java verso l'hardware USB.

La classe *UsbManager* permette di listare e comunicare con i dispositivi USB connessi, che, qualora collegati tramite un hub USB, possono essere in numero variabile.

La classe *UsbDevice* rappresenta un dispositivo USB una volta connesso e contiene i metodi per accedere alle sue informazioni, interfacce e *endpoints*.

La classe *UsbInterface* rappresenta l'interfaccia di un dispositivo USB, che definisce un set di funzionalità per quest'ultimo. Un dispositivo può sempre avere più di un'interfaccia attraverso la quale avvengono le comunicazioni.

La classe *UsbEndPoint* rappresenta l'*endpoint* di un'interfaccia di un dispositivo USB, ovvero un canale attraverso il quale vengono trasferite le informazioni. Ogni interfaccia può avere uno o più *endpoint*; tipicamente esistono *endopoint* di ingresso e uscita per permettere una comunicazione bilaterale con il dispositivo.

La classe *UsbDeviceConnection* rappresenta l'oggetto di connessione al dispositivo, tramite il quale vengono trasferiti i dati attraverso gli *endpoint*. Inoltre questa classe permette di inviare e ricevere dati nelle in modalità sincrona o asincrona.

La classe *UsbRequest* rappresenta un oggetto di richiesta asincrona per la comunicazione con un dispositivo attraverso la classe *UsbDeviceConnection* sopra citata.

La classe *UsbConstants* definisce costanti che corrispondono alle definizioni presenti nel file linux/usb/ch9.h [31] del kernel di Linux, poiché l'architettura Linux è alla base della piattaforma Android.
In particolare questo file racchiude le costanti e le strutture che sono necessarie per le API dei dispositivi USB. Queste costanti sono utilizzate nella definizione del modello di dispositivo USB, definito nel capitolo 9 del documento [38].
Qui sono presenti anche le costanti necessarie per la modalità aggiuntiva USB On-The-Go che, come già citato permette a sistemi di funzionare sia come master/host sia come slave/periferica [31].

Per poter usufruire delle funzionalità di connessione USB OTG nello sviluppo di una applicazione Android è necessario inserire informazioni aggiuntive all'interno del file *manifest*. Il file *manifest*, univoco per ogni applicazione, rappresenta informazioni essenziali che il sistema deve elaborare nelle prime fasi di esecuzione.
In particolare, oltre alle dichiarazioni delle classi *view controller*, dette *activity*, del package di sviluppo e dei permessi necessari, devono essere dichiarati dei particolari *flag*. Tra questi *flag* è presente quello per autorizzare l'applicazione ad interfacciarsi all'hardware USB OTG. Oltre a questo è necessario specificare il livello di API target necessario al funzionamento, che per poter utilizzare le api USB OTG deve essere superiore o uguale alla versione 12. Inoltre è possibile già specificare nel *manifest* quali dispositivi USB sono compatibili con l'applicazione, in conformità a determinati codici di tipologia e del produttore hardware.
È possibile inoltre predisporre un'applicazione Android in modo che venga notificata quando un dispositivo USB, utilizzato durante l'esecuzione, viene collegato. Questa operazione funziona anche nel caso in cui l'applicazione non sia in esecuzione.

La proprietà utilizzata per questo scopo è quella rappresentata dall'*intent-filter*, sempre specificata nel file *manifest*. Una volta notificata l'applicazione, l'utente può scegliere se attivare la connessione USB ed eseguire il programma.
Allo stesso modo è possibile inserire un *flag* all'interno del *manifest* in modo che l'applicazione sia notificata della disconnessione fisica del dispositivo, in modo da disconnettere anche la comunicazione via software.
Se non è stato attivato il processo automatico per il quale l'applicazione viene notificata del nuovo collegamento di un dispositivo, è comunque necessario che l'utente dia la conferma per l'utilizzo del device USB tramite l'applicazione. Infatti nel momento della connessione via codice, prima che questa avvenga, viene mostrato un avviso all'utente, il quale deve confermare la volontà di utilizzare tale dispositivo tramite l'applicazione.
Per poter scambiare dati con il dispositivo, l'applicazione deve registrare gli *endpoint* di *input* e *output* per poter creare una connessione bilaterale. Nel Capitolo 4 è descritto in dettaglio il procedimento di utilizzo per lo sviluppo del prototipo proposto.

3.4 Open Sound Control

Dopo aver delineato nel Capitolo 2 di questo lavoro le caratteristiche principali del protocollo per lo scambio di informazioni musicali Open Sound Control, in questo paragrafo ne sono descritte le relative caratteristiche tecniche e implementative [43].
I dati associabili al protocollo di trasporto OSC sono i seguenti:

- **int32**: Integer a 32 bit big-endian[7].

- **OSC-timetag**: tag temporale 64 big-endian come descritto in seguito.

- **float32**: 32-bit big-endian IEEE 754 floating point[8].

- **OSC-string**: una sequenza di caratteri ASCII non nulli seguiti da un carattere nullo più altri caratteri opzionali nulli (da 0 a 3) per rendere il numero totale di bit multiplo di 32.

- **OSC-blob**[9]: oggetto binario di grandezza int32, seguito da un numero variabile di byte da 8 bit di dati arbitrari, seguiti da un numero variabile (da 0 a 3) di zeri addizionali per rendere il numero totale di bit multiplo di 32.

La grandezza di ogni dato atomico nel protocollo è sempre un multiplo di 32 bit. L'allineamento dei dati con parole di lunghezza 32 bit garantisce performance maggiori nel modo in cui una CPU gestisce la memoria. Come descritto in precedenza, per mantenere questo allineamento è necessario inserire dati arbitrari in modo che la fine di una struttura dati coincida con l'inizio della struttura successiva.

[7] Il big-endian, è una modalità usata dai calcolatori per immagazzinare in memoria dati di dimensione superiore al byte. Nel formato big-endian i byte più significativi sono memorizzati all'indirizzo di memoria più piccolo ed i byte successivi negli indirizzi più grandi.
[8] IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985). Standard per la codifica di numeri in virgola mobile.
[9] Acronimo per *binary large object*: tipo di dato usato per la memorizzazione di dati di grandi dimensioni in formato binario.

3.4.1 Pacchetti OSC

L'unità base, definita in [43], del protocollo di trasmissione OSC è il pacchetto OSC. Ogni applicazione che invia pacchetti OSC è detta *client* OSC mentre ogni sistema che riceve pacchetti OSC è detto *server* OSC.
Nel Capitolo 4 di questo elaborato sarà mostrato come, per la definizione appena citata, l'applicativo sviluppato assume sia ruolo di *client* OSC che ruolo di *server* OSC per l'invio e la ricezione di pacchetti OSC.
Il numero di byte da 8 bit di cui è composto un pacchetto OSC è sempre un mutiplo di 4.
Nel protocollo OSC non è prevista una tecnologia preposta per la trasmissione dei dati. Ogni applicativo che utilizza OSC può scegliere quale metodo per l'invio dei pacchetti, tipicamente scegliendo tra i protocolli UDP o TCP/IP.
Il contenuto di un pacchetto OSC può essere costituito da un messaggio OSC o da un bundle OSC, descritto in seguito.
I pacchetti OSC vengono marchiati in base alla tipologia nel primo byte dei dati.

3.4.2 Messaggi OSC

Un messaggio OSC è formato da uno specifico *address pattern,* da un *type tag* ed infine da un numero variabile di argomenti.
L'*address pattern* è una stringa il cui primo carattere deve essere uno *slash* "/", utile per il riconoscimento e l'indirizzamento dei dati presso un *server* OSC.
Il *type tag* è una stringa che identifica la tipologia dei dati costituiti dagli argomenti.
Questa stringa è costituita da un carattere "," iniziale e da ulteriori caratteri rappresentativi del tipo di argomento separati da virgola, uno per ogni argomento inserito nel messaggio.
Questo significa che nello stesso messaggio OSC possono coesistere più tipologie di dati; in particolare i tipi standard di dato ammessi sono int32, float32, stringhe e *blob*.
Esistono *type tag* che non rientrano nella categoria standard, dei quali il più rilevante per questo elaborato è quello che definisce la tipologia di dato del messaggio MIDI. In questo caso specifico il tipo di dato MIDI è costituito da 4 byte ordinati dal MSB[10] al LSB[11]. I dati

[10] *Most Significant Byte*
[11] *Less Significant Byte*

inseriti sono quindi numero di porta per la comunicazione di network, *status byte* MIDI, primo *data byte* e secondo *data byte*.
L'utilizzo di questo *tag* non è vincolante per l'impacchettamento di dati MIDI, poiché riconducibili a dati binari; per questo motivo si è scelto di non utilizzarlo nell'implementazione del protocollo OSC del *case study*.
In conclusione gli argomenti OSC possono essere considerati come una sequenza di dati binari rappresentanti ogni singolo argomento inserito nel messaggio OSC.

3.4.3 Bundle OSC

Un *bundle* OSC è da considerarsi come contenitore per altri *bundle* o messaggi OSC.
Prima del contenuto del *bundle* viene anteposta la stringa speciale *"#bundle"* e un argomento definito come *timetag* OSC.
OSC non definisce un sistema per la sincronizzazione dei dati inviati: un messaggio OSC deve essere elaborato da un *server* OSC, come definito in precedenza, non appena lo stesso viene ricevuto. Al contrario con i *bundle* OSC si utilizza il *timetag* OSC per valutare la tempistica necessaria entro la quale elaborare i dati; se il *timetag* corrisponde o è precedente al timestamp del runtime del server, il dato deve essere immediatamente elaborato, mentre se il *timetag* è superiore al timestamp corrente, il calcolatore deve creare una coda di elaborazione in attesa del tempo corretto di elaborazione.
Il *timetag* OSC aderisce alla specifica per timestap del protocollo NTP internet (*network time protocol*[12]).

3.4.4 Indirizzamento OSC

Grazie alla struttura degli *address pattern* descritti in precedenza, è possibile considerare la struttura di indirizzamento per il *server* OSC come una struttura ad albero. L'ultimo argomento dell'*address pattern* porta alla chiamata alla procedura o funzione che dovrà utilizzare ed elaborare i dati presenti nel messaggio OSC.
Un esempio di *address* OSC può essere considerata la seguente stringa: "/radice/ramo1/sottoramo1/foglia1/". Ogni *address* OSC inizia con il carattere "/" e lo

[12] NTP: The Network Time Protocol - http://www.ntp.org/

stesso è utilizzato per separare gli attributi seguenti. L'ultimo attributo indica solitamente la funzione che deve processare i dati contenuti nel messaggio OSC.
Il processo di indirizzamento è detto *dispatching* del messaggio OSC verso il metodo OSC che coincide con l'*address pattern* appena descritto.
Oltre ad un indirizzamento sequenziale è possibile utilizzare tecniche di *pattern matching*[13].
Nel Capitolo 4 è descritto l'*address pattern* scelto per il *server* OSC del prototipo implementato.
Il *server* OSC utilizzato per ricevere ed elaborare sarà trattato nel prossimo paragrafo.

3.5 Pure Data

Il *case study* presentato nel Capitolo 4, qualora utilizzato per scambiare messaggi OSC, presenta un *server* OSC in grado di ricevere ed elaborare questo tipo di messaggi, sviluppato utilizzando il linguaggio di programmazione visuale open source Pure Data.
Nelle iniziali fasi di sviluppo è stato creato un server java per la ricezione dei messaggi OSC, ma una seconda implementazione tramite Pure Data, si è rivelata meno complessa da strutturare, mantenendo allo stesso modo la caratteristica principale di cross compatibilità su diversi sistemi operativi.
In seguito è presentata la struttura principale del framework Pure Data, con particolare attenzione alle caratteristiche di gestione del linguaggio MIDI.

Pure Data è un linguaggio di programmazione visuale per l'elaborazione dei segnali audio in tempo reale [11], sviluppato inizialmente dall'ingegnere Miller Puckette, il quale ha collaborato anche in precedenza allo sviluppo del software Max/Msp, framework con la stessa finalità di elaborazione audio ma in questo caso prodotto commerciale.
I linguaggi di programmazione visuale, anche definiti tramite la sigla VPL, permettono di creare programmi manipolando elementi grafici ad alto livello, i quali si occupano di tradurre le modifiche visuali in codice nativo. Pure Data, e in generale i linguaggi di programmazione visuale, sono basati sul concetto di forme (associati al concetto informatico di oggetto) e la loro correlazione tramite frecce e linee.

[13] Riconoscimento di un *pattern* all'interno di una sequenza di caratteri.

L'applicazione primaria di Pure data è l'elaborazione dei segnali audio, tuttavia esistono diverse librerie, dette *externals* in Pure Data, il cui nome specifico è *Graphics Environment for Multimedia* (GEM), per la manipolazione di immagini in 2D e la creazione di scene 3D. Inoltre è possibile interfacciarsi a Pure Data con una vasta gamma di controller esterni per la creazione di installazioni interattive.
Pure Data non è un software proprietario, ma bensì sviluppato sotto licenza BSD.[14]
Uno dei principali vantaggi di Pure Data è dato dalla possibilità di modificare i programmi durante il *runtime*[15], rendendone il suo utilizzo particolarmente indicato per performance live.
Molti *external* sviluppati dagli utenti della comunità attorno al progetto, sono regolarmente inclusi negli aggiornamenti del framework.

3.5.1 Pure Data: Visual Programming

Il concetto di flusso dei dati collegato alla programmazione visuale è detto *dataflow* [4]; in Pure Data i dati fluiscono attraverso le connessioni e sono in seguito processati dagli oggetti grafici che rappresentano concetti complessi di sound design. I programmi sviluppati con Pure Data, detti *patch*, risultano spesso visivamente molto complessi poiché non esistono limiti spaziali nella organizzazione grafica della stessa *patch*.
Sono ora descritti i principali componenti visuali necessari alla compilazione di una *patch* in Pure Data.

3.5.2 Oggetti

Il componente "oggetto", mostrato in figura 3.4, rappresenta un contenitore in cui i dati entrano, vengono elaborati, ed escono. Gli ingressi a questi componenti vengono detti *inlets* mentre le uscite *outlets*. Gli *inlets* sono posti per convenzione nella parte superiore di un oggetto, mentre gli *outlets* nella parte inferiore. Il numero di ingressi e uscite dipende da che strumento viene associato ad ogni oggetto. Per istanziare un oggetto è necessario

[14] Le licenze BSD sono una famiglia di licenze permissive, senza copyleft, per software. Molte sono considerate libere ed open source. Il loro nome deriva dal fatto che la licenza BSD originale fu usata originariamente per distribuire il Unix (BSD), una revisione libera di sviluppata presso l'Università di Berkeley.
[15] Momento in cui un software è in esecuzione.

scriverne la sigla all'interno dell'oggetto, dove è presente una *label* testuale apposita per questo inserimento.
Se la *keyword* inserita viene riconosciuta dal sistema, l'oggetto mostra gli *inlets* e *outlets* corrispondenti, altrimenti l'oggetto stesso assume una colorazione rossa indicante il probabile errore nella *keyword*. Gli oggetti di Pure Data si dividono tra quelli del *core* e quelli presenti come *externals*, per alcuni di quest'ultima categoria è necessario importare il modulo aggiuntivo tramite l'oggetto *import* seguito dal nome del modulo.

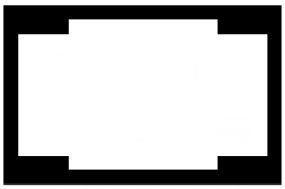


Figura 3.4: Oggetto Pure Data

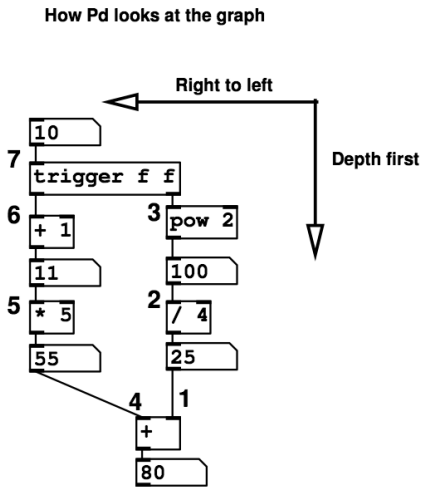
3.5.3 Dati e connessioni

Come descritto in precedenza, Pure Data utilizza delle connessioni grafiche dette *wires*, per collegare più oggetti tra loro e per definire il flusso dei dati attraverso i vari oggetti.
Il collegamento viene creato a partire da uno degli *outlets* di un oggetto verso uno degli *inlets* dell'oggetto successivo. Le connessioni che trasmettono segnale audio hanno uno spessore maggiore rispetto alle connessioni che trasportano dati. I messaggi MIDI vengono trasportati quindi a livello grafico da connessioni sottili, poiché non veicolano informazioni audio.
Le tipologie di dati che vengono processate dagli oggetti possono essere segnali audio, messaggi e frame video.

3.5.4 *Patch*

Come già descritto, in Pure Data un programma eseguibile è definito come *patch*, assumendo quindi una concezione differente da quella classica dello sviluppo software, dove una *patch* è solitamente una serie di aggiornamenti successivi alla pubblicazione, atti a correggere bug o ad apportare modifiche e migliorie al software.

L'area all'interno della quale vengono disposti oggetti e connessioni è detta *canvas*; all'interno di questa area ad ogni step i dati sono immessi in un oggetto grazie alle connessioni fino al completamento della catena creata per mezzo delle stesse.
Ogni oggetto mantiene uno stato durante tutta l'esecuzione del programma, ma è possibile cambiarne la struttura durante il *runtime*, entrando nella modalità *modifica*, le cui operazioni effettuate hanno effetto immediato. Gli oggetti preposti per l'elaborazione dei messaggi restano in attesa fino a ogni nuova ricezione, quelli utilizzati per l'elaborazione dei segnali audio restano invece sempre attivi a meno che non sia esplicitamente inviato un comando di spegnimento. Le *patch* create con Pure Data possono quindi essere considerate sistemi *event driven*[16].



In figura 3.5 è possibile osservare in che modo l'interprete di Pure Data valuta l'ordine di esecuzione del flusso di dati.
In questo caso viene valutato per primo il ramo dell'albero costituito dagli oggetti più a destra e attraversato in profondità, in seguito quello di sinistra.

Figura 3.5: Pure Data dataflow

L'architettura di Pure Data si compone di diversi processi che sono attivi durante l'esecuzione di una *patch*. Il processo principale, detto *pd*, ha il compito di interpretare e di tradurre in linguaggio macchina ciò che avviene all'interno del *canvas* e di gestire le funzionalità audio, il processo *pd-gui* si occupa della gestione dell'interfaccia grafica, mentre il processo *pd-watchdog* monitora le risorse di sistema durante l'esecuzione di una *patch* ed eventualmente interrompe la stessa qualora le risorse di sistema richieste non fossero più disponibili. In figura 3.6 è possibile osservare l'architettura appena descritta

[16] Paradigma di programmazione nel quale il flusso dell'esecuzione del programma è scandito da eventi generati solitamente dall'interazione dell'utente con GUI o controllers esterni. Il Thread principale dell'applicazione è detto *main loop* il quale resta in attesa degli eventi per i quali resta in ascolto, i quali generano chiamate alle funzioni dette *callback*.

con la presenza di dispositivi input come tastiere MIDI e controller i quali si collegano al framework tramite le interfacce di gestione input/output. La presenza dei componenti di gestione del protocollo OSC già nella interfaccia input/output di Pure Data indica come sia stato possibile interfacciare i messaggi OSC generati dal software Android, implementato e descritto nel Capitolo 4, senza bisogno di utilizzare librerie o soluzioni esterne.

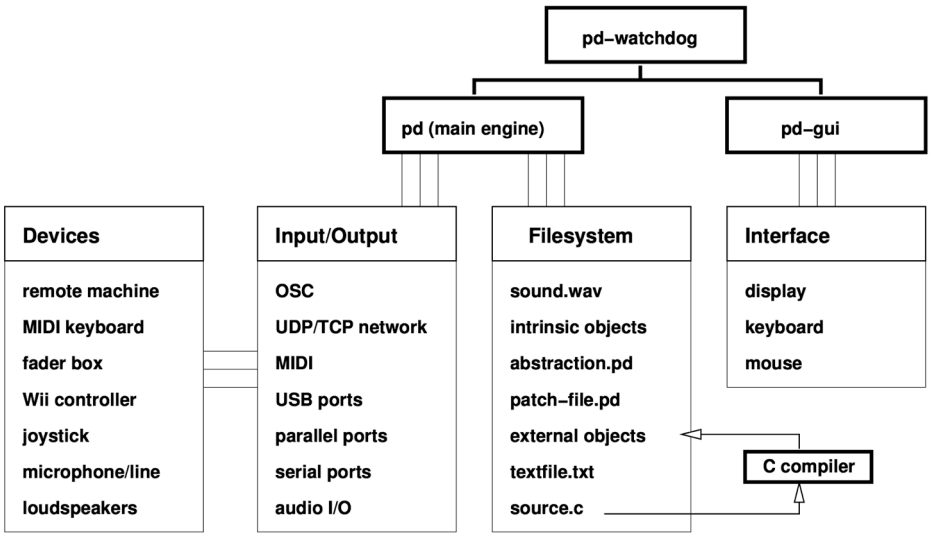


Figura 3.6: Architettura software di Pure Data

3.5.5 Gestione MIDI e OSC

Sono in seguito descritte le principali funzionalità di interfacciamento tra il protocollo MIDI e il framework Pure Data, grazie alle quali è stato sviluppato un *server* OSC per il *case study* proposto in Capitolo 4, tramite il quale vengono ricevuti ed inviati messaggi MIDI opportunamente inseriti in messaggi OSC.

Pure Data è in grado di collegarsi nativamente con i driver MIDI del sistema operativo sul quale è installato; in ambiente OS X (usato per lo sviluppo del prototipo descritto nel Capitolo 4), si interfaccia con il driver MIDI nativo IAC o direttamente con le sessioni, via network, di RTP MIDI, che saranno descritte in seguito.

I messaggi definiti nel protocollo MIDI sono rappresentati in Pure Data tramite oggetti, uno per ogni tipologia di messaggio *channel voice*, come mostrato in figura 3.7, ed un oggetto generico *midi in* / *midi out* per la categoria *system message*, come descritto in [24].

MIDI in object		MIDI out object	
Object	Function	Object	Function
notein	Get note data	noteout	Send note data.
bendin	Get pitchbend data −63 to +64	bendout	Send pitchbend data −64 to +64.
pgmin	Get program changes.	pgmout	Send program changes.
ctlin	Get continuous controller messages.	ctlout	Send continuous controller messages.
touchin	Get channel aftertouch data.	touchout	Send channel aftertouch data.
polytouchin	Polyphonic touch data in	polytouchout	Polyphonic touch output
polytouchin	Send polyphonic aftertouch.	polytouchin	Get polyphonic aftertouch.
midiin	Get unformatted raw MIDI	midiout	Send raw MIDI to device.
sysexin	Get system exclusive data	No output counterpart	Use midiout object

Figura 3.7: Oggetti Pure Data per la gestione in ingresso e uscita di messaggi MIDI

La sezione per la ricezione dei messaggi MIDI presenta gli oggetti Pure Data con i rispettivi *outlets*, nella parte inferiore degli oggetti, per la successiva elaborazione del *dataflow*, mentre la sezione per l'invio di messaggi MIDI presenta gli oggetti Pure Data con i rispettivi *inlets* in ingresso, nella parte superiore degli oggetti.

È possibile intercettare i messaggi in arrivo da un dispositivo MIDI collegato alla macchina oppure generare messaggi MIDI da inviare a dispositivi MIDI esterni o ad altri software tramite driver di interfacciamento e collegamento.

Come esiste il supporto per la gestione del linguaggio MIDI in Pure Data, esistono anche una serie di funzionalità per permettere l'utilizzo del protocollo OSC descritto in precedenza per lo scambio di metadati musicali attraverso la rete.

A differenza del supporto MIDI, quello per Open Sound Control non è presente nelle librerie del *core* native di Pure Data ma può essere trovato in diverse implementazioni aggiuntive. La versione estesa di Pure Data "Pd-extented" presenta al suo interno la libreria "mrpeach" sviluppata da Martin Peach, come descritto in [3] e in [11], che permette allo sviluppatore, una volta importata, tramite l'oggetto Pure Data *import mrpeach*, di accedere ad alto livello a tutte le funzionalità necessarie all'implementazione del protocollo OSC.

Le funzioni OSC specifiche incluse nella libreria *mrpeach* consistono nella possibilità di creare un messaggio OSC, a partire da un pacchetto OSC tramite l'oggetto *packOSC*, e di prelevarne i dati durante la ricezione tramite l'oggetto *unpackOSC*.
L'instradamento dei dati attraverso l'albero creato tramite gli indirizzi OSC avviene tramite l'oggetto *routeOSC*.
In seguito sono descritte le tipologie di dato associabili a un messaggio OSC (tabella 3.1) all'interno del framework di Pure Data con le relative sigle da utilizzare per la creazione e l'invio di una lista di valori tramite l'oggetto messaggio *sendtyped* che definisce a priori i tipi di dato da inviare all'oggetto *packOSC* (figura 3.8).

Tipologia di dati associabili come argomenti di un messaggio OSC	Sigla
Integer	i
Float	f
String	s
True	T
False	F

Tabella 3.1: OSC Data types

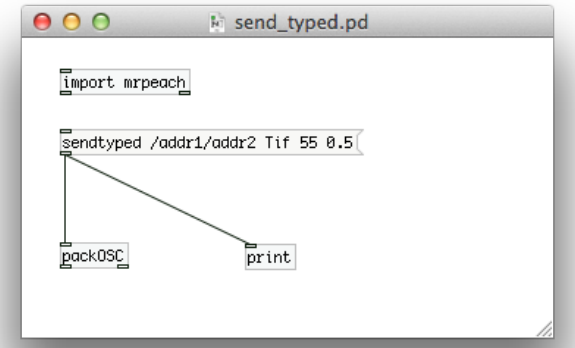


Figura 3.8: Creazione pacchetto OSC a partire dal messaggio sendtyped

La libreria *mrpeach* include anche oggetti per l'invio e la ricezione di dati sulla rete, con tipologia di connessione UDP o TCP. Gli oggetti responsabili per questa funzione sono le coppie *udpsend / udpreceive* e *tcpsend / tcpreceive.*

È necessario inviare un messaggio del tipo *connect* <*ip_address*> <*port*> al rispettivo oggetto *udpsend* o *tcpsend* prima di poter inviare dati. Per la disconnessione è necessario inviare il messaggio *disconnect*.
All'interno di una stessa rete LAN/WAN è possibile inserire sia l'indirizzo IP della macchina destinataria sia il nome associato. La porta predefinita per le comunicazioni via rete in Pure Data è la porta 9001, ma è in ogni caso possibile sceglierne una differente.
La scelta della tipologia di connessione UDP o TCP è fondamentale per la tipologia di applicativo che vuole utilizzare il protocollo OSC e va scelta di volta in volta in base alle necessità implementative. Nel Capitolo 4 è affrontato questo argomento e motivata la scelta effettuata per il *case study* proposto.
A differenza del protocollo MIDI, il protocollo OSC lascia il compito di definire il *namespace* dei pattern di indirizzo OSC allo sviluppatore.
In [46] è proposta una metodologia per la definizione di un *namespace* standardizzato chiamato "SynOSCopy" per la rappresentazione di informazioni musicali per il collegamento di sintetizzatori. Oltre alla completa implementazione dei messaggi del protocollo MIDI, vi sono parametri aggiuntivi utili nei casi in cui il suddetto protocollo non sia sufficiente a rappresentare tutte le funzionalità degli strumenti e sintetizzatori più moderni.

3.6 RTP / MIDI

Il *case study* proposto nel Capitolo 4 presenta, come descritto in precedenza, anche la possibilità di comunicare tramite il protocollo RTP (*Real Time Protocol*) per lo scambio di dati MIDI.
In questa sezione sono descritte le principali caratteristiche tecniche e funzionali di tale protocollo. RTP/MIDI non può essere considerato come un protocollo alternativo al MIDI, in quanto non vuole rappresentare una alternativa ma piuttosto una modalità di trasmissione particolarmente adatta al mezzo trasmissivo wireless. Il protocollo MIDI non è infatti ridefinito o modificato, viene semplicemente racchiuso all'interno della struttura definita per il trasporto, il *payload* RTP, come descritto in seguito.
Prima di analizzare il protocollo RTP MIDI è necessario analizzare il protocollo RTP, del quale il primo ne propone una estensione.

3.6.1 Real Time Protocol

Real Time Protocol è uno standard per la definizione di un formato per l'invio di informazioni audio e video attraverso una rete IP. L'area primaria di utilizzo di tale protocollo è quella delle tecnologie streaming come telefonia, video conferenze e servizi televisivi sulla rete. Tuttavia tale protocollo non garantisce un servizio che permetta garanzie sulla qualità del trasporto dati.

In parallelo al trasporto dei dati con il formato descritto in RTP, esiste un protocollo di supporto detto RTCP (*Real Time Control Protocol*) che veicola informazioni scalabili sulla qualità del servizio (QoS) e permette la sincronizzazione di più *stream* multimediali insieme [28].

Il protocollo RTP nasce per essere utilizzato con UPD a livello di trasporto ma risulta comunque utilizzabile anche con TCP.

La preferenza verso il protocollo di trasporto UPD nasce dal fatto che in molte applicazioni multimediali, la perdita di un pacchetto può non rivelarsi problematica, in quando risultante in una piccola porzione temporale del contenuto. Tuttavia non sono specificati algoritmi e tecnologie per supplire a tali perdite, perché è compito dello sviluppatore software, interessato a utilizzare RTP, implementare tali algoritmi sulla base delle statistiche di connessione fornite da RTP/RTCP [24]. Nelle tecnologie che coinvolgono streaming multimediali è solitamente preferibile una prestazione superiore sulla tempistica di arrivo dei pacchetti piuttosto che sulla congruenza di ricezione degli stessi.

Il protocollo RTP fornisce inoltre la possibilità di ridefinire la struttura del formato del pacchetto RTP, per estendere le proprie funzioni in diversi ambiti.

Come già descritto i componenti principali di RTP sono il protocollo di trasferimento dati (RTP) e il protocollo di controllo per le statistiche sulla sessione (RTCP) il quale occupa una porzione ristretta della banda rispetto al primo. In aggiunta sono definiti: un protocollo opzionale di *signalling*[17] ed uno per la descrizione dello stream[18].

Una sessione RTP consiste principalmente nell'invio di dati verso un indirizzo IP su due porte differenti, in genere contigue, per il flusso RTP e quello RTCP, il tutto ripetuto per ogni flusso multimediale, separando quindi stream audio e video.

[17] Protocollo per lo scambio di messaggi sull'inizio, la modifica e l'interruzione della sessione.
[18] Protocollo per l'invio di informazioni sullo stream della sessione. Tale protocollo non trasporta informazioni multimediali ma solamente metadati associati.

3.6.2 RTP MIDI Payload

Nel 2004 è stato presentato un documento preliminare per la definizione di un formato di pacchetto RTP per il trasporto delle infomazioni del protocollo MIDI [16], divenuto standard riconosciuto (RFC 4695) dall'Internet Engineering Task Force (IETF) nel 2006.
Nel formato proposto in [18] (RCF 6295), è possibile l'invio di tutte le tipologie di messaggi MIDI descritti nel documento di specifiche MIDI [32], fornendo una tecnologia in grado di supportare il controllo remoto di strumenti musicali con l'ausilio di strumenti per prevenire la perdita di pacchetti e per l'utilizzo anche in reti LAN o wireless.
I problemi riguardanti la perdita di pacchetti in una sessione MIDI, sono dovuti essenzialmente alla mancata ricezione di eventi come *note on* e *note off*. L'eventuale perdita crea artefatti nell'esecuzione e in particolare nel primo caso si definiscono come artefatti transienti, ovvero relativi al singolo evento mancato, mentre nel secondo caso vengono definiti come artefatti indefiniti, dove la perdita del pacchetto inficia per un tempo maggiore l'esecuzione. La consistenza temporale di un artefatto di tipo indefinito perdura fino a quando lo stesso messaggio perso inizialmente non è ricevuto una seconda volta, e in questo caso viene elaborato ripristinando lo stato che era andato perso. Le soluzioni proposte e descritte in seguito per il recupero di informazioni mancanti risultanti da perdita di pacchetto, sono tutte risolte senza tecnologie di acknowledgment o relative ritrasmissioni sulla rete.

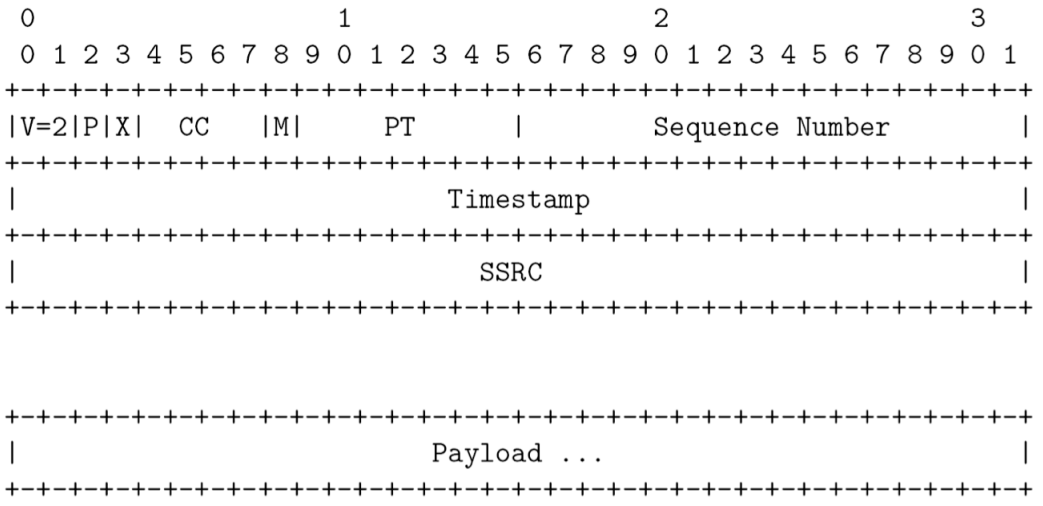


Figura 3.9: RTP packet

In figura 3.9 è mostrato graficamente il formato del pacchetto RTP. La personalizzazione per RTP MIDI avviene nella ridefinizione della struttura del *payload*.
IL *timestap* RTP permette di verificare la contiguità dei pacchetti ricevuti e quindi stabilire l'effettiva perdita di uno o più pacchetti.
La struttura del *payload* MIDI, in figura 3.10, è costituita da: header, MIDI list contenente i comandi MIDI, i valori temporali *delta time* ed infine uno spazio opzionale detto recovery journal[19].

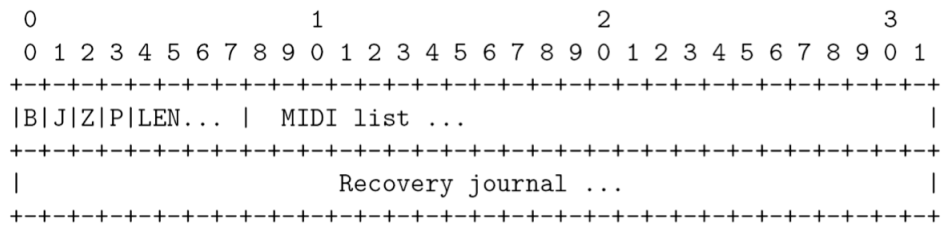


Figura 3.10: Payload RTP MIDI

Il campo LEN raffigurato in figura 3.10 rappresenta il numero di byte contenuti nella MIDI list, ovvero la lista di eventi midi contenuti in un pacchetto RTP. Il campo B può assumere valore "0" oppure "1"; nel primo caso indica che la MIDI list è formata al massimo da 15 byte, nel secondo caso può arrivare a contenere 4095 byte per pacchetto RTP, ottimizzando l'overhead generato dall'header del pacchetto e dalle altre informazioni aggiuntive in particolari casi di applicativi streaming.
La lista di eventi MIDI è formata da coppie di valore temporale detto *delta time* e dall'evento MIDI associato. Il campo Z, se presente con valore associato "1", indica che il primo delta time ha valore zero, altrimenti assume valore relativo al *timestamp* di ricezione dell'evento MIDI.
Anche per RTP MIDI, il protocollo RTCP fornisce statistiche dal destinatario al mittente circa la qualità del servizio e l'ultimo numero di sequenza ricevuto, presente nel campo EHSNR (Extended Highest Sequence Number Received), fornendo di fatto un feedback a bassa occupazione di banda su un percorso parallelo, senza l'introduzione di overhead temporale tipico delle connessioni TCP.

[19] Le tecniche di *journaling* associate ai file system permettono di tenere traccia delle modifiche da eseguire prima che vengano effettivamente eseguite. Questo permette di recuperare le operazioni non ancora eseguite dopo un evento critico come errori di sistema o interruzione dell'energia, con conseguente spegnimento prima dell'effettiva esecuzione di tutte le operazioni in coda.

Il *recovery journal* presente nel payload descritto in precedenza è un contenitore di informazioni aggiuntive opzionale: è compito del flag "J" del payload RTP MIDI specificarne o meno la presenza. Il *recovery journal* va inteso come un log dei soli eventi MIDI che possono generare artefatti, a partire da un determinato punto temporale detto *checkpoint*. È quindi il ricevente che, in base a un numero non congruente di pacchetto atteso è in grado, analizzando il *recovery journal*, di aggiornare la MIDI list in modo corretto, correggendo eventuali artefatti che si sarebbero altrimenti manifestati.
Il *recovery journal* è suddiviso in sezioni, dette capitoli, nei quali risiedono gli ultimi eventi MIDI, suddivisi per categoria, che vengono inseriti per l'eventuale recupero in seguito a un pacchetto perso.

Capitolo 4

Case Study: “Widi App”

Il prototipo sviluppato, come anticipato, consiste in un’applicazione per piattaforma Android che permette il collegamento USB di dispositivi MIDI, se presente il supporto OTG, fornendo funzionalità di connessione USB e la possibilità di effettuare l’indirizzamento dei messaggi MIDI ricevuti dal dispositivo esterno attraverso rete IP.
L’idea iniziale è stata quella di analizzare e implementare un’applicazione per la connessione dei dispositivi MIDI e successivo indirizzamento dei messaggi su reti LAN/WAN per piattaforma Android, considerando il fatto che, al tempo di scrittura di questo elaborato, non esiste ancora una metodologia standardizzata per questo scopo. Il fatto che tale applicativo standard non sia ancora presente, è correlato al fatto che le applicazioni dedicate alla sintesi sonora presenti sul market Google sono ancora in numero ristretto, se confrontate con quelle disponibili per piattaforma iOS, a causa della complessità nella gestione dell’audio in real time su piattaforma Android. Tuttavia uno degli esempi più notevoli è descritto in [48], con la simulazione di un sintetizzatore Yamaha DX7.

In particolare il caso d’uso che ha portato allo sviluppo di questo prototipo è quello relativo all’utilizzo dell’applicazione come ripetitore di messaggi MIDI, nel tentativo di emulare gli odierni dispositivi wireless MIDI dedicati, tuttora molto costosi sul mercato.
Durante l’implementazione si è tentato di ottenere i risultati migliori possibili in termini di latenza temporale; l’applicativo è stato in seguito testato al fine di valutarne l’effettiva possibilità di utilizzo per performance musicali in tempo reale.

L’applicazione è costituita da un *client* Android e un *server* OSC come descritto in Capitolo 3.5.
In questo paragrafo saranno descritti i dettagli implementativi principali e le strategie utilizzate per far fronte alle problematiche emerse durante lo sviluppo dell’applicazione.
Per lo sviluppo dell’applicazione Android è stato utilizzato l’ambiente di sviluppo Eclipse, opportunamente corredato degli strumenti forniti da Google per lo sviluppo.

In questo capitolo sono anche descritte le modalità di debugging seguite durante il lavoro.
Nel paragrafo 4.5.1 è descritto il *server* OSC implementato, utilizzato per ricevere i messaggi e reindirizzarli verso altre istanze dell'applicativo "Widi" oppure instradando i messaggi verso altri applicativi del sistema operativo in grado di ricevere messaggi MIDI sul quale è in esecuzione il server.
Nel paragrafo 4.6 sono descritte e approfondite le scelte riguardanti il design grafico dell'applicazione.

4.1 Funzionalità applicazione "Widi"

Come già anticipato, l'applicazione "Widi" permette di utilizzare dispositivi MIDI, attraverso il supporto OTG, su device Android.
L'applicazione guida l'utente in un processo di collegamento attraverso il quale vengono selezionati gli *endpoint*[20] USB del dispositivo collegato, in modo trasparente all'utente.
Poiché il supporto è fornito soltanto per i dispositivi midi che presentano già una interfaccia di tipo USB, è comunque possibile collegare strumentazione MIDI che utilizzi soltanto porte MIDI a 5 pin, come descritto in precedenza, tramite interfacce come quella mostrata in figura 4.1.

Figura 4.1: Interfaccia MIDI-USB Roland UM-ONE

Tale interfaccia è stata testata e utilizzata durante la fase di sviluppo ed è risultata compatibile con l'applicazione proposta.

[20] Collezione di interfacce di un dispositivo USB attraverso il quale avviene il flusso di dati della comunicazione.

Con questo strumento aggiuntivo, l'utente finale è quindi in grado di rendere compatibili con il protocollo USB anche dispositivi MIDI concepiti prima dell'avvento delle interfacce USB.
Una volta effettuato il collegamento con il dispositivo MIDI, tramite procedura grafica, è possibile scegliere quale tipo di tecnologia di indirizzamento wireless su rete LAN/WAN utilizzare per l'invio del flusso dati MIDI.
Le due implementazioni proposte consistono, come già descritto in precedenza, nell'utilizzo del protocollo OSC e quello definito dallo standard RTP MIDI.
La scelta di uno dei due protocolli porta a differenti schermate grafiche.
Selezionando il protocollo OSC, una successiva schermata mette in grado l'utente di selezionare quale host attualmente raggiungibile nello stesso network utilizzare per l'invio dei dati.
Selezionando invece il protocollo RTP MIDI, la schermata successiva presenta i controlli necessari per interfacciarsi al protocollo RTP MIDI Apple Bonjour, che permette il procedimento di discovery *ZeroConf* listando i canali RTP MIDI disponibili per la connessione.
A seguito di queste operazioni viene mostrata una schermata comune attraverso la quale è possibile mostrare o meno un log con i messaggi in arrivo dal dispositivo.
Attraverso un pulsante posto nella barra superiore è possibile in qualsiasi momento disattivare la connessione con il dispositivo MIDI.

4.2 Architettura client Android

Il client Android è stato sviluppato aderendo il più possibile al design pattern *model view controller*.
L'utilizzo di questo pattern permette una chiara distinzione tra i componenti dell'applicazione quali parti grafiche, classi di mantenimento e manipolazione dei dati, e i cosiddetti *controller*, ovvero quei componenti che si occupano di gestire le interazioni utente, modificare lo stato dei dati ed eventualmente aggiornare i componenti grafici.

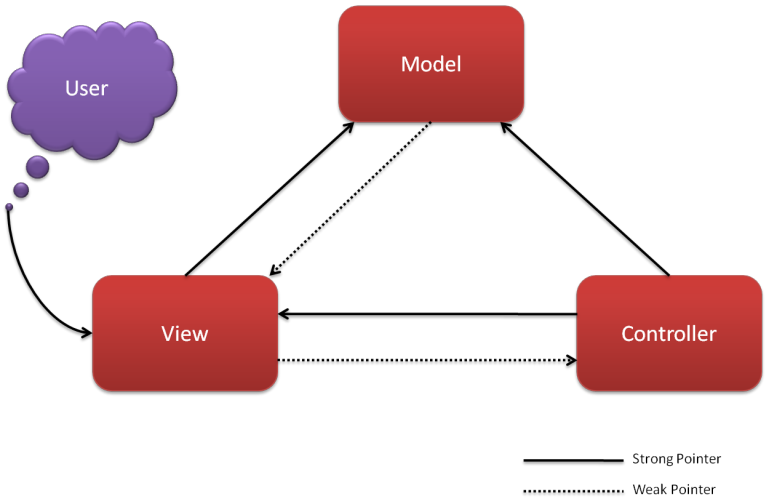


Figura 4.2: Schema pattern Model View Controller

In figura 4.2 è possibile osservare la struttura base dei componenti e la loro interazione secondo il pattern *model view controller*. Il pattern è spesso definito come *meta-pattern*, ovvero come insieme di più pattern. Le componenti grafiche, concepite con strutture gerarchiche, sono spesso indicate con il pattern detto *composite* [8]. Le componenti di model invece, ovvero le parti in cui risiedono i dati e la logica per la loro manipolazione, implementano il pattern *observer* [8], mediante il quale le classi grafiche *view* e i *controller*, vengono notificati di modifiche dello stato e possono agire di conseguenza.

Il vantaggio principale di questa suddivisione consiste nel poter modificare ogni componente in modo indipendente dagli altri, una volta definite delle specifiche interfacce di comunicazione.

Tuttavia essendo un pattern definito ad alto livello, spesso non è applicabile in modo completo per via del framework utilizzato o per via di esigenze particolari.

In Android, per esempio, la logica che definisce i controller non aderisce completamente alla definizione canonica del pattern *model view controller*. Infatti in questo caso esistono classi apposite, dette *Activity* in cui coesistono aspetti riguardanti le *view*, generate da appostiti file xml, e i *controller*.

La stessa sottile differenza si ritrova nel framework iOS dove si hanno le classi *ViewController* dove di fatto coesistono gli stessi componenti appena citati per il framework Android.

Tuttavia è buona norma implementare in modo separato controller dedicati a operazioni particolari non riguardanti la gestione grafica.

4.2.1 Package *view*

All'interno del package *view* sono contenute le classi di tipo *Activity* e *Fragment* [15] che descrivono uno o più task collegati a una interfaccia grafica. Una descrizione dettagliata nel paragrafo 4.6 spiega le scelte effettuate a livello di design grafico.
In particolare esiste una classe *Activity* che agisce da contenitore e di volta in volta aggiorna il proprio contenuto con classi di tipo *Fragment*, contenenti differenti interfacce grafiche, necessarie per la procedura grafica di collegamento del dispositivo MIDI.
Come definito in precedenza, all'interno delle classi del package *view* coesistono anche le funzionalità di controller, che gestiscono gli eventi generati dalla interfaccia grafica.

4.2.2 Package *model*

Nel package model si trovano le classi necessarie alla descrizione degli oggetti utilizzati nel progetto e alla definizione delle strutture dati principali.
In particolare in questo package è stata inserita la libreria utilizzata per la gestione dei messaggi OSC. Nel paragrafo 4.5.1 è descritta in dettaglio tale libreria e le ottimizzazione che sono state implementate per renderla utilizzabile nel progetto.
Inoltre è presente una classe *Constants* per il recupero statico di alcune costanti globali durante l'esecuzione dell'applicazione ed una classe *WidiDataModel* che permette il salvataggio in memoria di dati durante il runtime, implementata secondo il pattern di programmazione *singleton*. L'implementazione secondo questo pattern prevede che sia creata una sola istanza della classe, raggiungibile in modo univoco in ogni punto dell'applicativo.
Nel momento in cui diversi oggetti richiedono questa classe, si è certi che l'istanza della classe a cui si accede sia sempre la stessa. In questo modo si evita duplicazione di dati e accesso non congruente alle strutture dati.
Una volta compreso lo scopo della creazione di un'unica istanza della classe, sorge il problema di gestire come questo oggetto debba essere creato la prima volta e in che modo sia possibile accedervi successivamente.
Esistono diverse metodologie, che, a fronte del raggiungimento dello stesso scopo, si comportano in modo più o meno corretto nei casi di multithreading.

Per fare in modo che in ambiente multithread l'instanza dell'oggetto sia creata una sola volta, si tende a sincronizzare il metodo che restituisce l'istanza stessa. In questo modo la prima volta che il metodo costruttore viene chiamato, viene creata l'istanza e da quel momento verrà sempre restituita quest'ultima, grazie ad un controllo posto come prima istruzione all'interno del metodo.
Tuttavia la sincronizzazione del metodo costruttore introduce un overhead temporale per la completa esecuzione delle istruzioni. Una possibile ottimizzazione è fornita dalla metodologia definita come doppio controllo: in questo caso viene sincronizzata la classe stessa solo nel caso della creazione, non sulle successive restituzioni dell'istanza.
Tuttavia nel prototipo proposto si è scelto di agire in modo differente, descritto qui in seguito.
Poiché l'istanza della classe *WidiDataModel* risulta necessaria già dai primi istanti di esecuzione dell'applicazione, non si ottiene un vantaggio dall'attesa della creazione dilazionata nel momento della prima effettiva richiesta di utilizzo.
La tipologia scelta di creazione dell'istanza è quella definita come *eager initialization* [8].
Utilizzando questa tecnica, l'istanza della classe viene già creata nella definizione della variabile associata all'oggetto della classe stessa.

```
Public class Singleton {
          private static final Singleton INSTANCE = new Singleton();
          private Singleton() {}
          public static Singleton getInstance() {
                    return INSTANCE;
          }
}
```

Figura 4.3: Pattern singleton eager initialization

In figura 4.3 è possibile osservare come già a livello di classe sia creata l'istanza dell'oggetto java e come il metodo *getInstance*, che restituisce sempre la stessa, non sia sincronizzato ed inoltre non effettui nessun controllo poiché si è certi che l'oggetto sia già stato creato.
La mancanza della sincronizzazione appena descritta è utilizzabile solo nei casi in cui si è sicuri che differenti thread non vadano a modificare in modo concorrente le strutture dati, ma nel caso del prototipo proposto questo non può avvenire.

Il vantaggio principale in questa scelta è che, a fronte di un overhead temporale maggiore nel momento in cui la *JVM*[21] carica la classe poiché è già necessaria la creazione dell'oggetto, tuttavia la mancanza di controlli e sincronizzazione sul metodo *getInstance* rendono in seguito l'accesso all'oggetto più veloce.

4.2.3 Package *communication*

Le principali tecniche di networking utilizzate sono descritte nel paragrafo 4.5. Qui è presentata la struttura generale del package.
In questo package sono incluse le classi che racchiudono le funzionalità di networking per l'invio dei messaggi MIDI in ricezione da dal device MIDI.
In particolare sono presenti 4 classi:

- ***OSCTransmitter***
- ***AsyncOSCWrapper***
- ***HostDiscover***
- ***ConnectionManager***

La classe *OSCTransmitter* si occupa dell'invio dei messaggi MIDI OSC attraverso una porta OSC rappresentante una socket per invio dati tramite UDP.
La classe *AsyncOSCWrapper* rende l'invio dei dati asincrono tramite l'utilizzo dell'oggetto *AsyncTask* fornito dalla SDK Android e descritto in paragrafo 4.5.1.
La classe *HostDiscover* contiene le funzioni necessarie per listare altri dispositivi connessi alla stessa rete LAN/WAN.
Infine la classe *ConnectionManager* viene utilizzata per verificare lo stato di connessione del dispositivo sulla rete WIFI.

4.2.4 Package *test*

Questo package contiene una classe specifica per effettuare una misurazione del tempo necessario a ricevere un messaggio MIDI e instradarlo verso la classe *OSCTrasmitter* per l'invio via wireless. Il procedimento di testing è descritto nel Capitolo 5.

[21] Java Virtual Machine.

4.3 Sviluppo e Debug

Per lo sviluppo dell'applicazione Android è stato utilizzato il software Eclipse, opportunamente dotato delle estensioni fornite dalla SDK Android.
Per il debug e il testing è stato utilizzato un device fisico (Samsung Galaxy Nexus), poiché tramite emulatore non sarebbe stato possibile sfruttare le caratteristiche di collegamento USB.
Considerando la particolarità dell'applicazione, per la quale è necessario collegare via USB un dispositivo MIDI, non è stato possibile usufruire della modalità di debug via terminale classica fornita dal *tool* ADB[22]. Il debug su Android prevede infatti che il device sia collegato via USB alla macchina sulla quale si sviluppa il codice sorgente.
Tuttavia esiste una modalità di debug wireless, grazie alla quale è possibile installare i pacchetti applicazione e monitorarne l'esecuzione. In questo modo è stato possibile "occupare" la porta USB con i dispositivi MIDI e contemporaneamente monitorare via wireless l'esecuzione dell'applicazione.
I comandi per collegarsi via wireless al device, tramite terminale sono i seguenti:

1. **adb tcp/ip <port_number>** : dichiara la nuova modalità di connessione tcp/ip sulla porta definita
2. **adb connect <ip_number> : <port_number>** : collega tramite la modalità dichiarata sopra il dispositivo al indirizzo <*ip_number*> sulla porta <*port_number*>.
3. **adb usb** : disabilita le connessioni in modalità tcp/ip

[22] ADB: Android Debug Bridge. È un tool da linea di comando fornito con SDK Android che permette di comunicare con una istanza di un emulatore Android o con un dispositivo connesso via USB. Le informazioni di debug vengono visualizzate sulla console tramite il comando "adb logcat".

4.4 Driver MIDI

In questo paragrafo è descritto il funzionamento del driver software implementato per interfacciare i dispositivi MIDI con Android, grazie alle potenzialità fornite dal supporto OTG descritto in precedenza.
L'insieme delle classi che compongono il driver MIDI ha la funzione di specificare in che modo il device MIDI si interfaccia con il resto del client.
La classe principale che descrive il device MIDI è *UsbMidiDevice*. Al suo interno vengono utilizzati i tool forniti dalla SDK Android per il collegamento USB, con l'aggiunta delle funzionalità MIDI specifiche. Una volta collegato il dispositivo viene eseguito un thread asincrono che contatta il database online [37] ed è in grado di tradurre, se presenti, le informazioni di *vendor* e modello in modo *human readable*.
Per sfruttare questa funzionalità è necessario un collegamento attivo ad internet e quindi non solo un router che crei una rete locale.
Android fornisce la possibilità alle singole applicazioni di notificare l'utente quando un dispositivo USB viene connesso con la conseguente richiesta di attivazione dell'applicazione. È inoltre possibile specificare per quale categoria di dispositivo USB essere notificati in base alle specifiche definite in [9] e in [39].
Tuttavia si è scelto di non fornire all'applicazione questa funzionalità per focalizzare l'attenzione dell'utente sulla procedura grafica di connessione al dispositivo MIDI.
Infatti l'utente è portato a collegare il dispositivo e premere un pulsante per iniziare la connessione. Successivamente, attraverso il driver, appare in un apposita finestra *dialog* a *pop-up* il dispositivo con *vendor* e *nome* aggiornati, se trovati sul database online citato in precedenza.
Attraverso la procedura di connessione vengono quindi recuperati gli *endpoint* per la comunicazione da e verso il dispositivo connesso tramite le classi *UsbMidiInput* e *UsbMidiOutput*. L'utente deve selezionare gli ingressi e le uscite MIDI prescelte, presentati attraverso due differenti *view* selettore che compaiono in successione.
Una volta selezionati il dispositivo è collegato e pronto a inviare e ricevere messaggi attraverso la porta USB OTG.
Una volta effettuata la connessione, è salvato in memoria un riferimento al dispositivo tramite l'oggetto della classe *UsbMidiDevice*, nella classe *WidiDataModel* descritta in precedenza.

In seguito alla selezione dei canali in ingresso e uscita verso il dispositivo, sono salvate sempre nella classe *WidiDataModel* anche gli oggetti *UsbMidiInput* e *UsbMidiOutput*, interfacce di comunicazione con il dispositivo USB.

I messaggi in arrivo dal dispositivo sono processati da un thread in ascolto che, grazie ad una apposita classe effettua il parsing dei messaggi MIDI e per ogni tipologia di messaggio MIDI ricevuto, rimanda la gestione dell'evento alla classe *MidiReceiver* tramite *callback*[23]. Il vantaggio di questo approccio è la possibilità di personalizzare il comportamento della gestione dei messaggi ricevuti; la logica della manipolazione dei dati ricevuti dal dispositivo MIDI viene definita infatti nella classe *MidiReceiver*. La gestione degli eventi si traduce con la logica d'invio dei dati in base al protocollo di comunicazione scelto, OSC o RTP MIDI.

[23] Tramite il procedimento di *callback* viene passato un blocco di codice come parametro di una funzione. Tale funzione ha il compito di eseguire il blocco di codice con gli altri parametri con la quale è stata chiamata.

4.5 Networking

In questo paragrafo sono descritti gli aspetti implementativi delle tecniche di networking utilizzate per l'invio dei dati dal client Android verso i possibili destinatari e alcune ottimizzazioni finalizzate alla minimizzazione della latenza d'invio.
I messaggi possono essere inviati infatti ad altre istanze dell'applicazione "Widi", verso il *server* OSC, oppure verso un applicativo in grado di supportare la tecnologia RTP MIDI (Apple/Bonjour).

4.5.1 MIDI / OSC

La prima modalità di networking implementata per il progetto è stata quella relativa al protocollo OSC. Le principali funzionalità che il progetto deve avere per poter implementare tale protocollo sono la definizione dei messaggi OSC e la modalità di invio su rete IP tramite il protocollo UDP o TCP.
Essendo l'applicazione "Widi" pensata per scambiare dati su una rete locale, si è deciso di utilizzare il protocollo UDP, più veloce nell'instradamento dei messaggi, e di testarne in seguito l'utilizzabilità in termini di perdita di pacchetti e latenza.
Il primo approccio testato è stato quello di utilizzare la libreria *libpd*, ovvero un porting del software Pure Data per Android. Tuttavia la versione *libpd*, traduzione della versione *Vanilla* di Pure Data, non racchiude al suo interno molte delle librerie aggiuntive di comune utilizzo come, ad esempio, la libreria *mrpeach* descritta in precedenza. In sostanza non è stato possibile utilizzare, con la versione base di *libpd*, i componenti per la compatibilità con il protocollo OSC. Per questo motivo sono stati compilati per Android i file in linguaggio C della libreria *mrpeach*, grazie allo strumento NDK[24] fornito dall'SDK Android, come descritto in [2]. Una volta ottenuti questi strumenti si è rivelato complesso sviluppare un'interfaccia per la comunicazione dei mesaggi MIDI verso il layer *libpd*; dovendo inoltre gestire l'invio e la ricezione dei dati su rete IP, sebbene funzioni presenti in Pure Data, si è scelto di implementare tali funzioni in codice nativo java.

[24] Il framework NDK fornisce un set di tool che permettono di implementare parti di codice in linguaggio nativo come C e C++. Inoltre forniscono i metodi per compilare librerie in linguaggio macchine per poter essere utilizzate dalle applicazioni Android.

Prima di analizzare l'implementazione delle funzionalità di networking tramite il protocollo OSC è brevemente descritta la logica che permette di effettuare il collegamento tra l'applicazione Android e un'altra istanza dell'applicazione o un altro software in grado di manipolare i dati OSC.

Una volta effettuato il collegamento del dispositivo e selezionato il protocollo OSC per l'invio dei dati, come descritto in Appendice A, viene mostrata una lista con gli host selezionabili per l'invio dei messaggi OSC.

La lista è aggiornata in modo pressoché istantaneo da due componenti che lavorano in modalità asincrona e in parallelo, fornendo una esperienza utente superiore alla possibilità di inserimento manuale dell'indirizzo IP di output.

Nella classe *WidiDataModel* è presente una struttura dati in cui vengono inseriti tutti gli indirizzi delle macchine contattabili nella sottorete.

Il primo componente *HostDiscover* si occupa di scansionare tutti gli indirizzi IP della sottorete e, una volta verificata o meno l'esistenza di una macchina raggiungibile presso tale indirizzo, viene opportunamente aggiornata la struttura dati presente in *WidiDataModel*.

La procedura di scansione degli indirizzi IP è resa asincrona rispetto al thread principale da un oggetto di tipo *AsyncTask*.

Il secondo componente *NdsHelper* scansiona la sottorete cercando gli host che sono si sono registrati per il servizio di gestione OSC tramite il protocollo *ZeroConf*, conosciuto anche come *Rendezvous* o *Bonjour* e proposto in [29].

In [19] è proposto un framework distribuito per performance musicali tra diversi host utilizzando MAX/MSP; anche in questo caso viene citato il protocollo *ZeroConf* con messaggi OSC per il processo di discovery e ritenuto funzionale per tale scopo. Tuttavia in questo caso, data la complessità dei messaggi per la sessione, è stato utilizzato un bus multicast UDP sia per il processo di discovery che per l'invio dei messaggi di sessione.

Il protocollo standard IETF *ZeroConf* [30] vieneco utilizzato per la configurazione dinamica dei nodi di una rete IP. L'idea alla base del protocollo è di creare una metodologia per la connessione automatica di macchine sotto la stessa rete IP registrate per fornire un determinato servizio. Un esempio è ciò che accade con l'utilizzo delle stampanti di rete, che in automatico vengono listate nella sottorete.

In questo modo ogni istanza dell'applicazione si registra per il servizio di gestione di messaggi OSC tramite protocollo UDP tramite la sigla *_osc._upd* come proposto in [29] e

in [7]. In questo modo può essere vista da altre istanze dell'applicazione o altri software in grado di scansionare la rete per la ricerca del servizio *_osc._udp*. Inoltre, dopo aver effettuato la registrazione al servizio, tramite il componente *NsdHelper* viene scansionata la sottorete in cerca di altri host in grado di fornire tale servizio. Una volta trovati eventuali host che forniscono il servizio, viene aggiornata in modo asincrono la struttura dati presente in *WidiDataModel* contenente gli host raggiungibili con l'indirizzo IP e la stringa "OSC Service", in modo che l'utente venga notificato del fatto che sicuramente tale host avrà la capacità di elaborare i messaggi OSC inviati.

Con i due componenti *NsdHelper* e *HostDiscover*, si ottiene quindi una lista aggiornata di host raggiungibili, dove è evidenziata l'informazione degli host che sono registrati per il servizio OSC. Si è scelto di listare comunque gli host raggiungibili anche nel caso in cui non fossero registrati al servizio OSC poiché alcuni software potrebbero non implementare il protocollo *ZeroConf* ma essere comunque in grado di elaborare i messaggi OSC. È compito dell'utente selezionare l'host che ritiene adatto al corretto funzionamento dell'applicazione.

L'aggiornamento della struttura dati degli host avviene in modo da evitare duplicati nel caso in cui sia il componente "HostDiscover" e sia "NsdHelper" rilevino lo stesso indirizzo IP valido.

Una volta registrati al servizio OSC diversi host è possibile osservare in figura 4.4 i servizi disponibili nella sottorete tramite un apposito software, in questo caso "Bonjour Browser" per piattaforma OS X.

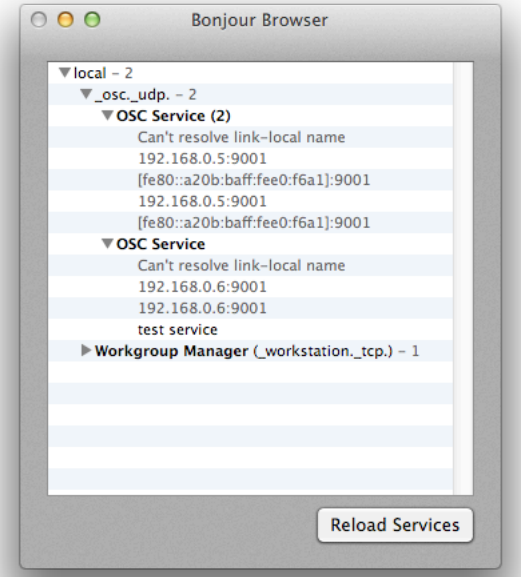


Figura 4.4: Bonjour Browser e i servizi *_osc._udp* nella sottorete

Sono in seguito descritti i dettagli implementativi per l'invio dei dati tramite protocollo OSC.

Uno strumento che si è rivelato molto utile in questa fase è stata la libreria java open source di classi *javaosc* presentata in [50] e che implementa già tutte le funzionalità OSC sopracitate.

Prima di descrivere alcune modifiche e tentativi di ottimizzazione implementati sulla base della libreria *javaosc*, ne sono descritti ora i principali componenti.

Le classi che lavorano ad alto livello sono *OSCMessage*, *OSCPortIn* e *OSCPortOut*.

OSCMessage rappresenta un messaggio OSC singolo, non inserito in un *bundle* di messaggi; per i *bundle OSC* esiste la apposita classe *OSCBundle* che presenta al suo interno la possibilità di inserire *tag* temporali per l'ordine di esecuzione da parte di un *server* OSC. Il costruttore della classe *OSCMessage* prevede l'inserimento dell'*address* OSC e uno o più argomenti opzionali, inseribili anche in seguito alla costruzione dell'oggetto. *OSCPortIn* e *OSCPortOut* estendono la classe astratta *OSCPort* e rappresentano rispettivamente una porta in ingresso per la ricezione e una porta in uscita attraverso la quale inviare messaggi OSC.

OSCPortIn rappresenta un oggetto in grado di restare in ascolto tramite thread, grazie all'interfaccia java *Runnable*, dell'arrivo di messaggi OSC con un determinato *address*

OSC. Una volta creato l'oggetto con il riferimento alla porta di ascolto e aver settato per quale indirizzo OSC monitorare i messaggi in ingresso, è possibile far partire il thread di ricezione.

OSCPortOut rappresenta un oggetto in grado di inviare un messaggio OSC verso un indirizzo IP su di una determinata porta. Il costruttore di questa classe prevede quindi come parametri l'indirizzo destinatario d'invio, incapsulato come oggetto *InetAddress*[25] e il numero della porta associato.

Il metodo *send* si occupa di convertire in byte le informazioni relative all'indirizzo e agli argomenti del messaggio OSC da inviare, e di instradarle attraverso una *socket*[26], opportunamente inserite in un oggetto java *DatagramPacket*. L'oggetto *DatagramPacket* è utilizzato per creare un servizio d'invio dati senza connessione (UDP). Ogni messaggio è instradato da una macchina ad un'altra basandosi soltanto sulle informazioni contenute all'interno del pacchetto; i pacchetti potrebbero effettuare strade differenti per arrivare a destinazione e potrebbero arrivare in ordine differente rispetto all'invio o essere persi [22].

L'oggetto *DatagramPacket* viene inviato tramite un oggetto della classe *DatagramSocket* che rappresenta in java una *socket*.

Nella piattaforma Android, qualora si voglia usufruire di tecniche di networking per l'invio e la ricezione dei dati, tali procedure non dovrebbero essere eseguite nello stesso thread dell'interfaccia grafica. Esiste una modalità attivabile via codice detta *StrictMode* tramite la quale è possibile interagire con la rete nello stesso thread dell'interfaccia grafica; è tuttavia consigliato soltanto per scopi di debug.

È stato necessario quindi trovare una metodologia che potesse rendere le funzioni di invio e ricezione asincrone rispetto al processo principale.

È stato necessario quindi scegliere una modalità di invio dati tramite la classe *OSCPortOut* asincrona rispetto al thread principale.

Si è scelto di suddividere la logica di invio dati presente nella classe *OSCTransmitter*, presente nel package *communication*, dalla logica di esecuzione asincrona, in modo da poter in futuro ottimizzare ogni componente in modo indipendente.

[25] La classe "Inet Address" rappresenta un indirizzo IP, permette la costruzione dell'oggetto associato tramite indirizzo IP sotto forma di stringa.

[26] Endpoint accessibile via API per la comunicazione bidirezionale attraverso reti IP, i cui parametri principali sono indirizzo IP e numero di porta associato.

La classe *OSCTransmitter* si occupa quindi soltanto di inviare un messaggio OSC attraverso una porta tramite un oggetto della classe *OSCPortOut*, il cui riferimento è salvato univocamente nella classe *WidiDataModel*.
La classe *OSCTransmitter* è utilizzata in modo asincrono rispetto al thread principale tramite la classe *AsynchOSCWrapper*, che rappresenta un task asincrono, estendendo la classe Android *AsyncTask*.
Anche la classe *OSCTransmitter* è implementata secondo il pattern singleton; questo permette di non dover ricreare ad ogni invio messaggio l'oggetto associato, ma ne è utilizzata sempre la stessa istanza.
Anche in questo caso, come per la classe *WidiDataModel*, l'istanza è creata a tempo di compilazione della classe, eliminando la necessità di controllarne l'esistenza di volta in volta, rendendone quindi di fatto più veloce l'accesso. Questo si traduce in un minor tempo necessario per l'invio dei messaggi OSC, considerata l'esigenza di invio dati con la minor latenza possibile per la performance in *pseudo real time*.
La classe astratta *AsyncTask*, una volta implementata, permette di ridefinire alcuni metodi che ad alto livello gestiscono operazioni in background. È possibile eseguire codice prima e dopo che vengano effettuare le operazioni in background; è altresì possibile notificare l'interfaccia grafica circa i progressi del task asincrono [15].
Considerata la sola necessità di rendere asincrono l'invio del messaggio OSC tramite socket, è stato reimplementato soltanto il metodo per l'invio asincrono denominato *doInBackground*.

Sono ora presentate le modifiche che sono state effettuate alla libreria *javaosc* [50]. Inizialmente la libreria OSC per java è stata utilizzata per creare un messaggio OSC, oggetto della classe *OSCMessage*, per ogni messaggio ricevuto dal dispositivo MIDI collegato via USB. Una volta creato l'oggetto OSC e inviato, tale oggetto non era quindi più utilizzato nel codice.
Questo fatto si traduceva nella creazione di un numero molto elevato di oggetti java, considerando che in una performance musicale, a ogni pressione di un tasto, o conseguente rilascio, viene generato un messaggio MIDI. In una performance di alcuni minuti, si

raggiunge spesso un numero di messaggi nell'ordine di qualche migliaio di messaggi, con la conseguente impossibilità di gestione delle funzionalità di *garbage collection*[27] di java.
Si è quindi cercato di fare in modo che fosse generato un solo messaggio per ogni tipologia di evento MIDI, e aggiungere la capacità alla classe *OSCMessage* di svuotare le proprie strutture dati, per poter riutilizzare lo stesso oggetto più volte.
Sono ora mostrate le porzioni di codice riguardanti la classe *OSCMessage* modificata con un metodo per lo svuotamento delle strutture dati e l'interfaccia *MidiReceiver* che gestisce le funzioni di *callback* a seguito della ricezione dei messaggi MIDI dal dispositivo.

[27] In informatica per garbage collection (letteralmente *raccolta dei rifiuti*, a volte abbreviato con GC) si intende una modalità automatica di gestione della memoria, mediante la quale un sistema operativo, o un compilatore e un modulo di run-time, liberano le porzioni di memoria che non dovranno più essere successivamente utilizzate dalle applicazioni.

```
package com.unimi.lim.widitest.model.osc;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.LinkedList;
import java.util.List;
import com.unimi.lim.widitest.model.osc.utility.OSCJavaToByteArrayConverter;

public class OSCMessage extends OSCPacket {
      //variabile stringa per l'indirizzo OSC
       private String address;
       //struttura dati per gli argomento del messaggio OSC
      private List<Object> arguments;
      //costruttore : istanzia un oggetto messaggio OSC vuoto
      public OSCMessage() {
            arguments = new LinkedList<Object>();
      }
      //costruttore : istanzia un oggetto messaggio OSC con un indirizzo OSC
      public OSCMessage(String address) {
            this(address, (Collection<Object>) null);
      }
       //costruttore : istanzia un oggetto messaggio OSC con un indirizzo OSC e u
       collection di argomenti
      public OSCMessage(String address, Collection<Object> arguments) {
            this.address = address;
            if (arguments == null) {
                  this.arguments = new LinkedList<Object>();
            } else {
                  this.arguments = new ArrayList<Object>(arguments);
            }
            init();
      }
      //metodo getter per la variabile privata address
      public String getAddress() {
            return address;
      }
      //metodo pubblico aggiunto per poter svuotare la struttura dati degli argomenti
      public void clearData(){
            arguments.clear();
      }
      //metodo setter per la variabile privata address
      public void setAddress(String address) {
            this.address = address;
      }
      //metodo per l'aggiunta di argomenti (qualora fosse stato istanziato senza argomenti)

      public void addArgument(Object argument) {
            arguments.add(argument);
      }
                                        [ . . . ]
```

La logica di riutilizzo degli oggetti *OSCMessage* è racchiusa quindi nel metodo *clearData*.
Infine si è scelto di creare un messaggio OSC per ogni tipologia di messaggio MIDI implementato e di aggiornare a ogni nuovo evento MIDI in arrivo il contenuto degli argomenti.
In questo modo per la categoria di messaggi *channel voice* vengono creati solo 7 messaggi OSC e riaggiornati di volta in volta con i dati corretti.
Inoltre si è deciso di inviare i dati in modalità seriale, come avviene nel MIDI via cavo, e di sincronizzare i metodi di invio in modo da evitare modifiche concorrenti allo stesso oggetto OSC. In questo modo un nuovo evento MIDI in arrivo dovrà attendere che il precedente messaggio sia stato inviato prima di aggiornare la struttura dati del messaggio OSC, evitando problemi di concorrenza multipla.
È ora mostrata la classe anonima java che implementa l'interfaccia *MidiReceiver*, dove risiede la logica appena descritta. Considerando la similitudine tra i metodi per la gestione dei differenti messaggi MIDI, è mostrato soltanto il metodo per i messaggi *note on* MIDI.

```
// classe anonima java che implementa l'interfaccia MidiReceiver per la gestione degli
// eventi MIDI

private final MidiReceiver receiver = new MidiReceiver() {

        // messaggi OSC istanziati una volta sola per ogni messaggio channel voice
        // con il relativo indirizzo OSC

        private OSCMessage noteOnMessage = new OSCMessage("/midi/noteon");
        private OSCMessage noteOffMessage = new OSCMessage("/midi/noteoff");
        private OSCMessage controlChangeMessage = new OSCMessage("/midi/cc");
        private OSCMessage polyMessage = new OSCMessage("/midi/pa");
        private OSCMessage programChangeMessage = new OSCMessage("/midi/pc");
        private OSCMessage pitchBendMessage = new OSCMessage("/midi/pb");

        // wrapper asincrono per l'invio dei messaggi tramite la classe OSCTransmitter

        private AsyncOSCWrapper theWrapper;

        // metodo per la gestione dell'evento MIDI note on

        // metodo sincronizzato per l'invio del relativo messaggio OSC
        // altre chiamate al metodo devono attendere l'esecuzione
        // per evitare concorrenza e modifica inattesa dei parametri
        // del messaggio OSC prima dell'invio

        @Override
        public synchronized  void onNoteOn(int channel, int key, final int velocity) {

                // aggiunta argomenti al messaggio OSC

                noteOnMessage.addArgument(key);
                noteOnMessage.addArgument(velocity);
                noteOnMessage.addArgument(channel);

                try {
                        // creazione ed esecuzione del wrapper asincrono AsyncTask

                        theWrapper = new AsyncOSCWrapper();

                        // il metodo get() invocato sull'esecuzione del wrapper
                        // forza l'attesa dell'esecuzione del codice relativo al
                        // metodo execute, permettendo l'invio seriale dei messaggi

                        theWrapper.execute(noteOnMessage).get();

                        // una volta inviato il messaggio viene svuotata la struttura
                        // dati del messaggio OSC e l'oggetto può essere riciclato
                        // per le successive chiamate al metodo onNoteOn()

                        noteOnMessage.clearData();

                        // gestione possibili errori dovuti alla non esecuzione del wrapper
                        // asincrono con stampa sulla UI tramite metodo toast(String s)

                } catch (InterruptedException e) {
                        toast("Error while sending message ...");
                e.printStackTrace();
                } catch (ExecutionException e) {
                        toast("Error while sending message ...");
                        e.printStackTrace();
                }


                                        [ . . . ]
```

Con le metodologie descritte sono stati risolti diversi problemi iniziali di perdita di messaggi che in realtà erano causati dalla modifica inattesa dei parametri OSC prima dell'effettivo invio del messaggio.

4.5.2 Server OSC

Sono ora descritte le principali caratteristiche del *server* OSC implementato per ricevere i messaggi su un computer, con compatibilità multipiattaforma (Windows, Mac OS X, Linux), ed eseguire l'indirizzamento dei messaggi verso altri software.

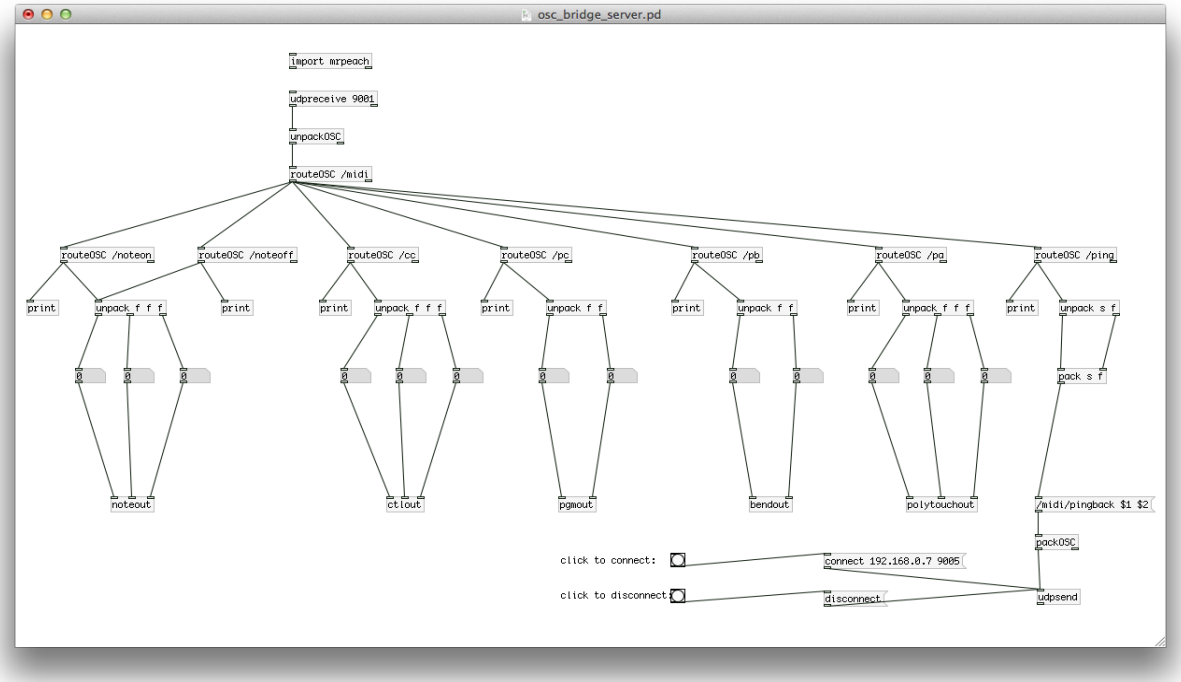


Figura 4.5: *Server* OSC UI

In figura 4.5 è possibile osservare la struttura grafica ad albero del *server* OSC implementato in Pure Data. Come già descritto, la struttura grafica della *patch* è strettamente correlata agli aspetti implementativi, poiché Pure Data è un linguaggio di programmazione visuale.
Il primo oggetto presente è quello necessario per importare la libreria aggiuntiva *mrpeach*, che permette di utilizzare oggetti per la gestione del protocollo OSC e invio dati via rete.

In seguito grazie all'oggetto *udpreceive* la *patch* resta in ascolto di pacchetti UDP sull'interfaccia di rete alla porta 9001, aggiunta nella dichiarazione dell'oggetto stesso.
La diramazione del flusso dati dell'albero costituito dagli oggetti Pure Data, prosegue con l'instradamento dei dati verso l'oggetto preposto all'invio in uscita dei messaggi MIDI.
I messaggi OSC in arrivo sull'interfaccia di rete alla porta 9001, sono instradati verso l'oggetto *unpackOSC* che permette di inoltrare i messaggi in base al loro indirizzo OSC. Una serie di oggetti *route* in successione instradano i messaggi OSC verso differenti oggetti *unpack*; l'indirizzo "/midi" è comune a tutti i messaggi, mentre l'indirizzo seguente indica il tipo di messaggio MIDI.
Gli oggetti *unpack*, uno per ogni tipologia di messaggio MIDI, instradano il numero esatto di parametri presenti nel messaggio in arrivo, verso gli ultimi oggetti del *path*, foglie dell'albero, che sono in grado di creare e spedire i messaggi MIDI verso l'interfaccia MIDI in uscita, selezionata nelle impostazioni del software Pure Data.
L'unico messaggio non indirizzato verso un oggetto per le uscite MIDI, è quello più a destra in figura 4.5 che, grazie all'oggetto *route /ping* e i successivi oggetti del sottoramo, spedisce nuovamente il messaggio verso il mittente per scopo di testing e misurazione della latenza del mezzo trasmissivo. La procedura è descritta in dettaglio nel Capitolo 5.
Il progetto "Widi" è stato sviluppato in ambiente Mac OS X, si è scelto quindi di utilizzare l'interfaccia MIDI in uscita da Pure Data fornita dal driver IAC.
Il driver IAC (Inter-application communication) permette di creare connessioni MIDI virtuali tra applicazioni in grado di ricevere messaggi MIDI. Una volta attivato tramite la maschera mostrata in figura 4.6, permette di inviare messaggi MIDI sulla porta in uscita e a loro volta altri software possono ricevere tali messaggi inviati verso il driver IAC. Il driver IAC costituisce in questo caso un'interfaccia per la comunicazione dei messaggi MIDI tra il *server* OSC e altri software verso i quali si vogliono instradare nuovamente i messaggi in arrivo.
La maschera di impostazioni del driver IAC, mostrata in figura 4.6, permette di creare diverse porte in uscita e in ingresso e redistribuirle così su differenti applicativi MIDI.

Figura 4.6: Impostazioni driver IAC

Sebbene il driver IAC sia disponibile solo su piattaforma OS X, esistono altri *tool* per il routing MIDI su altri sistemi operativi come "MidiOX" (Windows), "Patchage" (Linux, multipiattaforma).

Infine grazie agli oggetti *print* nella *patch* di Pure Data, è stato possibile monitorare il flusso di dati nella console principale del software.

4.5.3 RTP / MIDI

La seconda modalità per l'invio dei messaggi MIDI è quella implementata tramite il protocollo RTP/MIDI. Come già descritto in precedenza, questo protocollo fornisce diverse tecniche di recupero di informazioni perse senza *acknowledgement* nella connessione.

Per lo sviluppo di questa modalità è stata utilizzata una libreria per il linguaggio java, citata in [17], documento redatto dal gruppo di lavoro che ha pubblicato il documento ufficiale sulla tecnologia RTP/MIDI presente in [18].

Il nome della libreria è *nmj* ed è al momento di scrittura di questo elaborato l'unica implementazione per linguaggio java disponibile.

L'interfaccia esposta dalla libreria consiste in una classe *view* Android di tipo *dialog* che permette di avviare un protocollo di discovery tramite il quale vengono notificate altre

applicazioni in ascolto per canali RTP MIDI. Una volta scelto il canale RTP è possibile quindi usufruire della libreria per inviare i dati.

In questo caso i messaggi MIDI sono rappresentati da strutture dati byte array.

Come nell'implementazione per il protocollo OSC, anche in questo caso viene creato un byte array per ogni tipologia di messaggio MIDI *channel voice* e inviato tramite un oggetto di tipo *AsyncTask* del framework Android che rende asincrono il processo rispetto al thread della interfaccia grafica. Anche in questo caso ogni byte array viene riutilizzato per ogni invio successivo di messaggio dello stesso tipo.

La porzione di codice mostrata in seguito permette di osservare come la logica di creazione e gestione dei messaggi resti affine a quella implementata per il protocollo OSC.

```
// classe anonima java che implementa l'interfaccia MidiReceiver contenente i metodi per
// la gestione degli eventi MIDI

private final MidiReceiver receiver = new MidiReceiver() {

        // messaggi MIDI istanziati come byte array
        byte[] noteOnByteArray = new byte[]{(byte)0x90, (byte)0x48, (byte)0x7F};
        byte[] noteOffByteArray = new byte[]{(byte)0x90, (byte)0x48, (byte)0x7F};
        byte[] controlChangeByteArray = new byte[]{(byte)0x90, (byte)0x48, (byte)0x7F};
        byte[] programChangeByteArray = new byte[]{(byte)0x90, (byte)0x48};
        byte[] polyAftertouchByteArray = new byte[]{(byte)0x90, (byte)0x48, (byte)0x7F};
        byte[] pitchWheelByteArray = new byte[]{(byte)0x90, (byte)0x48, (byte)0x7F};
        byte[] afterTouchByteArray = new byte[]{(byte)0x90, (byte)0x48};

        //gestione evento MIDI noteOn con metodo sincronizzato per evitare
        //concorrenza nella modifica dei parametri del messaggio
        @Override
        public synchronized void onNoteOn(int channel, int key, final int velocity) {

                // sostituzione dei byte componenti del messaggio
                noteOnByteArray[0]=(byte) (144+channel);
                noteOnByteArray[1]=(byte) key;
                noteOnByteArray[2]=(byte) velocity;
                try {
                        // creazione ed esecuzione in line del wrapper asincrono AsyncTask
                        // il metodo get() invocato sull'esecuzione del wrapper
                        // forza l'attesa dell'esecuzione del codice relativo al
                        // metodo execute, permettendo l'invio seriale dei messaggi

                        new AsynchWrapper().execute(noteOnByteArray).get();
                        // gestione possibili errori dovuti alla non esecuzione del wrapper
                        // asincrono con stampa sulla UI tramite metodo toast(String s)
                } catch (InterruptedException e) {
                        toast("Error while sending message ...");
                e.printStackTrace();
                } catch (ExecutionException e) {
                        toast("Error while sending message ...");
                        e.printStackTrace();
            }

                                        [ . . . ]
```

La ricezione dei messaggi tramite il protocollo RTP/MIDI può essere eseguita su piattaforma OS X e Windows. Mentre per OS X è utilizzabile un componente del sistema operativo, per Windows è necessario installare un software compatibile come quello presentato in [47]. Per piattaforma Linux non esistono, attualmente, software in grado di gestire connessioni RTP/MIDI.

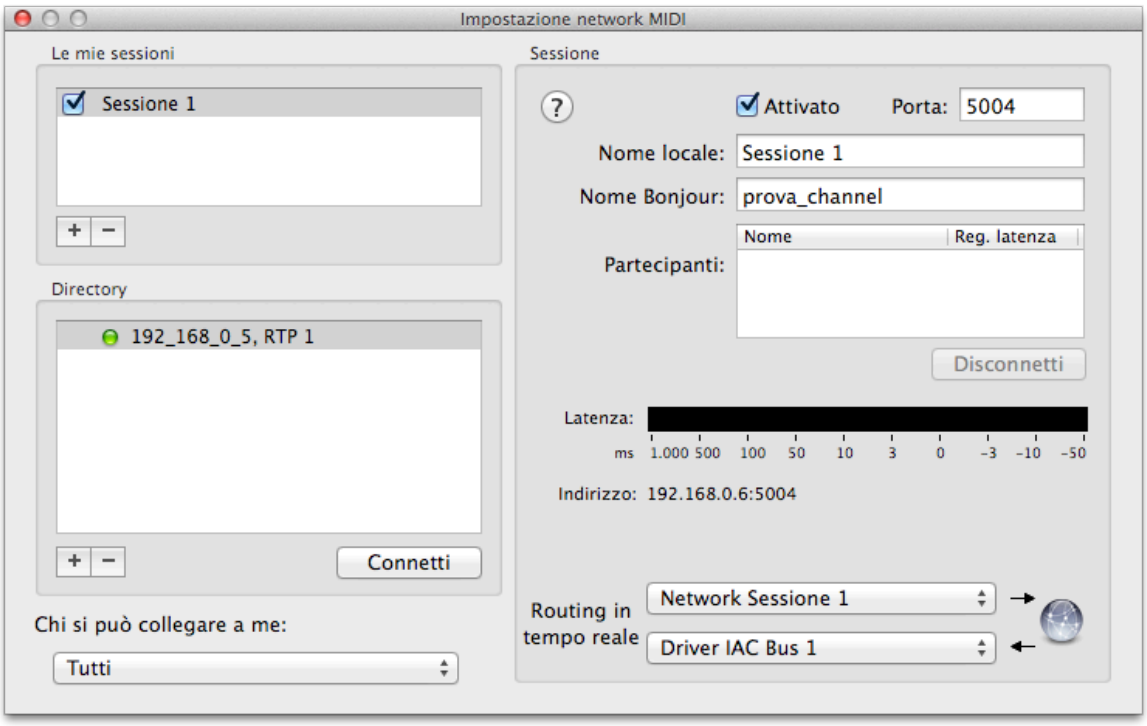


Figura 4.7: Setup network MIDI su piattaforma OS X

In OS X il *tool* preposto per gestire le connessione RTP/MIDI è quello mostrato in figura 4.7; oltre alla gestione di più canali virtuali RTP/MIDI è in grado di interfacciarsi nativamente alle interfacce MIDI del sistema operativo. In OS X il framework che si occupa di questo compito è chiamato *Core MIDI*; offre accesso hardware ad elevate performance ai dispositivi MIDI e una serie di astrazioni software per gli applicativi che necessitano l'utilizzo di hardware MIDI e routing dei messaggi [1].

Un'interfaccia software permette alle applicazioni di dialogare con i layer sottostanti: un server MIDI si occupa di gestire i messaggi e di dialogare a livello kernel con i componenti di I/O, laddove avviene la gestione degli input e degli output hardware (USB, FireWire, Seriale, etc). Come mostrato in figura 4.7 e descritto in [41], la maschera di impostazioni network MIDI di OS X, fornisce anche una stima della latenza dei mesaggi in arrivo,

grazie alla comparsa di linee rosse sulla barra di monitoraggio di colore nero, una per ogni messaggio in arrivo. Ogni barra rossa scompare lentamente lasciando visibilità ai nuovi messaggi in arrivo. Inoltre è possibile ritardare tutti i messaggi in ingresso in modo da regolare la latenza impostando il valore a lato della voce "partecipanti" ed effettuare il routing dei messaggi in arrivo da una sessione verso altre interfacce MIDI, come per esempio verso il driver IAC.

4.6 Design

L'applicazione "Widi", compatibile dalla versione 4.0 del sistema operativo Android, presenta alcuni dei principali *design pattern* proposti dalle specifiche Android. In particolare è fatto largo uso del componente *action bar*, ovvero la barra superiore dell'applicazione dove è presentato il nome, sottoforma di logo, dell'applicazione e alcuni *shortcut* per l'accesso alle funzionalità di settings. Il nome dell'applicazione è stato inserito come *custom view* per l'*action bar*. È riportato il nome dell'applicazione renderizzato a partire da un font di tipo *true type font*. In questo modo su ogni dispositivo, con qualsiasi risoluzione, verrà presentato il logo ad alta risoluzione, senza necessità di inserire una risorsa grafica differente per ogni tipologia di risoluzione. Inoltre è presente un' animazione personalizzata che permette alla prima lettera della stringa "Widi" di ruotare su se stessa in modo da creare il gioco di parole formato da "Midi", ottenuto grazie alla "W" rovesciata, come mostrato in figura 4.9. Oltre alla personalizzazione del componente *action bar* anche i pulsanti presenti nell'applicazione sono stati personalizzati tramite risorse grafiche in formato xml, che specificano diverse caratteristiche per i differenti stati (bottone attivo, disattivo, premuto). In figura 4.8 è presentata la schermata iniziale dell'applicazione dove è possibile osservare la personalizzazione della *action bar* e della grafica dei bottoni. Infine tutte le *view* sono state implementate con il concetto di *fluid layout*: grazie ad alcuni accorgimenti e all'utilizzo di sistemi metrici proporzionali forniti dall'SDK Android, l'applicazione è compatibile con differenti risoluzioni e densità di schermo, sia su device smartphone che su tablet.

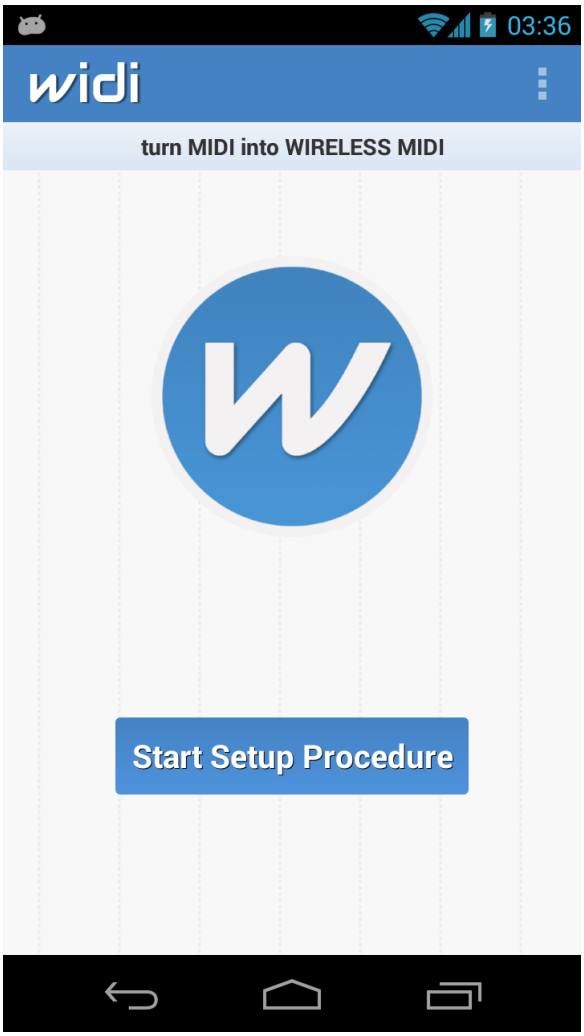


Figura 4.8: Homescreen “Widi” app

Figura 4.9: Logo animato su action bar Android

Capitolo 5

Testing

In questo capitolo è descritto il protocollo di testing utilizzato per la valutazione dell'applicazione proposta.
I test sono stati effettuati durante tutto lo svolgimento dell'implementazione del progetto e sono stati utili alla correzione di bug e al miglioramento generale delle prestazioni del prototipo.
La tipologia principale di test effettuati sull'applicazione "Widi" ha riguardato la misurazione della latenza e delle relative prestazioni riguardanti l'invio dei dati dal client verso l'host ricevente.

5.1 Logging

Sebbene le modalità di misurazione della latenza sono state differenti nel caso di utilizzo del protocollo OSC o RTP/MIDI, in entrambi i casi è stato utilizzato un componente di *logging* asincrono.
La latenza totale nell'esperienza utente, durante l'utilizzo dell'applicazione, è da considerarsi come sommatoria delle latenze di elaborazione del client, sommata alla latenza del mezzo trasmissivo wireless e la latenza di elaborazione dell'host ricevente.
Nel caso dell'utilizzo del protocollo OSC, siccome i dati sono inviati tramite UDP, non è possibile monitorare la rete e i *timestamp* relativi ai pacchetti di *acknowledgement*; per questo motivo sono state misurate le somme parziali tramite le tecniche descritte in seguito.
Utilizzando invece il protocollo RTP/MIDI è stata invece misurata la latenza del mezzo trasmissivo tramite lo strumento presentato nel paragrafo 4.5.3, ovvero la maschera di setup network MIDI fornita da OS X, sommata alla misurazione della latenza del client..
Le latenze del client sono state misurate con il componente di *logging,* il cui compito è quello di scrivere su file il *timestamp* di ricezione degli eventi MIDI attraverso l'interfaccia software di comunicazione con la porta USB, presso la quale il dispositivo MIDI è collegato. Per ogni messaggio inviato, il componente di *logging* scrive anche il *timestamp*

a seguito dell'invio del messaggio, una volta che l'evento MIDI è stato opportunamente impacchettato e spedito sull'interfaccia di rete. Il componente di *logging* lavora in modo asincrono rispetto alle operazioni del client, in modo da falsare i dati di latenza il meno possibile. Nella fase di testing sono stati eliminati altri evnetuali messaggi di log per la console di Android, poiché essi sono elaborati in modo sincrono e potrebbero portare a statistiche di latenza maggiori, data l'importanza nell'applicazione "Widi" del minor tempo possibile nell'esecuzione del codice.

L'applicazione "Widi" è pensata principalmente per inviare i messaggi verso un host di tipo *pc*; utilizzando una macchina OS X per il testing è stato considerato trascurabile il valore di latenza generato dalla gestione presso il framework *Core MIDI* di OS X, sviluppato per lavorare in *real time*.

Il router utilizzato per compiere i test è di tipologia 54mbs, da considerarsi *entry level* rispetto a router con prestazioni più elevate in commercio.

Le statistiche ottenute con i dati registrati dal componente di *logging* sono state elaborate in seguito tramite script in linguaggio di programmazione Python.

5.2 Latenza con protocollo OSC

In questo paragrafo sono mostrati i risultati ottenuti tramite l'utilizzo del protocollo OSC dal client. Gli eventi MIDI vengono convertiti in messaggi OSC e instradati tramite il protocollo UDP sull'interfaccia di rete. La misurazione della latenza del mezzo trasmissivo è stata fatta creando un messaggio di *pingback* che viene rispedito al mittente dal server OSC. Il messaggio è inviato ogni 50 eventi MIDI di *note on* o *note off* ricevuti e presenta un peso in byte paragonabile a quello di un evento MIDI. I *timestamp* di invio e ricezione vengono anche in questo caso scritti su file dal componente di *logging* asincrono. Il tempo necessario per l'invio e la ricezione sul mezzo trasmissivo è stato quindi diviso per 2 in modo da dare una rappresentazione approssimativa della latenza.
In tabella 5.1 sono presenti i dati di latenza per l'elaborazione del client tramite protocollo OSC.

Dispositivo	**Latenza media su 2000 eventi MIDI ricevuti (ms)**	**Percentuale degli invii su 2000 eventi MIDI ricevuti con tempistica inferiore a 5ms**	**Percentuale degli invii su 2000 eventi MIDI ricevuti con tempistica inferiore a 10ms**
Samsung Galaxy Nexus	1,649	98,45	100

Tabella 5.1: latenze invio messaggi client "Widi" con protocollo OSC

Le statistiche relative al tempo necessario per inviare un messaggio OSC dal client all'host ricevente tramite messaggio di *pingback*, sul mezzo trasmissivo wireless, sono mostrate in tabella 5.2

Dispositivo	**Latenza media su 90 messaggi pingback distribuiti sull'invio di 4500 messaggi totali (ms)**	**Percentuale degli invii con tempistica inferiore a 5ms**
Samsung Galaxy Nexus	3,93	75,5

Tabella 5.2: latenze invio messaggi client "Widi" con protocollo OSC

In figura 5.1 è mostrato come la latenza del mezzo trasmissivo con messaggi di *pingback* sia sempre risultata inferiore al valore di 10 ms.

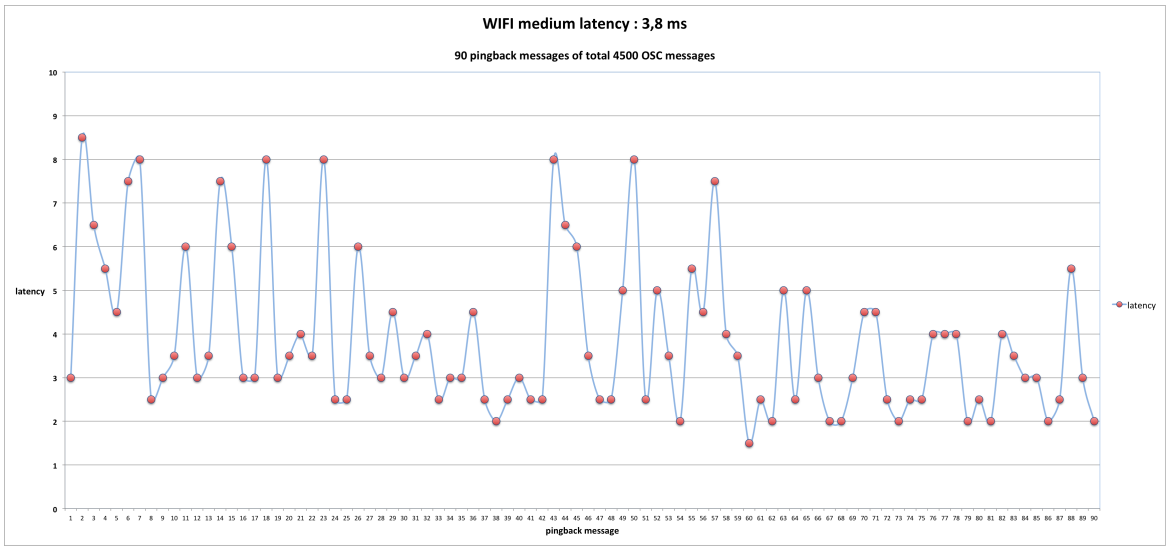


Figura 5.1: Latenze mezzo trasmissivo con messaggi di pingback

5.3 Latenza con protocollo RTP/MIDI

Per la misurazione della latenza del client si è proceduto come descritto per il protocollo OSC. Il componente di *logging* asincrono salva i timestamp di ricezione evento MIDI e quello relativo all'effettivo invio su interfaccia di rete del messaggio che in questo caso è costituito da un byte array.
In tabella 5.3 è mostrata la latenza media di ricezione da interfaccia USB del messaggio MIDI e invio dal client tramite protocollo RTP/MIDI.

Dispositivo	**Latenza media su 2000 eventi MIDI ricevuti (ms)**	**Percentuale degli invii su 2000 eventi MIDI ricevuti con tempistica inferiore a 5ms**	**Percentuale degli invii su 2000 eventi MIDI ricevuti con tempistica inferiore a 10ms**
Samsung Galaxy Nexus	1,6425	97,7	99,85

Tabella 5.3: latenze invio messaggi client "Widi" con protocollo RTP/MIDI

La misurazione della latenza sul mezzo trasmissivo è stata ottenuta, come anticipato, con lo strumento *utility* di OS X per il settaggio delle sessioni MIDI.
Pur essendo una misurazione meno precisa, attraverso la quale non risulta possibile ottenere una lista di dati, in figura 5.2 è mostrata la maschera di monitoraggio della latenza di OS X. Ogni evento ricevuto appare come una barra rossa che lentamente scompare. Una serie di messaggi ripetuti mostra come la latenza d'invio sia sempre inferiore al valore di 10ms con valore minimo di circa 3ms.

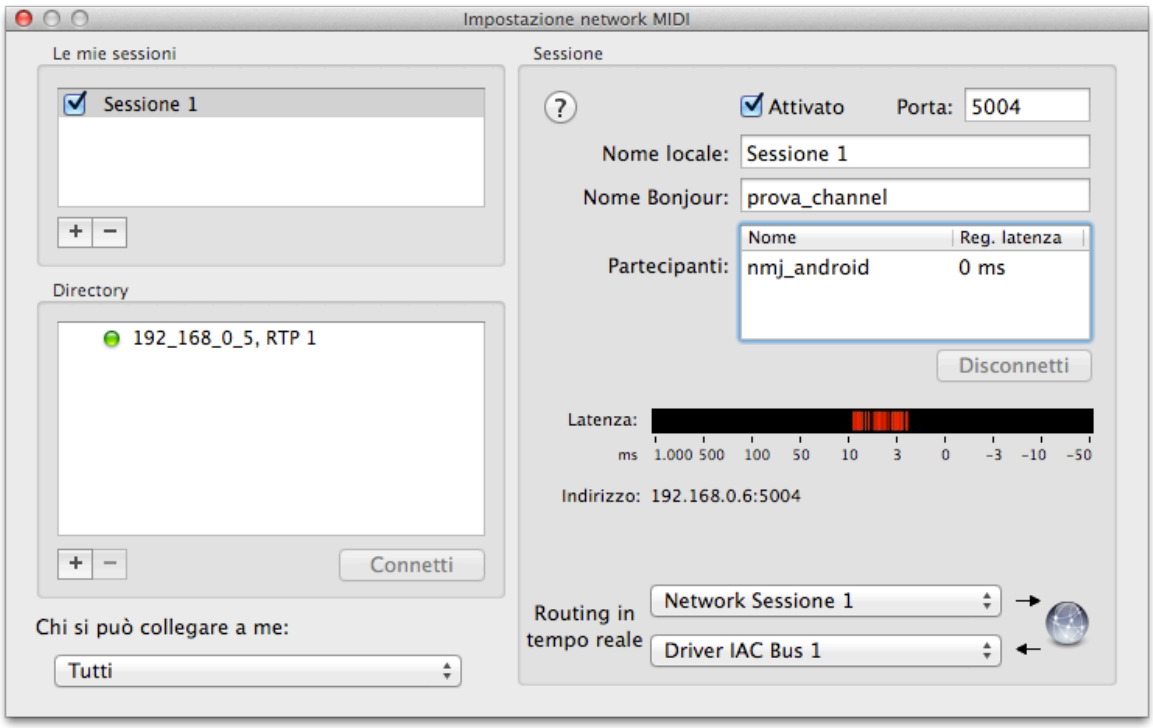


Figura 5.2: Monitoraggio latenze con protocollo RTP/MIDI.

Capitolo 6

Conclusioni e sviluppi futuri

L'analisi svolta per questo lavoro ha permesso di studiare alcune delle tecnologie allo stato dell'arte per quanto riguarda il controllo di strumenti musicali tramite metadati. Sebbene molte nuove soluzioni si siano proposte sul mercato come alternative al protocollo MIDI, ancora oggi non esiste una reale alternativa che sia in grado di soddisfare i requisiti di retrocompatibilità e di invio su rete.

Se da un lato il protocollo MIDI non fornisce metodologie per l'utilizzo sulla rete, utilizzando RTP/MIDI è possibile spedire i dati via wireless con sofisticate tecniche di controllo e recupero di eventuali messaggi persi.

Come evidenziato nel corso di questo lavoro, uno dei protocolli per lo scambio di metadati musicali attraverso la rete che più appare promettente e la cui comunità è molto attiva negli ultimi anni, è il protocollo OSC. Grazie alla implementazione relativamente semplice e alla natura *open source* del progetto, diverse applicazioni per dispositivi mobili hanno iniziato ad utilizzare tale protocollo per lo sviluppo di strumenti di controllo musicale.

Tuttavia OSC rappresenta ancora una soluzione non completamente matura in quanto non propone metodologie per il recupero di informazioni perse, qualora si utilizzi UDP, e la caratteristica di velocità dipenda, di fatto, dal tipo di trasporto utilizzato.

L'utilizzo di OSC su reti locali, come nel caso del *case study* proposto in Capitolo 4, rendono tuttavia tale protocollo sufficientemente sfruttabile senza incorrere in perdite di dati consistenti, anche con l'utilizzo di UDP.

In base alle analisi effettuate, un protocollo come OSC, proponendosi come nuova soluzione per lo scambio di informazioni e metadati musicali per il controllo di performance live, deve necessariamente confrontarsi con il protocollo MIDI per non restare una soluzione di nicchia, il cui utilizzo resti confinato in soluzioni sviluppate *ad-hoc*. Il protocollo HD-MIDI, di proprietà MIDI Manufacturer Association, essendo per sua natura retrocompatibile con il MIDI 1.0, e fornendo specifiche per l'invio tramite rete dei dati, appare come migliore alternativa al MIDI, ma essendo ancora in stato di sviluppo non può essere utilizzato per l'implementazione di applicativi.

L'applicazione "Widi" sviluppata propone un'interfaccia di comunicazione tra il protocollo OSC e il protocollo MIDI: l'utilizzo combinato delle due tecnologie apre le porte a innumerevoli applicativi che comprendano l'utilizzo dei diffusi dispositivi hardware MIDI. Molte applicazioni presenti sul mercato per dispositivi mobili confinano l'utilizzo del protocollo OSC per il controllo di strumenti musicali attraverso interfacce *touch* sulla UI. È evidente la potenzialità di tali interfacce poiché di larga diffusione e accessibili a tutti i possessori di smartphone e tablet; tuttavia l'utilizzo combinato di UI *touch* e periferiche MIDI, potrebbe portare a nuove forme ibride di interazione, in ambito musicale e non. Proprio per questo l'applicazione "Widi" potrebbe essere in futuro estesa per l'utilizzo combinato di periferiche MIDI e controller *touch* direttamente sul dispositivo, utilizzando il protocollo OSC o RTP/MIDI per lo scambio di dati con altri host.

Durante la fase di testing si è voluto analizzare quanto l'applicativo proposto possa effettivamente essere utilizzato in ambito *real-time*, fondamentale per le performance live musicali. Entrambe le soluzioni proposte, tramite protocollo OSC e RTP/MIDI sono risultate particolarmente efficaci nell'elaborazione e invio dei messaggi ricevuti dal dispositivo USB MIDI. Le statistiche rivelano come i due protocolli siano sostanzialmente paragonabili nelle tempistiche di elaborazione del client, con un leggero vantaggio nel numero di invii sotto i 5ms e i 10ms con l'utilizzo di OSC.

Tuttavia è da considerarsi anche il vantaggio a favore del protocollo RTP/MIDI per quanto riguarda le tecniche di recupero dei dati persi, come mostrato nel paragrafo 3.6.2.

L'invio dei dati attraverso il mezzo trasmissivo wireless non garantisce una costante tempistica d'invio e risulta meno efficiente rispetto alla trasmissione via cavo. Tuttavia grazie alle statistiche effettuate emerge come la latenza, tramite l'utilizzo di entrambi i protocolli proposti, si sia sempre rivelata inferiore al valore di 25ms, valore descritto in paragrafo 2.2.1 come massimo accettabile in applicativi interattivi. Nella maggioranza dei casi la latenza risulta anche inferiore dei 10ms descritto come limite per gli applicativi *real-time* da Wright in [45]. È possibile affermare che l'applicazione "Widi" può quindi essere utilizzata in ambiti *prosumer* dove sia necessario trasportare informazioni MIDI senza l'utilizzo di cavi. La natura di applicativo per *smatphone* e *tablet* mobile, rendono la soluzione comparabile con i dispositivi wireless MIDI presenti sul mercato.

Successivi studi potrebbero portare a una ottimizzazione del codice dell'applicazione in modo da ridurre ulteriormente la latenza di elaborazione e invio dei messaggi così come

l'utilizzo di router differenti e più performanti potrebbero portare alla diminuzione della latenza generale generata. Inoltre è possibile studiare metodologie che permettano di utilizzare i *timetag* OSC per ridurre in modo dinamico la latenza percepita adattandone il valore in modo progressivo da parte di un server OSC.

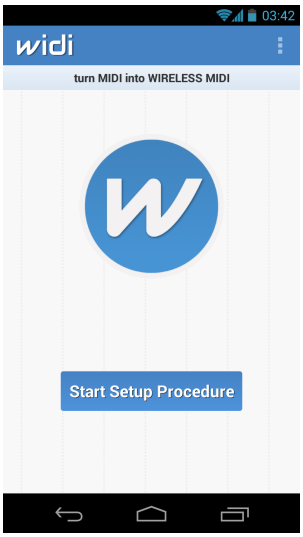
Le tecniche implementate per l'applicazione potrebbero inoltre essere inserite in una libreria in modo da poter essere sfruttata su differenti applicazioni.

Infine, sfruttando le componenti dell'applicazione in modo differente risulta possibile in futuro adattare l'applicativo "Widi" in modo da essere utilizzato come controller per ambiti non musicali come robotica o per casi in cui è necessario il controllo a distanza di software e hardware.

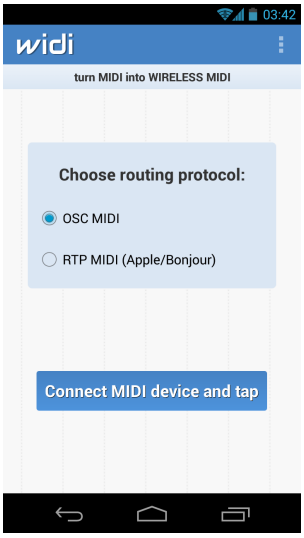
Appendice: Manuale “Widi” app

In questa appendice è presente un breve manuale per l’utilizzo dell’applicazione “Widi”.

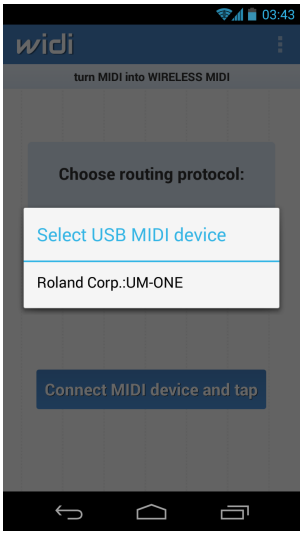
Setup Client



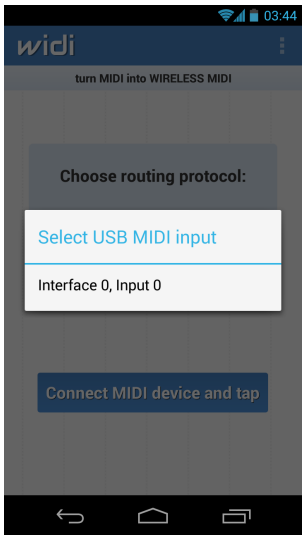
1) Homescreen: premere sul pulsante “Start Setup Procedure”

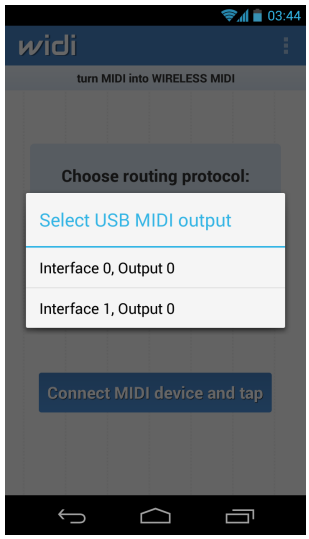


2) Selezionare un protocollo di comunicazione, connettere il dispositivo MIDI USB e premere il pulsante



3) Selezionare il dispositivo collegato tramite USB

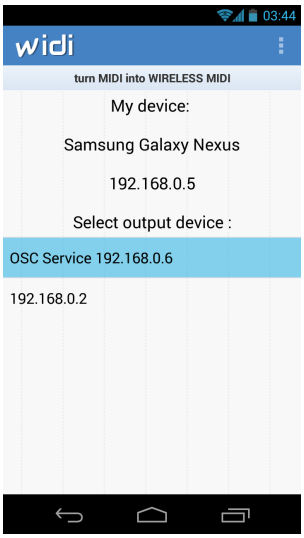




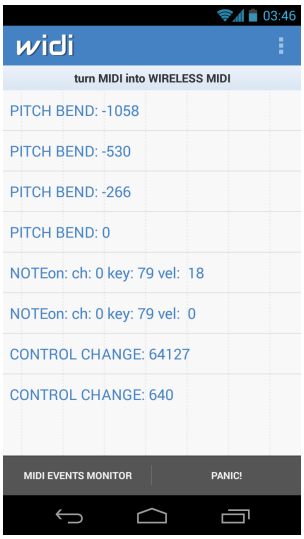
4) In successione: premere sul pulsante OK per dare il permesso all'applicazione di utilizzare il dispositivo MIDI USB, in seguito selezionare le interfacce di input ed output del dispositivo mostrate nei relativi dialog

Settaggio protocollo OSC

Se è stato selezionato il protocollo OSC per il routing, procedere secondo le seguenti istruzioni.



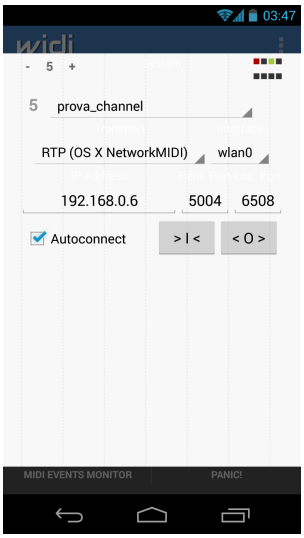
1) Selezionare l'host vero il quale indirizzare i messaggi MIDI tramite protocollo OSC



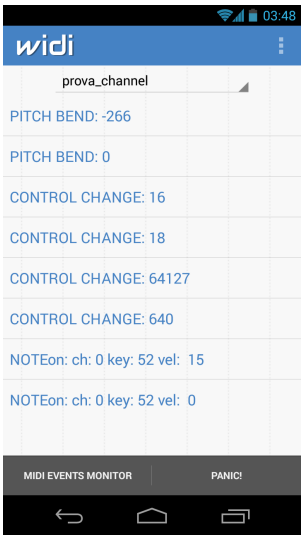
2) L'applicazione è ora in grado di inviare i messaggi tramite OSC; premere "MIDI EVENTS MONITOR" per attivare o disattivare il flusso di messaggi nella lista. Premere "PANIC!" per inviare un messaggio *all sounds off*, qualora una perdita di dati avesse introdotto artefatti nella performance.

Settaggio protocollo RTP/MIDI

Se è stato selezionato il protocollo RTP/MIDI per il routing, procedere secondo le seguenti istruzioni.

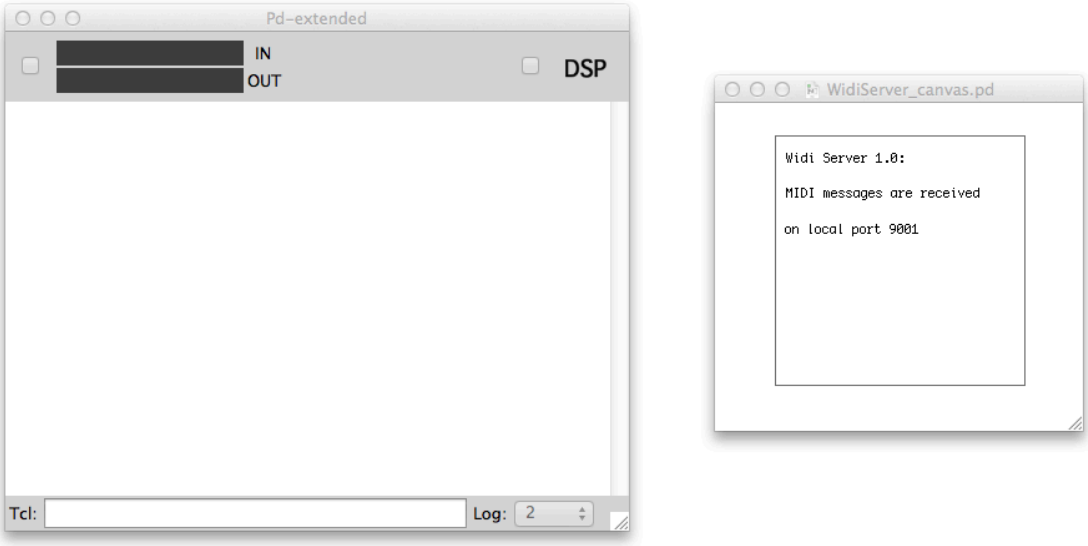


1) Nella maschera di settaggi selezionare nel primo spinner il canale RTP/MIDI desiderato. Premere il tasto “>I<” per attivare la connessione.



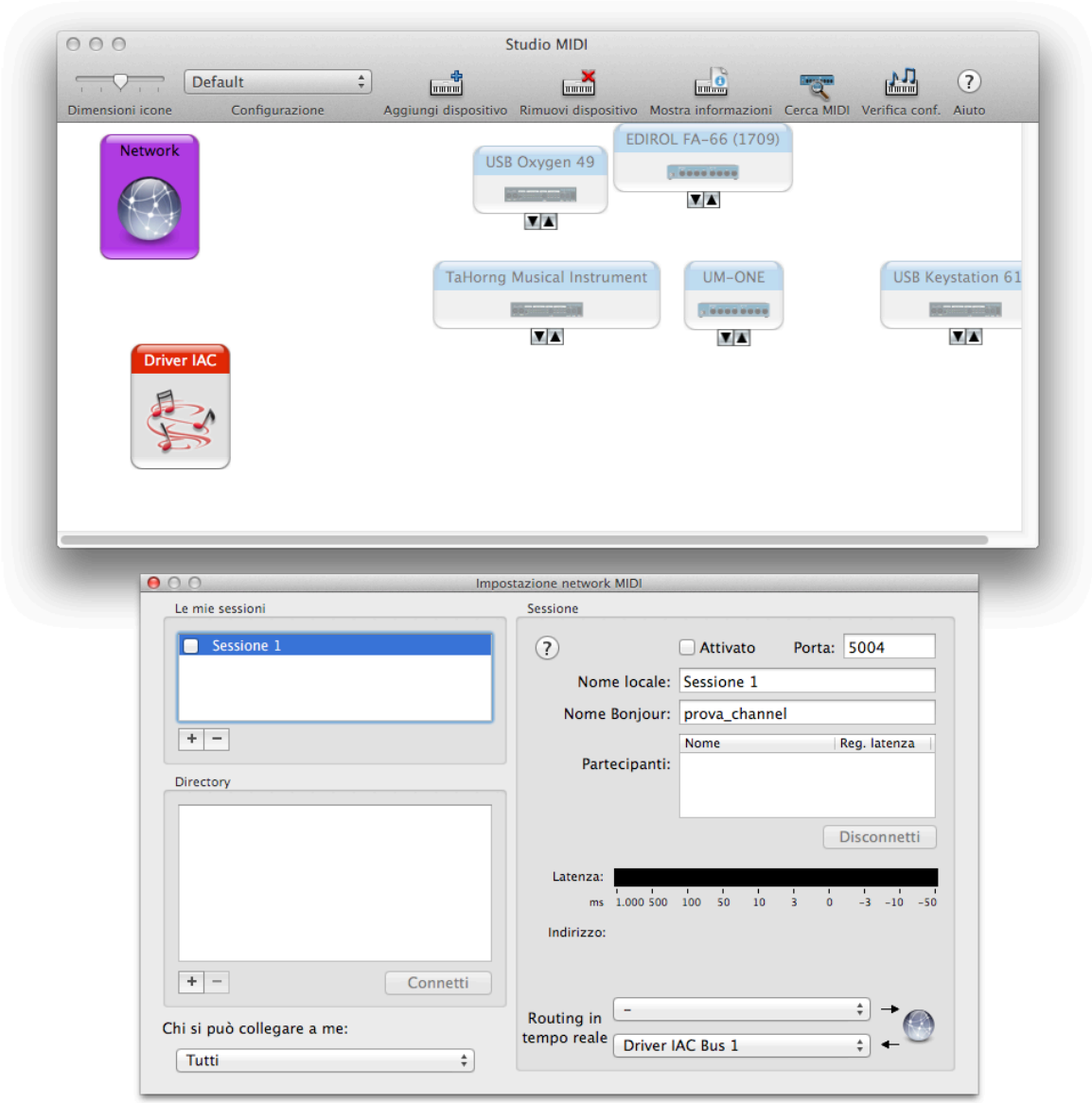
2) L’applicazione è ora in grado di inviare i messaggi tramite RTP/MIDI; premere “MIDI EVENTS MONITOR” per attivare o disattivare il flusso di messaggi nella lista. Premere “PANIC!” per inviare un messaggio *all sounds off*, per evitare eventuali artefatti.

Setup server OSC



Aprire la patch Pure Data. Una volta aperto l'applicativo è pronto per ricevere i messaggi OSC, convertirli in messaggi MIDI e reindirizzarli verso l'uscita MIDI selezionata nelle impostazioni del programma Pure Data.

Setup utility RTP/MIDI



Aprire l'utility "Configurazione Audio MIDI" su OS X, o seguire le impostazioni per il software RTP (Apple/Bonjour) prescelto su piattaforma Windows.

Cliccare su "Network", creare una sessione network con il tasto "+" e attivarla nella parte sinistra della maschera tramite il flag "attivato". Specificare quindi la porta di connessione desiderata e verso quale interfaccia reindirizzare i messaggi MIDI nella parte inferiore a fianco della label "Routing in tempo reale".

Bibliografia

[1] Adamson, Chris and Avila, Kevin, *Learning Core Audio: A Hands-On Guide to Audio Programming for Mac and iOS*, Addison Wesley, 2012, pp 257-261

[2] Brinkman, Peter, *Making Musical Apps*, O'Reilly Media Inc., 2012, pp 59-82

[3] Davinson, Patrick, "OSC", in *Pure Data* Floss Manuals – Free Manuals for Free Software, July 6, 2006, pp. 231-238, available at: http://en.flossmanuals.net/_booki/pure-data/pure-data.pdf, accessed at: January 17, 2014

[4] Farnell, Andy, *Designing Sound*, The MIT Press Cambridge, Massachusetts London, England, 2010 p145-236

[5] Fraietta, Angelo, *Open Sound Control: Constraints and Limitations*, Smart Controller Ptly Ltd, avaible at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.365.3641&rep=rep1&type=pdf, accessed at: February 15, 2014

[6] Freed et al., *Open Sound Control: A Flexible Protocol for sensor networking*, Department of Music, University of California at Berkley, avaible at: http://opensoundcontrol.org/files/OSC-Demo.pdf, accessed at: February 20, 2014

[7] Freed, Adrian and Schmeder, *Andy, Features and Future of Open Sound Control version 1.1 for NIME*, paper presented at the proceedings of the conference NIME 2009, Carnegie Mellon School of Music in Pittsburgh, Pennsylvania, USA

[8] Freeman, Eric, *Head First Design Patterns*, O'Reilly Media, 2004 ch.12 p34

[9] Future Technology Devices International Ltd., *Determining USB Peripheral Device Class*, 2007

[10] Google, *USB Host and Accessory*, Android Developer Documentation, avaible at: http://developer.Android.com/guide/topics/connectivity/usb/index.html, accessed at: October 24, 2013

[11] Gräf, Albert, pd-faust: An integrated environment for running Faust objects in Pd, Dept. of Computer Music, Institute of Musicology Johannes Gutenberg University 55099 Mainz, Germany, paper presented at Linux Audio Conference 2013

[12] Huber, David Miles, *The MIDI Manual: a Practical Guide to MIDI in the Project Studio*, Focal Press, 2007

[13] James, Stuart, *From Autonomous To Performative Contron Of Timbral Spatialization*, paper presented at the proceedings of Australasian Computer Music Conference 2012, July 12-15, 2012, Brisbane

[14] Juszkiewicz ,Henry et al., *Media-Accelerated Global Information Carrier: Engineering Specification Revision 3.0c*, Gibson Guitar Corporation, May 3, 2003

[15] Komatineni, S. and MacLean, D, *Pro Android 4*, Apress Publishing, 2012

[16] Kreidler, Johannes, *Programming Electronic Music in Pd* , <www.kreidler-net.de> 27-01-2009, avaible at: http://www.pd-tutorial.com/english/index.html, accessed at: November 27, 2013

[17] Lazzaro and Wawrzynek, *An RTP Payload Format for MIDI,* paper presented at the conference Audio Engineering Society – 117th Convention, October 28, 2004, avaible at: http://www.cs.berkeley.edu/~lazzaro/rtpmidi/, accessed at: February 20, 2014

[18] Lazzaro et al.,RTP *Payload Format for MIDI*, Internet Engineering Task Force (IETF), avaible at: http://www.rfc-editor.org/rfc/rfc6295.txt, accessed at: February 2, 2014

[19] Malloch, Joseph et al., *A Network-Based Framework for Collaborative Development and Performance of Digital Musical Instruments*, paper presented at 4th International Symposium, CMMR 2007, Copenhagen, Denmark, August 27-31, 2007

[20] McGuire, Sam, *Modern MIDI: Sequencing and programming using traditional and mobile tools*, Focal Press, 2014, chapter 1 and 3

[21] Music Trades Corp, *Celebrating 30 years of MIDI*, , November 2012, avaible at: accessed at: http://digitaleditions.sheridan.com/publication/?i=130912&p=20, January 15, 2014

[22] Oracle, *Class Datagram Packet*, avaible at: http://docs.oracle.com/javase/7/docs/api/java/net/DatagramPacket.html, accessed at: February 16, 2014

[23] Pais, Joao (2009), "Midi", in *Pure Data*, Floss Manuals – Free Manuals for Free Software, July 6, 2006, pp. 239-246, available at: http://en.flossmanuals.net/_booki/pure-data/pure-data.pdf, accessed at: January 17, 2014

[24] Peterson, Larry L, et al, *Computer Networks (4 ed.)*, Morgan Kaufmann, 2007, p.45

[25] Phillips, Dave, *An Introduction to OSC*, The Linux Journal, 2008

[26] Renaud, Alain B. et al, *Networked Music Performance: State Of The Art*, paper presented at the AES 30th International Conference, Saariselkä, Finland, March 13-15, 2007

[27] Schuett, Natan, *, The Effects of Latency on Ensemble Performance*, CCRMA DEPARTMENT OF MUSIC Stanford University, 2002

[28] Schulzrinne, H, et al., *RTP: A Transport Protocol for Real-Time Applications*, The Internet Society, July 2003

[29] The Center For New Music and Audio Technology (CNMAT), UC Berkeley, *Discover of OSC Clients and Servers on a Local Network*, avaible at: http://opensoundcontrol.org/discovery-osc-clients-and-servers-local-network, accessed at: February 23, 2014

[30] The IETF Zeroconf Working Group, *Zero Configuration Networking (Zeroconf)*, avaible at: http://www.zeroconf.org, accessed at: February 23, 2014

[31] The Linux Foundation, *Cross-Referenced Linux and Device Driver Code*, avaible at: http://www.cs.fsu.edu/~baker/devices/lxr/http/source/linux/include/linux/usb/ch9.h, accessed at: February 24, 2014

[32] The MIDI Manufacturer Association*, MIDI 1.0 Detailed Specification*, Document Version 4.2, Revised September 1995

[33] The MIDI Manufacturer Association, *MIDI Manufacturers Investigate HD Protocol*, avaible at: http://www.midi.org/aboutus/news/hd.php accessed at: March 3, 2014

[34] The MIDI Manufacturer Association, *White Paper: Comparison of MIDI and OSC*, 2008, avaible at: http://www.midi.org/aboutmidi/midi-osc.php, accessed at: March 5,2014

[35] USB Implementer Forum*, Introduction to USB On-The-Go*, avaible at: http://www.usb.org/developers/onthego/USB_OTG_Intro.pdf, accessed at: November 6, 2013

[36] USB Implementer Forum, *On-The-Go Supplement to the USB 2.0 Specification Revision 1.0*, December 18, 2001

[37] USB Implementer Forum, *The USB ID Repository*, avaible at: http://usb-ids.gowdy.us/read/UD/, accessed at: February 16, 2014

[38] USB Implementer Forum, *Universal Serial Bus (USB) 2.0 specification*, May 27, 2000

[39] USB Implementer Forum, *USB Class Codes*, avaible at: http://www.usb.org/developers/defined_class, last modified December 7, 2011

[40] USB Implementer Forum, *USB On-The-Go and Embedded Host Supplement to the USB 3.0 specification Revision 1.1*, May 10, 2012

[41] Wherry, Mark, *Mac OS X Tiger: A Musician Guide*, SOS: Sound On Sound Magazine, July 2005, avaible at: http://www.soundonsound.com/sos/jul05/articles/tiger.htm, accessed at: February 20, 2014

[42] Wilkinson, Scott, *Fire in the wire*, Electronic Musician, 08844720, July 1996, Vol. 12

[43] Wright, Matt, *Open Sound Control Specification 1.0,* version 1.0, March 26, 2002, avaible at: http://opensoundcontrol.org/spec-1_0, accessed at: November 20, 2013

[44] Wright, Mattew, *A comparison between of MIDI and ZIPI*, Center for New Music Audio Technology, Department of Music, University of California, Berkley, 1994

[45] Wright, Mattew, *Open Sound Control: an enabling technology for musical networking*, Center for New Music and Audio Technology (CNMAT), University of California at Berkeley, 2005

Sitografia

[46] Ehrentraud, F., *SynOSCopy*, 2007, avaible at: https://github.com/fabb/SynOSCopy/wiki, accessed on February 2, 2014

[47] Erichsen, Thobias, *rtpMIDI Tutorial*, avaible at: http://www.tobias-erichsen.de/software/rtpmidi/rtpmidi-tutorial.html , accessed at: February 20, 2014

[48] *iRig KEYS from IK Multimedia Becomes the First Lightning-compatible Music Keyboard, Sunrise*, Florida (PRWEB), 2003, accessed at: March 5, 2014 http://www.prweb.com/releases/2013/6/prweb10828595.htm

[49] Kasten and Levien, *High Performance Audio*, Google I/O session, May 15-17, 2013, Moscone Center, San Francisco, avaible at: https://developers.google.com/events/io/sessions/325993827, accessed at: March 9, 2014

[50] Ramakrishnan, Chandrasekhar, *Illposed Software - Java OSC*, avaible at: http://www.illposed.com/software/javaosc.html, accessed at: February 16, 2014

Ringraziamenti

Desidero ringraziare il Prof. Ludovico e il Dott. Baldan per la disponibilità e i preziosi consigli che mi hanno dato durante le fasi di sviluppo e di stesura di questa tesi.
Ringrazio i miei genitori, Valeria e i miei amici per il costante e fondamentale supporto durante questo percorso.