MIDI

Una guida al protocollo, alle estensioni e alla programmazione

Luca A. Ludovico

Milano University Press

Luca A. Ludovico

MIDI
Una guida al protocollo, alle estensioni e alla programmazione

Milano University Press

MIDI. Una guida al protocollo, alle estensioni e alla programmazione / di Luca Andrea Ludovico. Milano: Milano University Press, 2021.

ISBN 979-12-80325-11-2 (print)

ISBN 979-12-80325-28-0 (PDF)

ISBN 979-12-80325-32-7 (EPUB)

DOI 10.13130/milanoup.27

Pubblicato da:

Milano University Press
Via Festa del Perdono 7 – 20122 Milano
Sito web: https://milanoup.unimi.it
e-mail: redazione.milanoup@unimi.it

L'edizione cartacea del volume può essere ordinata in tutte le librerie fisiche e online ed è distribuita da Ledizioni (www.ledizioni.it)

Si ringrazia il Prof. Federico Avanzini che, nel ruolo di Volume Editor, ha fornito prezioso supporto scientifico e operativo alla pubblicazione.

A Ivan G. Zoppini,
la persona più generosa
che abbia mai conosciuto.

Indice

Prefazione

Dagli anni ’80 a oggi lo standard MIDI ha rivestito un ruolo particolarmente importante nel panorama dell’informatica musicale e della musica elettronica. Esso consiste in uno standard industriale, protocollo di comunicazione e interfaccia hardware tra strumenti musicali elettronici, prevalentemente digitali, cui tutto il mondo delle aziende – prima hardware e poi anche software – ha aderito.

Nato ufficialmente nel 1983 sotto l’egida della *MIDI Association*, dopo qualche anno di confronto e accordi tra buona parte delle industrie di strumenti musicali elettronici – specialmente statunitensi e giapponesi – lo standard MIDI ha mantenuto sempre viva la sua rilevanza tra critiche, proposte alternative, estensioni dello standard e integrazioni tecnologiche con l’audio digitale, l’editoria elettronica, internet, dispositivi mobili e altri ambiti musicali ulteriori; ha comunque sempre avuto ragione dei vari tentativi di sostituirlo con standard più avanzati e performanti.

Le pubblicazioni divulgative, tecniche e applicative che trattano lo standard MIDI sono centinaia, in tutte le lingue del mondo. Soprattutto sono moltissime le introduzioni tutoriali per musicisti.

Questo libro offre una nuova angolatura prospettica del tema e risponde ad un’esigenza che chiedeva di essere soddisfatta, combinando gli elementi fondamentali sullo standard con le sue estensioni più rilevanti e recenti. L’intento dichiarato è quello di fornire un quadro di riferimento e gli elementi tutoriali per chi è interessato allo sviluppo di applicazioni informatiche che utilizzino dati MIDI, nei vari ambiti applicativi di riferimento, sia per la formazione universitaria che per le attività professionali della musica.

Il contesto in cui nasce questo libro è l’esperienza ormai ventennale sulla formazione alla progettazione e sviluppo di applicazioni ICT in ambito musicale del corso di laurea triennale in Informatica Musicale e del relativo percorso del corso di laurea magistrale in Informatica dell’Università degli Studi di Milano, di cui l’autore è sempre stato protagonista e docente di vari insegnamenti specifici.

Goffredo Haus*

* Goffredo Haus ha fondato nel 1985 il Laboratorio di Informatica Musicale dell’Università degli Studi di Milano. È attualmente chair dell’IEEE Standards Association Working Group for XML Musical Application e Prorettore all’Innovazione Digitale dell’Università degli Studi di Milano.

Capitolo 0

Introduzione

MIDI è l'acronimo di *Musical Instrument Digital Interface*, ossia interfaccia digitale per gli strumenti musicali.

Il protocollo MIDI è stato entusiasticamente accolto e ampiamente utilizzato da musicisti e compositori fin dalla sua nascita, avvenuta tra il 1982 e il 1983 [6]. I dati MIDI sono un metodo molto efficiente per rappresentare le informazioni sulle performance musicali, e questo ha reso il MIDI un protocollo di interesse, non solo per compositori o esecutori, ma anche per applicazioni informatiche accompagnate da sonificazione e musica, quali le presentazioni multimediali o i giochi per computer. In un'epoca in cui le tecnologie di rete offrivano una banda limitata all'utente, il MIDI ha rappresentato la soluzione per condividere informazione musicale nella forma di *song* MIDI, e – nonostante la notevole evoluzione tecnologica – tuttora esistono ampie collezioni di file MIDI usati come basi musicali o per l'interscambio di informazione di partitura.

Grazie alla riduzione dei costi e all'ampliamento dell'offerta hardware, alla diffusione di interfacce PC/MIDI, al supporto da parte di sistemi operativi e linguaggi di programmazione e alla disponibilità di software compatibile, il protocollo MIDI si è affermato e ha conosciuto un successo duraturo.

MIDI è innanzitutto una specifica per il collegamento e il controllo di strumenti musicali elettronici. L'uso principale della tecnologia MIDI consiste nell'esecuzione e nella composizione musicale, sebbene essa trovi applicazione anche in molte aree correlate quali il missaggio, l'editing e la produzione audio, il campo videoludico, la robotica, il controllo delle luci di scena, le suonerie per telefoni cellulari, e via dicendo [24].

Al giorno d'oggi il MIDI è uno standard industriale per il collegamento di dispositivi eterogenei, tra cui strumenti musicali digitali, computer, tablet e smartphone. Il MIDI viene utilizzato da musicisti, DJ, produttori, educatori, artisti e amatori per creare, eseguire, apprendere e condividere musica e lavori multimediali.

La comunità di utenti e di sviluppatori è quanto mai viva, come testimoniato dall'affermarsi di recenti migliorie rispetto allo standard MIDI 1.0 (ad esempio, *MIDI Polyphonic Expression*) e dal rilascio a inizio 2020 delle specifiche MIDI 2.0.

Partendo da queste premesse, il presente testo punta a offrire una ricognizione completa dell'universo MIDI, dalle origini del protocollo alle sue più recenti evoluzioni.

0.1 A chi è destinato questo libro

Questo libro è pensato per tutti coloro che abbiano interesse verso il mondo MIDI e le sue molteplici declinazioni: dai principianti che necessitano di una guida graduale e scientificamente solida nell'esplorazione del protocollo, ai professionisti che desiderano conoscerne le più recenti evoluzioni o sviluppare applicazioni software per il MIDI.

Il testo raccoglie in maniera organica il materiale didattico accumulato dall'autore in 10 anni di docenza di Programmazione MIDI, insegnamento erogato dal Dipartimento di Informatica nell'ambito del percorso musicale della laurea magistrale in Informatica. L'esigenza di un

testo articolato e scientificamente accurato nasce da una ricerca condotta nella letteratura di riferimento, carente in particolare su due aspetti: le recenti novità in ambito MIDI, tra cui la versione 2.0 del protocollo, e il supporto offerto dai linguaggi di programmazione allo sviluppo di applicazioni MIDI. A questo riguardo vale la pena ricordare l'operato del gruppo di lavoro W3C sulla standardizzazione della *Web MIDI API*.

Il volume si rivolge pertanto a semplici appassionati, studenti e professionisti nel vasto ambito noto come *sound and music computing*, che spazia dalla produzione artistica attraverso l'uso o con il supporto di strumenti tecnologici alla progettazione e implementazione di tali strumenti.

0.2 Finalità e utilità del libro

Il testo ambisce ad approfondire l'universo MIDI da vari punti di vista, dalla storia del protocollo alla descrizione puntuale dei suoi messaggi, dal formato Standard MIDI Files alle integrazioni via via apportate dalla MIDI Association.

Il MIDI è un ecosistema complesso e in continua evoluzione, che abbraccia più versioni di un protocollo di comunicazione numerica tra dispositivi, altri protocolli specializzati in ambiti specifici, diverse categorie di strumenti hardware e software e formati di file per descrivere performance musicali o altri aspetti del protocollo. Ognuno di questi argomenti viene trattato nel dettaglio, fornendo, ove necessario, degli esempi chiarificatori.

La prima parte del testo, dedicata al protocollo originario e alle sue successive evoluzioni, non ha particolari prerequisiti. Il lettore con competenze elementari in ambito musicale e informatico sarà agevolato nel comprendere alcuni passaggi nel quadro di una trattazione dei vari argomenti che mira a essere esaustiva. Verranno in aiuto al lettore eventuali collegamenti ad altre parti del volume e, in caso di approfondimenti troppo dettagliati per gli scopi del libro, i riferimenti alla documentazione ufficiale riportati all'interno del testo.

Grande spazio viene inoltre lasciato alla programmazione per il MIDI. Particolarmente originale è la parte dedicata alla *Web MIDI API*, un'iniziativa di standardizzazione in ambito web condotta dal *W3C Web Audio Working Group*. Il volume introduce inoltre la libreria Java e il toolbox MATLAB dedicati a MIDI. Per motivi di brevità non sarà possibile accompagnare il lettore nell'apprendimento dei linguaggi e dei formalismi menzionati, ma la trattazione è corredata da un cospicuo numero di esempi appositamente commentati.

0.3 Struttura del libro e argomenti trattati

Il testo si articola in due parti, intitolate, rispettivamente, "Protocolli, estensioni e formati di file" e "Programmazione MIDI".

La prima parte si concentra sul protocollo MIDI e le sue estensioni, sui formati di file e sulla versione 2.0 dello standard. In particolare:

- il Capitolo 1 si concentra sulle basi del formato MIDI con riferimento alle specifiche 1.0. Oltre alla storia del protocollo, si affrontano le modalità di interscambio dei dati, le principali tipologie di dispositivi MIDI, i fondamentali concetti di ricevitore e trasmettitore e di porta e connettore MIDI, i più comuni schemi di collegamento tra dispositivi e i principali aspetti elettrici e circuitali relativi all'hardware MIDI;
- il Capitolo 2 muove dalla definizione di canale MIDI, di byte di stato e di byte di dati, per fornire poi una panoramica esaustiva sulle famiglie di messaggi MIDI;
- il Capitolo 3 descrive alcuni protocolli collaterali rispetto alle specifiche MIDI 1.0, in particolare MIDI Machine Control e MIDI Show Control;
- il Capitolo 4 si focalizza sull'estensione General MIDI e sulle ulteriori evoluzioni, incluse le recenti specifiche MIDI Polyphonic Expression;

- il Capitolo 5 descrive ed esemplifica adeguatamente il formato Standard MIDI Files, pensato per salvare e caricare dati relativi alle performance MIDI;
- il Capitolo 6 affronta le specifiche della versione 2.0 del protocollo, di recente rilascio; pur introducendo notevoli novità, esse rappresentano un'estensione di MIDI 1.0 e conservano la compatibilità con esso.

La seconda parte del libro affronta la realizzazione di applicativi software in ambito MIDI utilizzando diversi linguaggi e formalismi. Nello specifico:

- il Capitolo 7 introduce l'argomento della programmazione MIDI;
- il Capitolo 8 si occupa della *Web MIDI API*, un'iniziativa in fase di standardizzazione in ambito W3C volta a integrare il MIDI nella programmazione web;
- il Capitolo 9 descrive ed esemplifica il package javax.sound.midi, che fornisce interfacce e classi per l'I/O, il *sequencing* e la sintesi di dati MIDI in Java;
- il Capitolo 10 affronta l'*Audio Toolbox* di MATLAB, con particolare riferimento al supporto offerto a MIDI;
- il Capitolo 11 tratta il MIDI in tempo reale in Csound, noto linguaggio di programmazione audio e timbrica;
- il Capitolo 12 presenta il supporto a MIDI offerto da Pure Data nella versione "vanilla".

0.4 Convenzioni

Il presente testo adotta alcune convenzioni tipografiche per rendere più agevole la lettura e il riconoscimento di determinati elementi.

In generale il **grassetto** denota concetti di particolare importanza e l'*italico* parole in lingua inglese di uso non comune in Italiano.

I valori numerici che non riportano indicazioni a pedice sono da intendersi espressi in base 10. Nel testo si è scelto di impiegare la virgola come separatore decimale e il punto, ove richiesto, come separatore delle migliaia. Negli esempi di codice ci si rifà necessariamente a quanto previsto dai diversi linguaggi, quindi il punto assume il significato di separatore decimale e non esiste un carattere di separazione delle migliaia.

Nel testo si fa ampio uso di valori in base 2 e in base 16. In questi casi la base viene esplicitata a pedice (ad esempio 10100100_2 e $A4_{16}$). I numeri e/o le stringhe binarie si presentano solitamente come raggruppamenti da 8 cifre e, in caso di più ottetti, questi vengono separati da uno spazio per migliorarne la leggibilità. Le cifre esadecimali sono invece raggruppate a coppie, e più coppie vengono eventualmente distanziate tramite uno spazio.

Gli intervalli numerici sono rappresentati tra parentesi quadrate, con gli estremi separati da una virgola. Ad esempio, la notazione [0, 255] indica l'intervallo da 0 a 255 (estremi inclusi), e le sue rappresentazioni in base 2 e in base 16 risultano, rispettivamente, [00000000_2, 11111111_2] e [0_{16}, FF_{16}].

Per i messaggi MIDI viene adottata la convenzione tipografica del maiuscoletto (ad esempio PROGRAM CHANGE). Le stringhe binarie o esadecimali che costituiscono messaggi MIDI, contenuti di Standard MIDI Files, ecc. vengono stampate con carattere monospaziato, e – ove opportuno – separate dal resto del testo, come alla riga seguente:

```
10000000 01001100 01000000
```

Riguardo alla denominazione dei connettori nelle porte MIDI e al nome dei messaggi, il presente testo si rifà al documento di specifica originario. Ad esempio, le diciture MIDI IN, OUT e THRU vengono adottate in luogo di altre forme comunemente in uso quali MIDI In, Out e Thru.

Analogamente, NOTE-ON viene privilegiato ad altre varianti comunemente in uso quali NOTE-ON, NOTE ON, NOTEON, NOTE ON [1].

Le figure che danno una rappresentazione grafica dei messaggi MIDI sono strutturate ponendo il mittente sul lato destro e il destinatario sul lato sinistro. Per quanto possa apparire controintuitiva, questa disposizione consente di riportare in modo naturale gli ottetti binari dei messaggi scrivendo la cifra più significativa a sinistra.

Gli *snippet* di codice presenti nella seconda parte del testo sono riportati in carattere monospaziato e su sfondo grigio, come mostrato nell'esempio sotto riportato.

```
1    <!-- snippet di codice -->
2    <html>
3    ...
4    </html>
```

0.5 Supporto e risorse utili

La principale risorsa di riferimento per la documentazione ufficiale è il sito della MIDI Association [2], la cui missione è costituire e raccordare una comunità globale inclusiva di persone che fanno musica con il MIDI. Il sito rappresenta la raccolta ufficiale di informazioni su tutto ciò che riguarda la tecnologia MIDI, dai classici dispositivi *legacy* [3] ai protocolli di nuova generazione. Effettuando la registrazione gratuita alla MIDI Association, è possibile accedere ai materiali online e scaricare i documenti di specifica.

L'autore si rende disponibile a rispondere alle domande di chiarimento e approfondimento inviate all'indirizzo e-mail luca.ludovico@unimi.it, e sarà lieto di ricevere pareri, dare seguito a suggerimenti migliorativi e raccogliere segnalazioni riguardo a refusi ed errori.

Alcune risorse software di particolare interesse per la programmazione MIDI, per la creazione di catene virtuali e per il monitoraggio dello scambio di messaggi su personal computer vengono menzionate nel Capitolo 7.

0.6 Principali abbreviazioni

Nel testo vengono utilizzate le seguenti abbreviazioni:

- AMEI, Association of Music Electronics Industry
- API, Application Programming Interface
- bpm, beats per minute (pulsazioni al minuto)
- DAC, Digital to Analogue Converter (convertitore digitale/analogico)
- DAW, Digital Audio Workstation(s)
- DCB, Digital Communication Bus
- fps, frames per second (fotogrammi al secondo)
- GM, General MIDI
- GS, (Roland) General Standard o General Sound
- IANA, Internet Assigned Numbers Authority
- IMA, International MIDI Association
- IMUG, International MIDI User Group

[1] Per verificare l'incoerenza terminologica introdotta da differenti autori, basti pensare che in [28] viene regolarmente utilizzata la forma "Note-On" nella sezione *MIDI 1.0 Detailed Specification*, mentre nell'appendice *Additional Explanations and Application Notes* si parla di "Note On".

[2] https://www.midi.org/

[3] In campo informatico, si definisce *legacy* un sistema, un'applicazione o un componente obsoleti, che però continuano a essere usati poiché non si intende rimpiazzarli o non è possibile farlo.

- JMSC, Japan MIDI Standards Committee
- LSB, Least Significant Byte
- MIDI, Musical Instrument Digital Interface
- MIME, Multipurpose Internet Mail Extensions
- MMA, MIDI Manufacturers Association
- MMC, MIDI Machine Control
- MPE, MIDI Polyphonic Expression
- MSB, Most Significant Byte
- MSC, MIDI Show Control
- MTC, MIDI Time Code
- MUG, MIDI Users Group
- pc, personal computer
- PPQN, Pulse per Quarter Note
- SMF, Standard MIDI File(s)
- SMPTE, Society of Motion Picture and Television Engineers
- URL, Uniform Resource Locator
- USI, Universal Synthesizer Interface
- W3C, World Wide Web Consortium
- XG, (Yamaha) eXtended General MIDI

0.7 Errata corrige

Il presente libro è disponibile in versione a stampa *on demand* e in versione digitale *open access* sul sito libri.unimi.it [4]. Quest'ultima verrà periodicamente aggiornata al fine di correggere errori e imprecisioni.

L'errata corrige è disponibile all'URL https://ludovico.lim.di.unimi.it/download/libri/midi/errata_corrige.pdf.

0.8 Ringraziamenti

L'Autore desidera ringraziare particolarmente alcune figure che hanno contribuito all'opera a vario titolo.

Fondamentale è stato l'apporto di Goffredo Haus, professore del Dipartimento di Informatica "Giovanni Degli Antoni" dell'Università degli Studi di Milano, che per primo ha trasmesso all'Autore – ai tempi ancora studente di Ingegneria informatica – la passione verso le tematiche del *sound and music computing*. Il suo insegnamento di *Informatica applicata alla musica* ha rappresentato la prima occasione di confronto con il protocollo MIDI.

Un sentito ringraziamento va ai colleghi del Laboratorio di Informatica Musicale dell'Università degli Studi di Milano, in particolare ad Adriano Baratè per i preziosi consigli riguardo al Capitolo 9 e a Giorgio Presti per l'attenta rilettura del Capitolo 10.

[4] https://libri.unimi.it/

Parte I

Protocolli, estensioni e formati di file

Capitolo 1

Il formato MIDI

Questo capitolo si concentra sulle basi del formato MIDI con riferimento alle specifiche 1.0. Innanzitutto vengono richiamati i principi ispiratori alla base del protocollo (Paragrafo 1.1). Viene poi affrontata la storia del MIDI, fondamentale per comprenderne alcuni aspetti peculiari (Paragrafo 1.2). Una volta introdotte le modalità di interscambio dei dati (Paragrafo 1.3) e le principali tipologie di dispositivi MIDI (Paragrafo 1.4), vengono presentate alcune definizioni fondamentali, quali i concetti di ricevitore e trasmettitore (Paragrafo 1.5) e di porta e connettore MIDI (Paragrafo 1.6). Nel Paragrafo 1.7 si affrontano i più comuni schemi di collegamento tra dispositivi, in particolare il cosiddetto *daisy chaining*. Infine, nel Paragrafo 1.8 vengono descritti i principali aspetti elettrici e circuitali relativi all'hardware MIDI.

1.1 Principi basilari

Per **protocollo MIDI** si intende la struttura che definisce la semantica dei messaggi di controllo MIDI. Il protocollo MIDI si basa su due principi ispiratori: universalità ed espandibilità.

Ogni produttore di dispositivi MIDI, quando ne stabilisce la compatibilità, certifica che l'apparecchiatura riceva e analizzi i dati generati da qualsiasi altro dispositivo MIDI, interpretandoli o ignorandoli a seconda del comportamento atteso; analogamente qualsiasi dato MIDI generato e trasmesso deve poter essere analizzato (e quindi gestito o ignorato) da qualsiasi altro dispositivo MIDI, senza causare errori. Ogni messaggio MIDI ha lo stesso significato tanto per il mittente quanto per il destinatario.

Riguardo all'espandibilità, le specifiche MIDI costituiscono un insieme piuttosto vasto di comandi, ed è molto raro incontrare dispositivi in grado di rispondere a ogni messaggio previsto. In molti casi, non avrebbe nemmeno senso per una certa categoria di dispositivi rispondere a un determinato messaggio: è il caso dei *controller* e dei sintetizzatori digitali nei confronti di TUNE REQUEST, messaggio pensato per ricalibrare gli oscillatori interni. Quindi le specifiche MIDI contengono messaggi che codificano funzioni o comandi cui un dispositivo compatibile può rispondere o meno. Tale comportamento è perfettamente accettabile, a condizione che il dispositivo sia progettato per ignorare i comandi che non comprende e non cerchi di interpretarli, o – peggio ancora – entri in una condizione di errore. Questa caratteristica è un aspetto chiave che ha permesso di estendere le specifiche nel corso degli anni mantenendo la retrocompatibilità: i messaggi via via aggiunti alle specifiche, non comprensibili ai dispositivi meno recenti, vengono semplicemente ignorati.

1.1.1 Flessibilità

Le specifiche MIDI sono stato deliberatamente progettate per lasciare dei "buchi" (combinazioni di bit indefinite, fili nei connettori non associati a segnali, ecc.) in vista di sviluppi futuri. Ad esempio, il messaggio MIDI TIME CODE QUARTER FRAME non si trovava nelle specifiche originali, e solo a partire dal 1987 ha sostituito una posizione precedentemente marchiata come "undefined".

L'eventuale ricezione di MIDI TIME CODE QUARTER FRAME non porta i dispositivi costruiti prima di tale data in una condizione di errore, perché il messaggio viene semplicemente ignorato. Altri comandi MIDI sono stati ridefiniti, o il loro significato chiarito, attraverso integrazioni e revisioni periodiche delle specifiche, ma sono rimasti sintatticamente invariati e la loro compatibilità con i dispositivi MIDI esistenti non è mai stata compromessa.

Il protocollo MIDI contiene anche degli stratagemmi pensati fin dall'origine per renderlo intrinsecamente flessibile ed estensibile. Uno di questi meccanismi è l'utilizzo dei messaggi di sistema esclusivi, che consentono l'invio di sequenze più o meno lunghe di byte il cui significato è specifico per il destinatario (Paragrafo 2.10.1) o è stato concordato in un secondo momento tra produttori (Paragrafo 2.10.2). Un altro meccanismo di estensione è quello dei numeri di parametro registrati e non registrati (Paragrafo 2.5.4).

La specifica MIDI 1.0 ha costituito la versione più recente fino a inizio 2020, anche se il documento originario si è ampliato notevolmente nel corso degli anni. Il set di comandi originale è ancora supportato, nonostante sia nata una nuova versione del protocollo. Nell'ottica della compatibilità, ogni dispositivo MIDI deve essere in grado di comunicare con ogni altro dispositivo MIDI, anche se alcuni strumenti presenteranno caratteristiche assai più evolute e potenzialità estese rispetto ad altri. Questo è uno degli aspetti più interessanti delle specifiche MIDI: un dispositivo compatibile creato oggi è ancora in grado di comunicare con un prodotto di oltre 30 anni fa, il che ha garantito e continuerà a garantire caratteristiche di longevità ben superiori rispetto a molti altri standard informatici.

1.1.2 Semplicità

La specifica MIDI è relativamente semplice se confrontata con altri standard per la comunicazione o per la rappresentazione di informazione audio e musicale. Questo perché ci si avvale di un principio chiamato "intelligenza distribuita": ogni periferica MIDI, a seconda della propria funzione, presenta una logica di controllo atta a generare, analizzare, interpretare, manipolare e inviare messaggi.

Il protocollo di comunicazione riguarda quindi comandi per istruire le apparecchiature nel produrre suono (o altri effetti). I comandi vengono inviati sotto forma di messaggi per lo più brevi, normalmente costituiti da un numero di byte variabile tra 1 e 3. La struttura del linguaggio è semplice, e la varietà di messaggi necessari a codificare una performance e a controllare i dispositivi MIDI è relativamente piccola.

1.1.3 Aspetti economici

Un altro obiettivo dei primi progettisti MIDI era contenere i costi. Ciò è stato fatto nell'interesse generale, in modo che il maggior numero possibile di produttori (e di utenti finali) fosse incentivato ad adottare il protocollo. Quando questo è stato inizialmente introdotto, si stimava che l'aggiunta della tecnologia MIDI a un sintetizzatore digitale sarebbe costata tra 5 e 10 $ a livello di processi industriali.

Questa scelta spinse ad adottare alcuni compromessi tecnologici, finalizzati a contenere i costi connessi in particolare all'hardware. Ad esempio, l'utilizzo di connettori DIN a 5 pin e la limitata velocità di trasmissione dei dati (si veda il Paragrafo 1.3) sono il risultato di queste decisioni iniziali, che hanno poi influenzato l'intera evoluzione del protocollo.

1.2 Cenni storici

La storia del protocollo MIDI affonda le proprie radici in tecnologie e approcci molto lontani nel tempo, la cui trattazione dettagliata esulerebbe dagli scopi di questo libro. Si rimanda il lettore interessato alla collezione di articoli sulla storia e l'evoluzione del MIDI raccolti nel sito midi.org [37].

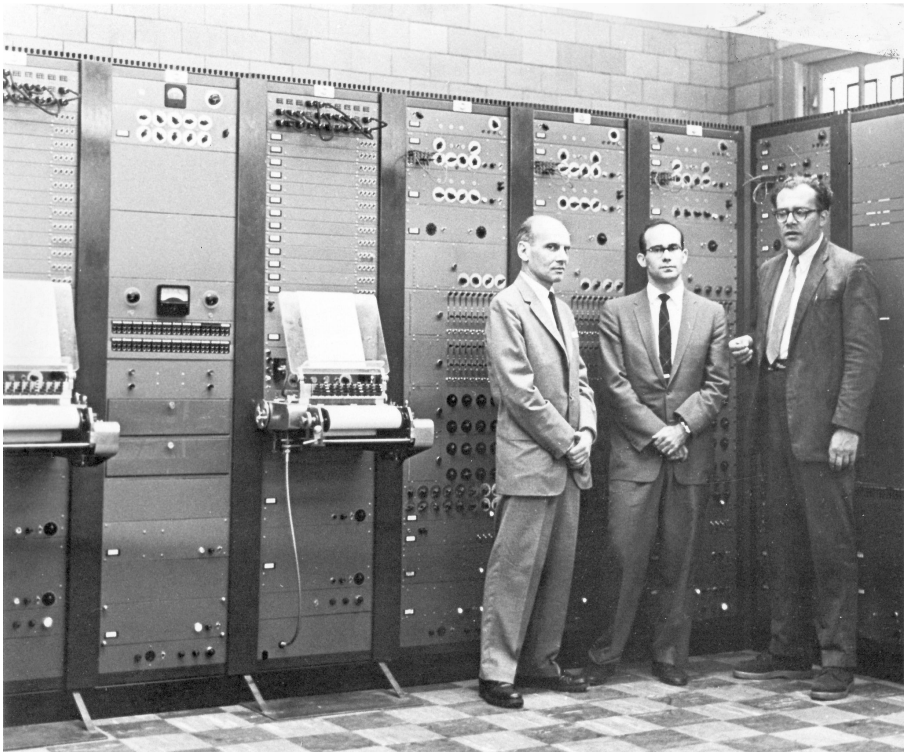


Figura 1.1. Milton Babbitt, Peter Mauzey e Vladimir Ussachevsky a fianco dell'*RCA Mark II Synthesizer* presso il Columbia University Computer Music Center (1958).

La parola *sintetizzatore* assume rilievo negli anni '50 con i dispositivi *RCA Mark I* e *Mark II Sound Synthesizer*, che hanno dimensioni paragonabili a quelle di una stanza (Figura 1.1). In particolare, il *Mark II*, progettato da Herbert Belar e Harry Olson in RCA e installato alla Columbia University nel 1957, rappresenta il dispositivo di punta del Columbia-Princeton Electronic Music Center. A oggi, esso viene considerato il primo sintetizzatore elettronico programmabile.

L'opera di pionieri quali Robert Moog, Don Buchla, Harold Bode, Pete Zinovieff e Dave Cockerell permette ben presto di ridurre gli ingombri e ottenere dispositivi più appetibili per il mercato. A Moog, in particolare, si riconosce il merito di aver svincolato il sintetizzatore dalla ricerca di laboratorio e averlo messo nelle mani dei musicisti. Un prodotto particolarmente interessante da questo punto di vista è il *Minimoog*, un sintetizzatore analogico monofonico messo in commercio nel 1970 da Moog Music e nuovamente commercializzato, con alcune piccole modifiche, nel 2016. Tra i suoi punti di forza si annovera il fatto di costituire uno dei primi sintetizzatori dal prezzo accessibile, poco ingombrante, leggero e relativamente semplice da programmare (Figura 1.2).

I compositori e i musicisti possono ora cominciare a esplorare le potenzialità timbriche dei sintetizzatori. Una pietra miliare, in tal senso, è l'album "Switched–On Bach", realizzato in studio della compositrice statunitense Wendy Carlos e pubblicato nel 1968. Esso contiene alcuni pezzi tra i più noti di Johann Sebastian Bach, tra cui l'intero Concerto brandeburghese n. 3, eseguiti con un sistema modulare Moog. Il successo di pubblico è notevole, tanto che l'album diventa uno dei primi dischi di musica classica a vendere 500.000 copie.

La popolarità dei sintetizzatori conosce un notevole impulso nel 1978, quando iniziano ad apparire dispositivi basati su microprocessori. In questo ambito si distingue una giovane società californiana, Sequential Circuits, grazie a un suo prodotto particolarmente fortunato: la tastiera con funzioni di sintesi *Prophet-5*. Sebbene assai limitato rispetto agli standard odierni, per i suoi tempi il *Prophet-5* offre buoni livelli di suonabilità, stabilità e polifonia. Il suo prezzo di circa 4.000$ risulta elevato per l'epoca, ma ben presto la concorrenza di sistemi simili rilasciati da Korg, Roland e Yamaha porta a un dimezzamento dei costi per l'utente finale.

Figura 1.2. Il *Minimoog Model D* del 1970 nella riedizione del 2016.

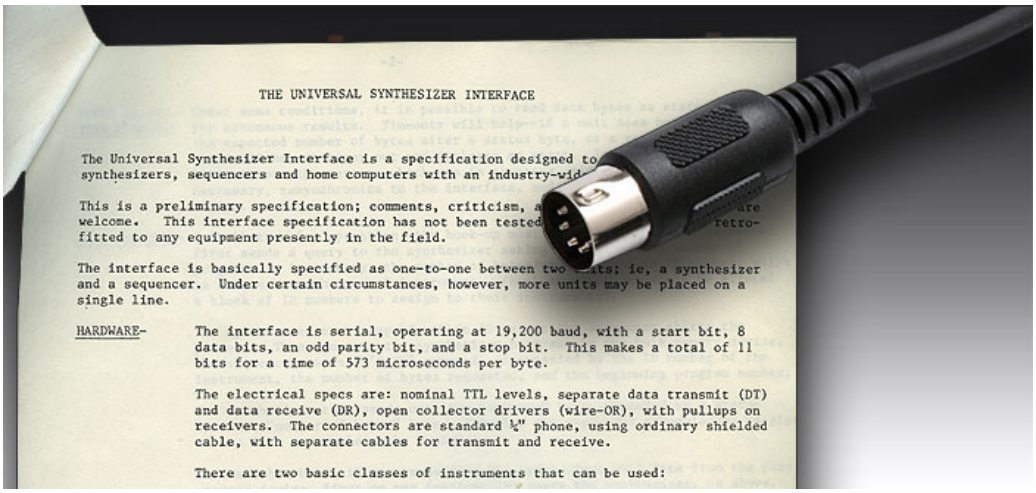
Le caratteristiche tecniche dei sintetizzatori continuano a evolvere, ma nei primi anni '80 la compatibilità e l'interoperabilità rappresentano ancora un grosso problema. L'eterogeneità nella progettazione dei sintetizzatori porta ogni produttore a definire, ad esempio, i dati di altezza e durata delle note secondo una propria logica. A livello di singola azienda viene condotto uno sforzo per progettare interfacce digitali il cui fine è connettere dispositivi diversi di un dato produttore, senza ancora supportare il dialogo tra dispositivi di produttori differenti. Ad esempio, Roland sviluppa il *Digital Communication Bus* (DCB), Yamaha la *Key Code Interface*, e via dicendo.

Alcuni esperti del settore, tra cui Dave Smith di Sequential Circuits e Ikutaru Kakehashi di Roland, cominciano a temere che la mancanza di compatibilità tra produttori diversi possa limitare la diffusione dei sintetizzatori, con l'inevitabile conseguenza di penalizzare la crescita delle vendite. A questo punto, grazie alla loro lungimiranza, l'idea di un sistema di comunicazione digitale universale inizia a circolare e a diffondersi: si tratta dell'alba del protocollo MIDI.

Le specifiche di tale interfaccia vengono presentate in forma embrionale nel corso dell'*AES Convention* tenutasi nel 1981 [1]. In questa occasione, il già citato Dave Smith e Chet Wood discutono pubblicamente un lavoro di ricerca incentrato sulla cosiddetta *Universal Synthesizer Interface* (USI) [34]. Nell'articolo, essa viene definita come una specifica progettata per fornire a sintetizzatori, *sequencer* e personal computer interconnessi un'interfaccia standard di livello industriale. Il protocollo di comunicazione prevede l'utilizzo di normali jack $1/4''$ e una velocità di trasmissione pari a 19,2 kBd [2]. L'obiettivo originario è consentire la connessione di un sintetizzatore a un *sequencer*, ma si prospetta già la possibilità di porre più unità in serie. Intenzione dichiarata degli autori è quella di definire specifiche preliminari e non testate sul campo, invitando la comunità scientifica e industriale a esplicitare commenti, critiche e proposte alternative. La Figura 1.3 mostra l'incipit del documento originario.

[1] 70th AES Convention, October 30-November 2, 1981. New York, NY, USA.

[2] Il *baud* (simbolo Bd) è un'unità di misura che prende il nome da Émile Baudot, inventore del codice Baudot utilizzato per le telescriventi. Nel campo delle telecomunicazioni e dell'informatica, il baud rappresenta il numero di simboli (dati) che viene trasmesso nell'unità di tempo di un secondo. Il *baud rate* viene spesso erroneamente confuso con il *bit rate*, ossia il numero di bit al secondo. Va sottolineato che i due concetti sono equivalenti solo se il simbolo considerato è la cifra binaria, mentre, in generale, la codifica di un simbolo può richiedere un numero di bit differente a seconda delle tecniche di modulazione usate. Ad esempio, nei modem che usano la modulazione numerica di ampiezza in quadratura, il *bit rate* è pari a 6 volte il *baud rate*.


THE UNIVERSAL SYNTHESIZER INTERFACE

The Universal Synthesizer Interface is a specification designed to
synthesizers, sequencers and home computers with an industry-wid

This is a preliminary specification; comments, criticism, a
welcome. This interface specification has not been tested
fitted to any equipment presently in the field.

The interface is basically specified as one-to-one between two
and a sequencer. Under certain circumstances, however, more units may be placed on a
single line.

HARDWARE- The interface is serial, operating at 19,200 baud, with a start bit, 8 data bits, an odd parity bit, and a stop bit. This makes a total of 11 bits for a time of 573 microseconds per byte.

The electrical specs are: nominal TTL levels, separate data transmit (DT) and data receive (DR), open collector drivers (wire-OR), with pullups on receivers. The connectors are standard ¼" phone, using ordinary shielded cable, with separate cables for transmit and receive.

There are two basic classes of instruments that can be used:

Figura 1.3. Incipit del lavoro scientifico che descrive la *Universal Synthesizer Interface*.

Figura 1.4. Il retro del sintetizzatore analogico *Sequential Circuits Prophet-600*.

Il progetto, ulteriormente sviluppato in cooperazione tra diverse aziende, fa il suo debutto alla fiera NAMM di Los Angeles del 1983, dove un sintetizzatore analogico *Prophet-600* di Sequential Circuits si dimostra in grado di comunicare con un tastiera *Jupiter 6* di Roland attraverso un cavo a 5 pin. La Figura 1.4 mostra il retro del sintetizzatore analogico *Prophet-600* del 1982, ove è possibile individuare, a destra del nome del modello, le connessioni MIDI IN e MIDI OUT.

Le prime specifiche del protocollo vengono pubblicate sotto il titolo “MIDI 1.0 Specification” [18] nell’agosto 1983 da parte dell’*International MIDI Association* (IMA), nota anche come *MIDI Users Group* (MUG) o *International MIDI User Group* (IMUG). Una copia del prezioso documento viene ritrovata solo nel novembre 2018 nell’archivio di Yamaha Corporation of America a Buena Park, in California. Sebbene composto da sole 14 pagine, il documento fornisce un’interessante panoramica sugli aspetti originari del protocollo, facendone emergere i punti salienti, le ambiguità che poi avrebbero portato al rilascio di specifiche più dettagliate, e anche qualche curiosità. Ad esempio, alcuni messaggi il cui significato sarebbe stato standardizzato non risultavano ancora definiti, come nel caso dei *Continuous Controllers 64–95*, genericamente definiti *Switches (On/Off)*. Una tabella del documento contiene l’elenco dei primi 14 produttori aderenti all’iniziativa; tra essi, oltre ai già citati Sequential Circuits e Roland, figurano Bon Tempi (*sic!*), Kawai, Korg, Yamaha e Moog Music. Il logo sul frontespizio differisce significativamente da quello oggi utilizzato, come mostrato nella Figura 1.5.

L’“IMA News Bulletin” pubblicato nel giugno del 1985, oltre ad annunciare un ventaglio di

Figura 1.5. Il logo MIDI che appare nel documento di specifica del 1983 (a sinistra) e quello ufficialmente adottato in seguito (a destra).

nuovi dispositivi MIDI (tra cui la batteria elettronica *Simmons SDS-9* compatibile con il protocollo), cita una nuova specifica MIDI 1.0, creata dal *Japan MIDI Standards Committee* (JMSC) e tradotta dal giapponese da parte della *MIDI Manufacturers Association* (MMA). Sono trascorsi ben due anni dalla prima versione delle specifiche MIDI distribuita dall'IMA. Il nuovo documento, in cui le specifiche vengono definite "dettagliate", serve a risolvere i problemi di comprensione e interpretazione da parte di molte aziende. Si tratta dell'ultimo documento approntato in lingua non inglese in ambito MIDI; da quel momento, i vari attori stabiliscono di aderire a una lingua unica per facilitare l'interscambio di informazioni. La necessità è dettata dal numero di attori (associazioni, aziende, team di ricerca) che hanno agito finora in grande autonomia. Sebbene JMSC, MMA e IMA si siano tutte costituite per favorire lo sviluppo e la gestione del MIDI, serviranno alcuni anni perché queste finalmente cooperino.

Vista l'abbondanza di organizzazioni che riuniscono gli attori interessati al MIDI a vario titolo, è bene fare chiarezza. L'IMA è la prima associazione a formarsi in ordine di tempo: essa si auto-proclama come l'organizzazione finalizzata allo sviluppo e alla promozione del MIDI. Si ricorda che l'IMA è l'associazione che registra il documento del 1983 presso il *Copyright Office* statunitense, una volta ottenute le specifiche iniziali da Sequential Circuits. Le aziende giapponesi interessate al MIDI, nell'implementare le specifiche MIDI 1.0, si rendono conto che sono necessari ulteriori dettagli e assumono il compito di produrre un documento aggiuntivo. A tale scopo, esse si consorziano all'interno del JMSC, lanciato ufficialmente nel novembre 1983 e originariamente aperto anche agli utenti finali. Nel giugno del 1984 il JSMC informa i colleghi dell'IMA riguardo alla preparazione di una spiegazione dettagliata delle specifiche, che sarebbe stata condivisa una volta completata. Il JSMC suggerisce la costituzione di un'analoga associazione dei produttori statunitensi ed europei. La decisione di formare la MMA viene presa durante il *NAMM show* del giugno 1984 a Chicago, e coinvolge in prima istanza Roland US, Yamaha US, Sequential Circuits e Oberheim. Si caldeggia la partecipazione e l'adesione all'MMA di tutte le aziende interessate. Anche se i documenti legali (in particolare, lo statuto) non vengono completati e depositati fino al 1987, l'associazione comincia a usare il nome MMA a partire da giugno 1984, come si evince dall'"IMA Bulletin" che fissa a giugno 1985 il primo incontro dell'MMA aperto ai futuri membri. Si riporta nella lingua originale uno degli obiettivi dichiarati per l'incontro:

> «to prepare and release an official and complete documentation to the MIDI 1.0 Specification, including the specification, detailed explanations, and a guide to creating implementation charts.»

Negli anni successivi, l'MMA apporta numerose migliorie alla tecnologia MIDI, aggiornando le specifiche per riflettere tali cambiamenti. Nel tempo, vengono pubblicate, nella forma di documenti indipendenti, ulteriori specifiche, tra cui si menziona il formato Standard MIDI Files (SMF) e il General MIDI, argomenti che verranno trattati negli appositi capitoli del presente testo.

Nel 1996 l'MMA decide di assemblare i vari documenti all'interno di una singola pubblicazione, intitolata "Complete MIDI 1.0 Detailed Specification" [28]: si tratta di una fotografia dello stato dell'arte riguardo alla tecnologia MIDI. Il documento è liberamente scaricabile dal sito della MIDI Association [3]. Nel dettaglio, i suoi contenuti riguardano:

- MIDI 1.0 Detailed Specification (specifiche elettriche e protocollo di comunicazione);
- General MIDI 1 (incluse le linee guida per gli sviluppatori General MIDI);
- il formato per gli Standard MIDI Files;
- le specifiche del protocollo MIDI Show Control;
- le specifiche del protocollo MIDI Machine Control;
- le specifiche del MIDI Time Code.

[3] https://www.midi.org/specifications-old/item/the-midi-1-0-specification

Dalle 14 pagine delle prime specifiche si passa a 58, incluse 7 pagine di appendici e 13 di tabelle. Modifiche e integrazioni successive al 1996 vengono pubblicate come documenti separati, anch'essi attualmente scaricabili dal sito della MIDI Association.

La più recente tappa della pluridecennale storia del MIDI ha luogo il 19 gennaio 2020, durante il meeting annuale dell'MMA. In tale occasione vengono approvate all'unanimità le "MIDI 2.0 specifications" [30], che segnano una pietra miliare nel percorso di sviluppo dello standard, tanto da comportare una progressione nella numerazione della versione principale. MIDI 2.0 introduce novità di grande rilievo, tra le quali la possibilità di comunicazione bidirezionale tra dispositivi, mantenendo comunque la compatibilità con MIDI 1.0.

1.3 Flusso di dati e temporizzazione

Il flusso di dati MIDI avviene tra un **mittente** (*sender*), ossia un dispositivo in grado di creare messaggi MIDI e inviarli ad altri dispositivi, e un **destinatario** (*receiver*), ossia un dispositivo in grado di ricevere messaggi MIDI e analizzarli attraverso l'operazione detta *parsing*. Un mittente può trasmettere messaggi a più destinatari, un destinatario può ricevere messaggi da più mittenti, e uno stesso dispositivo all'interno del *setup* MIDI può fungere tanto da mittente quanto da destinatario.

Secondo le specifiche originarie, lo scambio di dati MIDI avviene tramite un **flusso di bit asincrono unidirezionale a 31,25 Kbit/s**. Si può facilmente evincere come la velocità di trasmissione, ragguardevole secondo i parametri dei primi anni '80, appaia al giorno d'oggi molto limitata; si pensi al *bit rate* di alcune connessioni comuni, quali Bluetooth 4.0 + LE (circa 1 Mbit/s) o USB 3.x (da 4,8 a 20 Gbit/s).

Si osservi, inoltre, che il flusso è unidirezionale, quindi due dispositivi MIDI tra loro connessi possono comunicare attraverso l'invio di messaggi che viaggiano dal mittente al destinatario, e non viceversa. Una comunicazione bidirezionale è possibile solo aggiungendo un secondo canale trasmissivo, in cui i ruoli dei dispositivi si invertono.

Nel Paragrafo 2.2 si vedrà come i messaggi MIDI si articolino su uno o più byte. Sul canale trasmissivo, in realtà, per ciascun byte vengono trasmessi 10 bit: un bit di avvio (*start*), gli 8 bit di dati MIDI effettivi, e, infine, un bit di arresto (*stop*). Nei prossimi capitoli verranno ignorati i bit aggiuntivi legati alla trasmissione elettrica del segnale, e ci si concentrerà sui soli bit relativi al contenuto MIDI.

Il tipo di connessione è seriale, quindi non è fisicamente possibile inviare o ricevere più messaggi MIDI simultaneamente. Ad esempio, i messaggi generati dalla pressione simultanea di più tasti che formano un accordo, anche nell'ipotesi – poco plausibile – di perfetta simultaneità del gesto, verrebbero inviati sul canale trasmissivo uno dopo l'altro. A un diverso livello di astrazione, questo si verifica anche sui bit che costituiscono un singolo messaggio, trasmessi sul canale fisico secondo una logica seriale.

Poiché il MIDI è stato progettato per trasmettere dati sull'esecuzione musicale, esso deve fornire una temporizzazione sufficientemente accurata per preservarne l'integrità ritmica. Un orecchio allenato è sensibile a variazioni anche estremamente piccole, dell'ordine dei millisecondi. Si consideri, ad esempio, il tempo metronomico di 120 *beats per minute* (bpm), comunque inferiore ai circa 200 bpm cui corrisponde convenzionalmente l'indicazione *Prestissimo*. Se la pulsazione è il quarto, a 120 bpm tale figura ritmica dura 500 ms, quindi un trentaduesimo dura 62,5 ms e un sessantaquattresimo 31,3 ms: l'ordine è delle decine di millisecondi.

Si definisce *jitter* la variazione del tempo relativo tra due o più eventi. Si tratta di un fenomeno che ha rilevante impatto sulla percezione dell'integrità ritmica. Come riportato in [28], vari studi hanno dimostrato che un *jitter* dell'ordine dei millisecondi può essere udibile e avere un impatto sulla qualità percepita [4, 33, 35], e alcune ricerche hanno dimostrato che i musicisti sono in grado di controllare intervalli di tempo con una precisione di circa 1,5 ms [14].

Nell'ambito di una temporizzazione imprecisa gioca un ruolo fondamentale anche la *latenza* (*delay*), ossia il ritardo tra l'attivazione di un evento e il verificarsi del suo effetto. Nel caso della

performance musicale, un esempio di latenza riguarda il tempo che intercorre tra la pressione di un tasto e l'emissione del suono corrispondente. Per quantificare l'effetto in una performance musicale, si consideri la distanza tra i membri di un quartetto d'archi e si rammenti che il suono viaggia nell'aria a circa 340 m/s; nel caso peggiore, ossia tra gli strumentisti che si trovano alle estremità della formazione, la latenza dovuta alla propagazione del suono nell'aria è di circa 7 ms. Una latenza di 10 ms viene generalmente considerata impercettibile, purché il *jitter* sia modesto.

Con una velocità di trasmissione dati di 31,25 Kbit/s e 10 bit per byte di dati, l'invio dei più comuni messaggi MIDI (quali NOTE-ON e NOTE-OFF, che richiedono 3 byte ciascuno) impiega circa 1 ms. In pratica, per eventi perfettamente simultanei, il protocollo MIDI può fornire *jitter* inferiore a 1 ms e latenza di 3 ms o meno, il che è pienamente accettabile per la performance di un solista. La velocità di trasmissione si rivela meno adeguata quando i dati vengono immessi sul canale da più attori, causando dunque un maggiore traffico, mentre il protocollo di comunicazione può rivelarsi poco efficace per i dati MIDI generati da un sistema di elaborazione, estremamente preciso nel determinare la temporizzazione e potenzialmente in grado di gestire un gran numero di eventi musicali simultanei. In quest'ultimo scenario molti eventi dovranno "attendere il proprio turno" per essere trasmessi via MIDI; inoltre, eventi simultanei saranno ritardati di quantità di tempo differenti, il che si traduce in un degrado progressivo nella percezione del ritmo. Per minimizzare l'impatto di tale effetto, MIDI riduce il numero di byte trasmessi facendo affidamento sulla tecnica del *running status*, che sarà descritta nel Paragrafo 2.11.

1.4 Principali categorie di dispositivi MIDI

Il flusso di dati MIDI è normalmente originato da alcune specifiche tipologie di dispositivi: i *controller* e i *sequencer*.

Un ***controller* MIDI** è un dispositivo che si interfaccia con uno strumentista e traduce in tempo reale la performance di quest'ultimo in un flusso di dati MIDI. Trascurando le latenze introdotte dalle componenti meccaniche ed elettroniche, il gesto dello strumentista (ad esempio, la pressione di un tasto) viene immediatamente convertito nel corrispettivo messaggio MIDI, che viene immesso sul canale trasmissivo non appena il traffico dati lo consente.

Un esempio di *controller* è rappresentato dalle tastiere, storicamente la prima categoria di dispositivi di input presa in considerazione. Il protocollo MIDI e la sua terminologia sono stati influenzati profondamente da questa matrice storica. Basti pensare al termine *key* per indicare l'altezza della nota, o a *velocity* per indicare la sua intensità, facendo implicito riferimento alla rapidità del gesto di pressione del tasto da parte dello strumentista.

Esistono però numerose altre tipologie di *controller*la cui interfaccia può richiamare quella degli strumenti musicali tradizionali o meno. Nella prima categoria si possono annoverare, a titolo di esempio, chitarre, batterie, flauti, sassofoni e via dicendo (Figura 1.6a). Risulta, poi, molto ampia e variegata la casistica di *controller* che rientrano nella seconda categoria, offrendo al musicista un interfacce di controllo originali (Figura 1.6b).

Un ***sequencer* MIDI** è un dispositivo hardware o software che consente di acquisire, archiviare, modificare, combinare e riprodurre sequenze di dati MIDI. L'acquisizione, nel caso di un *sequencer*, non deve essere confusa con la registrazione del suono in formato digitale. Il *sequencer* infatti acquisisce parametri delle note musicali quali la durata di ciascuna nota, la sua altezza, l'intensità con cui è stata prodotta, e via dicendo. Se è pur vero che i *sequencer* software siano spesso in grado di operare anche sull'audio digitale, questi aspetti esulano dal dominio del MIDI e avvicinano piuttosto tali prodotti al mondo delle *digital audio workstation* (DAW).

Il destinatario del flusso di dati MIDI è generalmente un **sintetizzatore**, dispositivo volto a interpretare i messaggi MIDI relativi alla performance e tradurli in eventi sonori. Noto anche come modulo audio o modulo sonoro, in lingua inglese il sintetizzatore viene detto *synthesizer*, spesso abbreviato in *synth*, termine, questo, ampiamente in uso anche in lingua italiana. Le tipologie di suoni prodotti sono dette, nel gergo MIDI, *patch*, programmi (*programs*), algoritmi (*algorithms*) o timbri (*timbres*). Si tratta di sinonimi che, nel testo, potranno essere usati in modo intercambiabile.

(a) *Roland AE05 Aerophone Go.* (b) *AKAI LPD8 Pad Controller.*

Figura 1.6. Esempi di *controller* MIDI.

A ciascun timbro, i sintetizzatori normalmente assegnano un valore numerico detto numero di programma (*program number*) o numero di *patch* (*patch number*).

Un altro termine utilizzato talvolta per designare i sintetizzatori è **expander**, vocabolo che indica un modulo di espansione timbrico finalizzato alla generazione di suono. La configurazione al cui interno si collca un *expander* tipicamente include altri componenti, in quanto esso normalmente non presenta funzioni di *controller*.

Una categoria particolare di dispositivi per la produzione del suono è quella delle **drum machine**, strumenti hardware o software che hanno l'obiettivo di eseguire sequenze ritmiche riproducendo il timbro di strumenti a percussione. Sono dette più propriamente *drum synth* quelle dotate di un sintetizzatore analogico o digitale. Le *drum machine*, al pari di *sequencer* e sintetizzatori, sono state inventate assai prima del protocollo MIDI.

Va sottolineato che talvolta la distinzione tra *controller* e *synth* è logica anziché fisica. Ad esempio, le **tastiere con funzioni di sintesi** includono, all'interno di un unico dispositivo, tanto un *controller* atto a generare i messaggi MIDI quanto un sintetizzatore che li riceve e interpreta, originando così il suono. In pratica, tali dispositivi implementano una catena MIDI minimale. Queste unità sono anche in grado di inviare i messaggi MIDI generati ad altri dispositivi collegati, come pure di disaccoppiare le funzioni di *controller* e di *synth* attraverso lo spegnimento del cosiddetto controllo locale (*local control*), come descritto nel Paragrafo 2.7.8.

Riguardo ai dispositivi destinatari di flussi MIDI, si evidenzia, infine, come pure un *sequencer* possa rientrare a pieno titolo nella categoria. Nei *sequencer* i messaggi vengono ricevuti per poi essere opportunamente gestiti: ad esempio, manipolati, salvati o reimmessi sul canale trasmissivo.

La comunicazione MIDI non riguarda solo la connessione di un *controller* a un sintetizzatore. Al contrario, il protocollo consente di collegare un gran numero di dispositivi diversi secondo differenti schemi di connessione, che verranno approfonditi nel Paragrafo 1.7.

1.5 Ricevitori e trasmettitori

Nel Paragrafo 1.3 si è fatto cenno al ruolo di mittente e/o destinatario di un dispositivo all'interno del *setup* MIDI. Per rispondere a tali funzioni, uno strumento può contenere:

- un **trasmettitore** (*transmitter*), ossia un componente atto a trasmettere messaggi in formato MIDI ai dispositivi MIDI posti a valle;
- un **ricevitore** (*receiver*), ossia un componente in grado di ricevere messaggi in formato MIDI dai dispositivi posti a monte.

Nelle specifiche MIDI 1.0, tra le funzioni di un ricevitore, si elenca anche la capacità di analizzare i messaggi in arrivo, e tra quelle di un trasmettitore si individua anche la capacità di

generare i messaggi da immettere sul canale trasmissivo. Le funzioni svolte da questi componenti dunque includono operazioni legate non solo alla comunicazione (invio/ricezione), ma anche alla logica (produzione/analisi) dei messaggi.

Non sempre i dispositivi MIDI dispongono di entrambi i componenti descritti. Ad esempio, le funzioni svolte da un *controller* non richiedono strettamente la presenza di un ricevitore per gestire i messaggi in arrivo: è sufficiente un trasmettitore per immettere messaggi MIDI sul canale trasmissivo in risposta alle azioni del musicista. Allo stesso modo un sintetizzatore, il cui scopo principale è ricevere i messaggi MIDI relativi alla performance e trasformarli in suoni, necessita della sola presenza del ricevitore. Questo non esclude, però, che esistano *controller* dotati di ricevitori e *synth* equipaggiati con trasmettitori.

Il rapporto tra un trasmettitore e un ricevitore di messaggi MIDI, e – per estensione – il rapporto tra i dispositivi che contengono tali componenti, viene talvolta descritto come *master-slave*. Nella relazione che si instaura tra dispositivi, chi invia i messaggi (*master*) controlla chi li riceve (*slave*). Tale terminologia si ritrova, ad esempio, nella definizione di *master keyboard*, ossia una tastiera il cui fine è controllare uno o più sintetizzatori sulla catena MIDI a valle.

1.6 Porte e connettori

Nella sua definizione più completa, una porta MIDI include 3 **connettori**, che svolgono le seguenti funzioni:

1. **MIDI IN**, il connettore cui arrivano i messaggi in ingresso dal *setup* MIDI a monte del dispositivo;
2. **MIDI OUT**, il connettore tramite cui il dispositivo immette sul *setup* MIDI a valle i messaggi da esso generati;
3. **MIDI THRU** (dall'inglese *through*), il connettore che replica in uscita i messaggi che il dispositivo riceve in ingresso sul connettore MIDI IN.

A ciascun connettore è possibile collegare al più un cavo MIDI, aspetto che verrà approfondito nel Paragrafo 1.7.

In analogia con quanto detto nel Paragrafo 1.5, non tutti i dispositivi presentano porte MIDI complete, ossia dotate dei 3 connettori citati. Ad esempio, un *controller* a tastiera senza funzioni di sintesi ha il solo scopo di tradurre il gesto del musicista in messaggi da immettere sul canale trasmissivo; non di rado questa categoria di strumenti ha una porta MIDI costituita dal solo connettore MIDI OUT. Invece, una tastiera con funzioni di sintesi può generare audio non solo in risposta alla propria interfaccia di input (il cosiddetto controllo locale), ma anche a partire da messaggi MIDI provenienti dal *setup* a monte; per tale motivo la porta presenta anche il connettore MIDI IN.

È comune che la presenza di un connettore MIDI IN implichi anche quella del MIDI THRU, cosicché il dispositivo possa risultare "trasparente" rispetto alla propagazione dei messaggi MIDI nella catena. Infatti MIDI THRU, ove presente, replica i soli messaggi ricevuti dal dispositivo sul proprio MIDI IN, senza dunque considerare gli eventuali messaggi generati dal dispositivo stesso. Si pensi a un sintetizzatore, che per svolgere le proprie funzioni dovrà avere almeno un connettore MIDI IN. Affinché il sintetizzatore non costituisca unicamente un punto terminale della catena MIDI, ma consenta di propagare i messaggi (ad esempio a un ulteriore sintetizzatore posto a valle), la sua porta MIDI dovrà presentare anche un connettore MIDI THRU. In tal modo il sintetizzatore non solo riceverà i messaggi MIDI e genererà suono di conseguenza, adempiendo al proprio compito, ma permetterà anche che questi fluiscano senza alcuna alterazione ai dispositivi collegati in cascata.

In alcuni casi la differenza tra MIDI OUT e MIDI THRU è sfumata, come per la categoria di dispositivi detti *expander*, *patch bay* o *thru box*. Questi strumenti presentano più MIDI IN e più

Figura 1.7. Expander *Yamaha YME-8*.

MIDI OUT/THRU, allo scopo di replicare i messaggi in ingresso a un singolo MIDI IN su più MIDI OUT/THRU (*routing*), o di incanalare i messaggi di più MIDI IN su un unico MIDI OUT/THRU (*merging*). Per tali dispositivi, MIDI OUT e MIDI THRU dal punto di vista logico coincidono. Infatti, se da un lato la funzione del dispositivo è operare la replicazione/fusione di messaggi MIDI (in questo senso, il risultato dell'operazione svolta viene messo a disposizione sul connettore MIDI OUT), dall'altro lato si tratta di una replica di quanto arriva in MIDI IN (in questo senso, è più opportuno parlare di MIDI THRU). La Figura 1.7 mostra un expander dotato di 2 connettori MIDI IN e 8 MIDI OUT/THRU, con cui è possibile tanto fare *routing* quanto *merging* dei messaggi in ingresso.

Un altro esempio per approfondire la differenza tra MIDI IN, OUT e THRU riguarda le tastiere con funzioni di sintesi. Questa categoria di dispositivi tipicamente presenta porte MIDI dotate dei 3 connettori, in quanto la tastiera deve essere in grado di: 1. ricevere messaggi dall'esterno attraverso MIDI IN per inoltrarli al proprio *synth*, 2. immettere solo i propri messaggi sul canale trasmissivo via MIDI OUT, 3. replicare i soli messaggi ricevuti in ingresso tramite MIDI THRU. Questa configurazione, unita alla possibilità di abilitare o disabilitare il controllo locale, offre la massima flessibilità. Ad esempio, disaccoppiando la tastiera dal *synth* (*local control off*), si pongono i seguenti scenari:

- il sintetizzatore potrebbe comunque generare suono sulla base dei messaggi in arrivo dalla catena di dispositivi a monte sul proprio MIDI IN;
- il *controller*, pur non controllando il *synth*, originerebbe messaggi inviati sulla catena di dispositivi a valle sul proprio MIDI OUT;
- il dispositivo potrebbe inoltrare i messaggi in arrivo su MIDI IN attraverso MIDI THRU, escludendo ovviamente quelli generati dal *controller* stesso (che però, come detto, vengono resi disponibili su MIDI OUT).

Come detto, la porta MIDI standard si compone di 3 connettori, non tutti necessariamente presenti. Esistono poi dispositivi MIDI multi-porta, dotati per definizione di più porte, ossia più raggruppamenti logici di connettori. Un esempio di dispositivo multi-porta è già stato discusso: l'expander *Yamaha YME-8*, dotato di 2 MIDI IN e 8 MIDI OUT/THRU, rientra chiaramente nella categoria. Un altro esempio è dato dal sintetizzatore politimbrico *Yamaha MU80* che presenta due connettori MIDI IN identificati, rispettivamente, come MIDI IN A e B. Il vantaggio di disporre di più porte sta nella possibilità di collegare il dispositivo a più catene indipendenti.

Figura 1.8. Una catena MIDI minimale: collegamento di un *controller* a un *synth*.

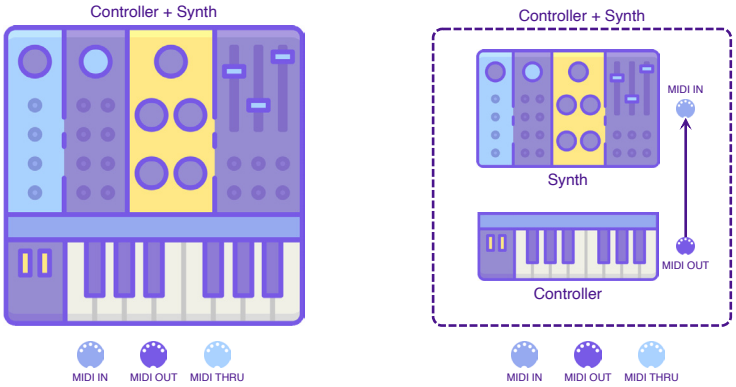


Figura 1.9. Rappresentazione grafica di un *controller* con funzioni di sintesi del suono (a sinistra) e dei suoi moduli logici (a destra).

1.7 Principali schemi di collegamento

I messaggi trasmessi a un dispositivo possono entrarvi solo attraverso il connettore MIDI IN, mentre possono uscire via MIDI OUT (se sono generati dal dispositivo stesso) e MIDI THRU (se sono repliche degli input). Pertanto, al fine di garantire una corretta comunicazione, un cavo MIDI dovrà essere connesso a MIDI OUT o MIDI THRU a una delle estremità, e, all'altra, invariabilmente a MIDI IN. Si osservi che i cavi MIDI sono simmetrici e i connettori hanno tutti la stessa forma, quindi non esistono impedimenti fisici alla realizzazione di collegamenti logicamente errati, quali un collegamento da MIDI THRU a MIDI OUT o un collegamento tra i MIDI OUT di due dispositivi.

Un esempio di catena MIDI minimale, costituita da una tastiera (*master*) che controlla un sintetizzatore (*slave*), viene mostrato nella Figura 1.8. Un caso particolare è quello delle tastiere con funzioni di sintesi del suono, che sono in grado di interfacciarsi con lo strumentista, leggerne il gesto e tradurlo in messaggi MIDI (modulo di controllo), come pure di produrre suono (modulo di sintesi). Per questi dispositivi la catena *master-slave* si trova implementata all'interno di un singolo dispositivo (Figura 1.9); la funzione *local control on/off* permette di abilitare o disabilitare l'invio di messaggi interni tra il *controller* e il synth.

Lo schema di collegamento più semplice è quello che coinvolge n dispositivi in cascata. Si sottolinea che la comunicazione nella versione 1.0 del protocollo MIDI è unidirezionale, quindi i cavi che collegano i dispositivi attraverso i loro connettori trasportano messaggi dai dispositivi a monte ai dispositivi a valle. Volendo creare una forma di retroazione tra mittente e destinatario, è necessario utilizzare due cavi distinti, impegnando altrettanti connettori su ciascuno strumento. In informatica, l'interconnessione di apparecchiature in serie tra loro viene detta *daisy chain*, letteralmente "catena di margherite". In MIDI questo schema si attua partendo da una sorgente di messaggi MIDI, ad esempio un *controller* o un *sequencer*, cui viene collegato un cavo che congiunge il proprio MIDI OUT con il MIDI IN di un primo dispositivo destinatario, che a sua volta replica i

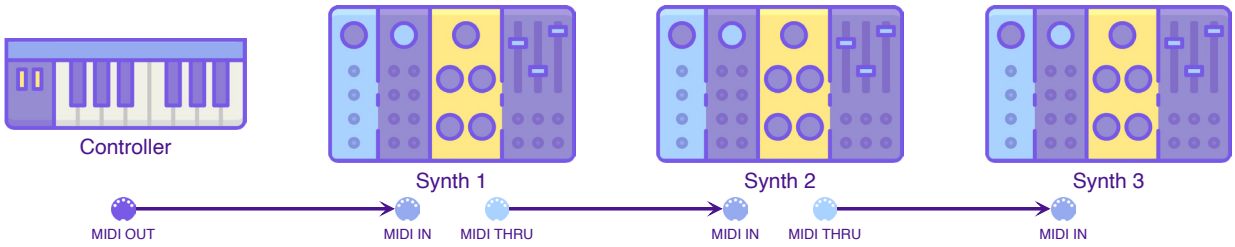


Figura 1.10. Un esempio di *daisy chain* in cui i messaggi di un *controller* vengono propagati a tre *synth* collegati in serie. Si osservi l'uso del MIDI THRU per i sintetizzatori.

messaggi MIDI grazie a un cavo posto tra il proprio MIDI THRU e il MIDI IN del successivo, e via dicendo. Quindi, nello schema di connessione più basilare, si ha un'unica sorgente *master* che invia i messaggi agli n dispositivi *slave* a valle; per questo motivo tutti i destinatari propagano i messaggi attraverso MIDI THRU anziché MIDI OUT. Un esempio di daisy chain che coinvolge un *controller* a tastiera e tre *synth* è mostrato nella Figura 1.10.

Dal punto di vista terminologico, quando lo schema di collegamento tra dispositivi è lineare, è corretto parlare di catena MIDI (*MIDI chain*); per schemi di connessione non lineari, come quelli descritti nel seguito, è meglio usare la locuzione più generale di "configurazione MIDI" (*MIDI setup*). Il concetto di "sistema MIDI", per quanto ampiamente utilizzato in luogo di *setup*, rischia di ingenerare ambiguità tra singolo dispositivo e insieme di apparati hardware e software in comunicazione tra loro, motivo per il quale verrà evitato.

Un problema legato all'interconnessione in serie è quello delle latenze, introdotte non solo dalla propagazione del segnale elettrico lungo i cavi, ma anche dai connettori e dalla circuiteria interna ai dispositivi. Si consideri che un ritardo considerato tollerabile nelle DAW è dell'ordine dei millisecondi. Creare lunghe connessioni *daisy chain* è quindi sconsigliabile.

Come è possibile, ad esempio, ripensare allo schema nella Figura 1.10 mantenendone le funzionalità senza ricorrere alla *daisy chain*? Vengono in soccorso le già citate *patch bay*, con la loro capacità di replicare i messaggi in ingresso su più porte in uscita. La Figura 1.11 mostra una soluzione alternativa allo schema illustrato nella Figura 1.10 che riduce le latenze.

Come ulteriore caso di catena non lineare, si voglia realizzare uno schema con comunicazione bidirezionale tra dispositivi. Un esempio pratico, da questo punto di vista, vede l'impiego di un *sequencer*, tanto come generatore di messaggi MIDI per una tastiera con funzioni di sintesi, quanto come ricevitore dei messaggi prodotti dalla tastiera stessa. Tale collegamento impegna necessariamente i connettori MIDI IN e MIDI OUT di entrambi i dispositivi. Lo schema può essere ulteriormente complicato dalla presenza di altri sintetizzatori a valle della tastiera, che però saranno destinatari dei soli messaggi provenienti dal *sequencer*, in quanto l'unico connettore libero sulla tastiera (nel caso di una sola porta MIDI) è MIDI THRU. La Figura 1.12 mostra il corrispondente schema di collegamento. Se il controllo locale della tastiera risultasse disattivato, i messaggi generati dal *controller* non verrebbero letti dal *synth* ma unicamente passati al *sequencer* via MIDI OUT, che a sua volta potrebbe miscelarli a quelli della sequenza già disponibile e inviarli attraverso il proprio MIDI OUT al *synth* che li riceverà tramite il MIDI IN del dispositivo.

Si considerino, infine, le possibilità offerte dai dispositivi multi-porta che non rientrano nella categoria delle *patch bay*. Un sintetizzatore dotato di due porte MIDI (e quindi tipicamente di due MIDI IN) permette di realizzare schemi a Y rovesciata, come quello mostrato nella Figura 1.13, in cui più strumentisti dotati di *master keyboard* controllano lo stesso modulo sonoro. Se ciascuna porta fosse dotata anche di MIDI THRU, lo schema diventerebbe a X, come se fosse costituito da due catene lineari indipendenti che condividono un nodo per poi tornare a separarsi.

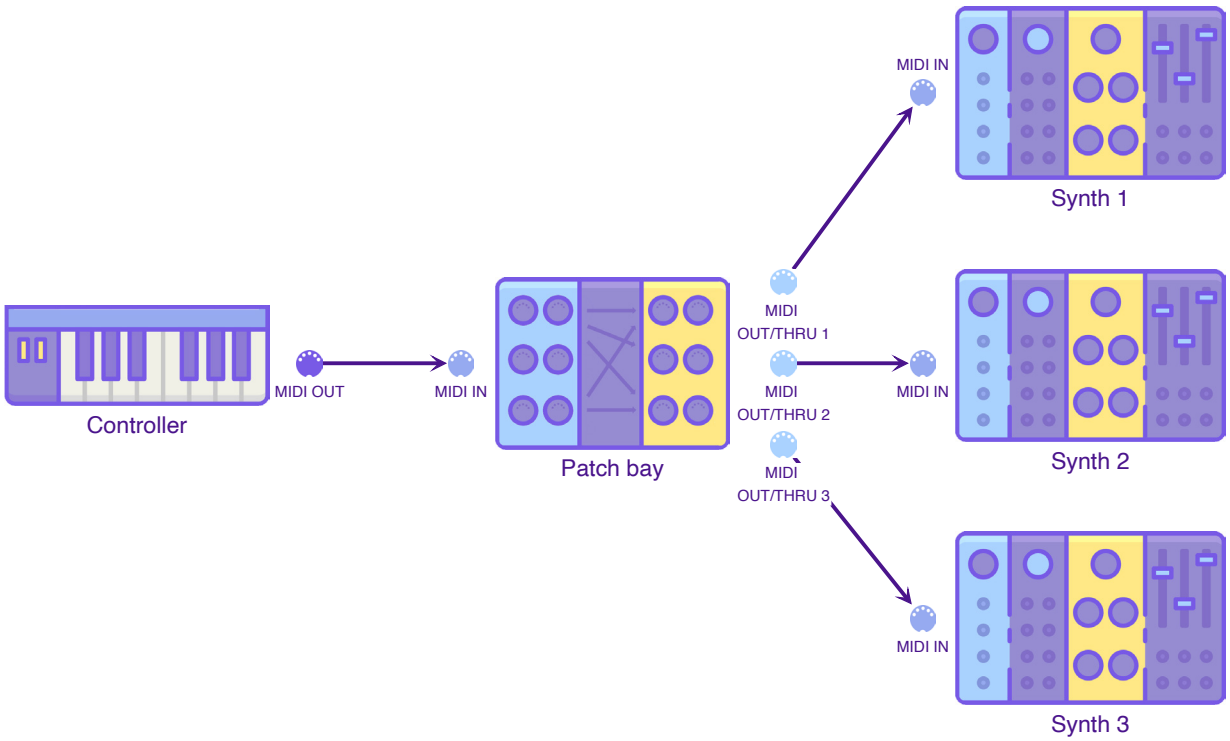


Figura 1.11. Alternativa alla *daisy chain* di Figura 1.10 utilizzando una *patch bay*.

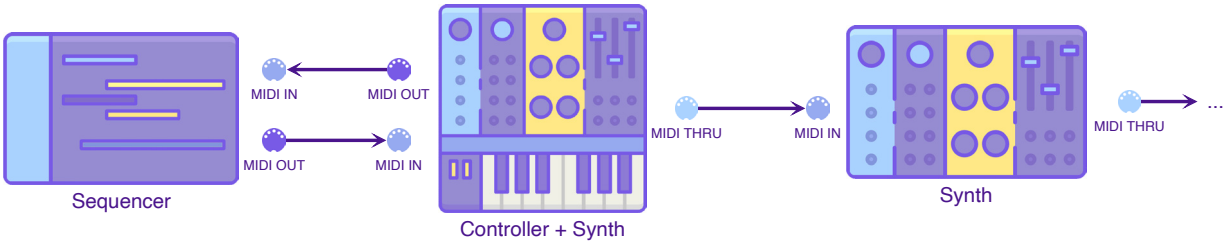


Figura 1.12. Un *setup* MIDI con comunicazione bidirezionale tra dispositivi.

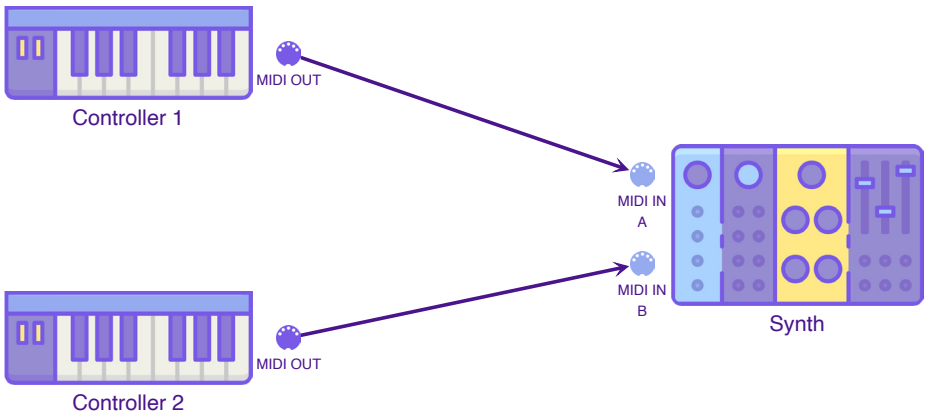


Figura 1.13. Un *setup* MIDI con schema a Y rovesciata, consentito dalla disponibilità di più porte MIDI nel sintetizzatore.

1.8 Aspetti elettrici e circuitali

Riguardo a porte e cavi, lo standard MIDI si basa su **connettori circolari DIN a 5 pin disposti a 180°** [4]. Le porte presentano connettori femmina, mentre i cavi normalmente hanno connettori maschio identici tra loro a entrambe le estremità [5].

Lo standard DIN a 5 pin era molto diffuso in Europa per ogni tipo di connessione audio, ma negli anni '90 tali connettori sono stati rimpiazzati dai più pratici jack. Anche le tastiere dei personal computer e i mouse, attualmente dotati in prevalenza di connessione USB, in passato utilizzavano lo standard DIN a 5 pin, in seguito abbandonato in favore del mini-DIN a 6 pin delle porte PS/2. Al giorno d'oggi, il MIDI costituisce uno dei pochi campi di applicazione per lo standard DIN a 5 pin.

La descrizione delle specifiche e i circuiti di esempio contenuti nel documento "MIDI 1.0 Electrical Specification" del 1983 [27] utilizzavano elettronica a 5 V, comune a quel tempo. Nel 2014 le specifiche sono state aggiornate per riflettere gli attuali requisiti di progettazione elettronica, tra cui i circuiti a 3,3 V [29].

L'interfaccia hardware MIDI opera a $31.25\,\mathrm{kBd}\ \pm 1\%$, in modo asincrono, con un bit di avvio (*start*), 8 bit di dati e un bit di arresto (*stop*). Il protocollo non prevede l'invio di bit di parità. Ciò produce un pacchetto di 10 bit in un periodo di $320\,\mu\mathrm{s}$ per byte seriale. Considerando il livello fisico del modello OSI, la rappresentazione dei bit è affidata ai valori di intensità di corrente: 1 logico corrisponde al livello di 0 mA (off), 0 logico al livello di 5 mA (on). All'interno del pacchetto, il bit di avvio è uno 0 logico e il bit di arresto è un 1 logico. I bit vengono inviati nell'ordine dal meno al più significativo. La Figura 1.14 esemplifica in forma grafica l'invio di un pacchetto di dati MIDI costituito da un ottetto di bit preceduto e seguito corrispettivamente dal bit di avvio e di arresto. Si noti che per i bit numerati da 0 a 7 c'è possibilità di scelta tra valore alto e basso di intensità di corrente, mentre il valore è sempre alto per il bit di avvio e sempre basso per quello di arresto.

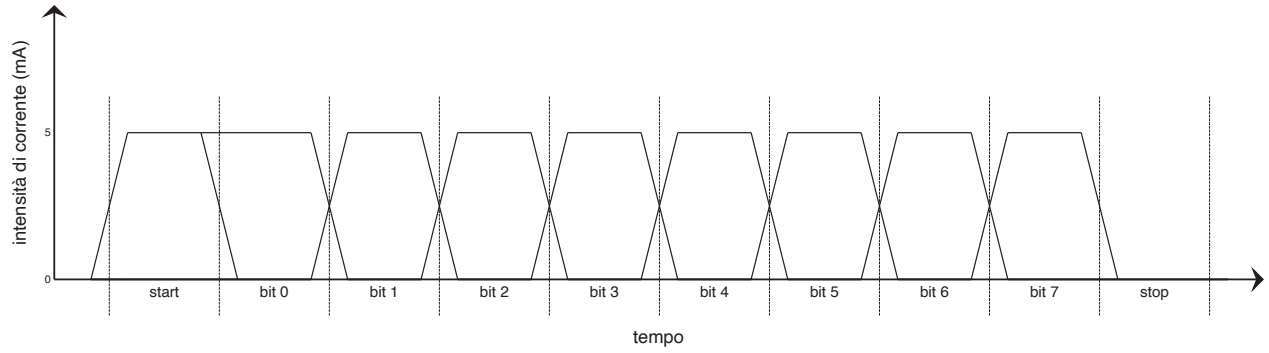


Figura 1.14. Invio di un ottetto di dati MIDI con bit di avvio e arresto e valori di intensità elettrica.

Secondo le specifiche il cavo MIDI deve essere lungo al massimo 15 m. A ciascuna estremità esso presenta un connettore maschio DIN a 5 pin, i cui piedini vengono numerati convenzionalmente da 1 a 5, secondo l'ordine mostrato nella Figura 1.15. Il cavo deve essere schermato elettromagneticamente, con la schermatura collegata al pin 2 a entrambe le estremità. Al suo interno si trova una coppia di conduttori ritorti (*twisted pair*), collegati, rispettivamente, ai pin 4 e 5. Sui connettori posti alle estremità del cavo, il pin 4 risulta collegato con il pin 4 e il pin 5 con il pin 5. Le specifiche tecniche non prevedono collegamenti sui pin 1 e 3, ma non escludono un loro possibile uso futuro (che non si è finora concretizzato).

Le specifiche MIDI 1.0 per le interfacce elettriche [27] prevedono un anello di corrente (*current loop*) completamente isolato. Il connettore MIDI OUT nominalmente invia +5 V attraverso un resistore da 220 Ω sul pin 4 del cavo DIN; il segnale elettrico viene ricevuto sul pin 4 del cavo DIN da parte del connettore MIDI IN del dispositivo ricevente, giungendo a un resistore di protezione da

[4] DIN è l'acronimo di *Deutsches Institut für Normung* (Istituto tedesco per la standardizzazione).

[5] Esistono cavi che fungono da prolunga e presentano un connettore maschio e un connettore femmina.

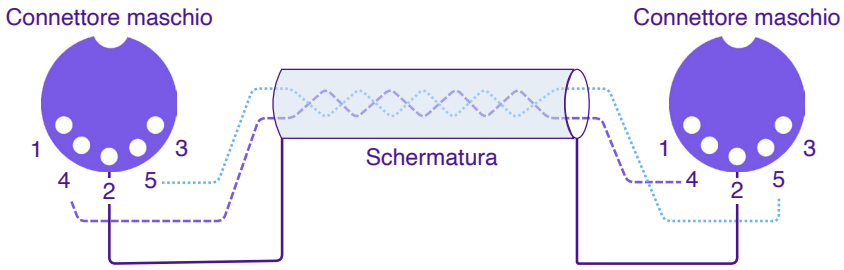


Figura 1.15. Rappresentazione di un cavo MIDI con connettori DIN a 5 pin.

220 Ω e al LED di un optoisolatore [6]. La corrente viene quindi rimandata indietro attraverso il pin 5 della porta MIDI IN fino a raggiungere il pin 5 dell'originaria porta MIDI OUT, trovando ancora una volta sul proprio cammino un resistore da 220 Ω, il che fornisce una corrente nominale di circa 5 mA. Nonostante il cavo conduca corrente tra i connettori, in realtà tra i dispositivi si realizza un isolamento ottico tale da renderli relativamente immuni da anelli di massa (*ground loops*) e da fenomeni di interferenza.

Allo standard DIN a 5 pin si sono via via affiancate altre possibilità di collegamento, progettate per connettere dispositivi di elaborazione privi di porte MIDI standard. Nei computer, queste ultime tipicamente si trovano solo su schede audio dedicate, il che – soprattutto in passato – sarebbe stato un fattore di forte limitazione alla diffusione di software compatibile con MIDI. Un primo passo in questa direzione è stata l'adozione della *game port*, il tradizionale connettore per dispositivi di input quali joystick e gamepad nei PC con architettura x86. Il connettore DA-15 della porta giochi presenta alcuni piedini inutilizzati di tipo +5 V e massa (GND), due dei quali possono essere riconvertiti a MIDI IN (pin 15) e MIDI OUT (pin 12). In tal modo è possibile collegare alla *game port* tanto una periferica di gioco quanto uno strumento MIDI, usando un cavo DA-15 maschio a un'estremità e dotato, all'altra estremità, di un connettore DA-15 femmina e due connettori DIN a 5 pin maschi (1.16).

Un'alternativa affermatasi in tempi più recenti è il collegamento via USB. I cavi USB MIDI hanno tipicamente un connettore maschio USB A a un'estremità e una coppia di connettori maschi MIDI IN e MIDI OUT all'altra. Tali cavi, quindi, trasportano fisicamente messaggi nelle due direzioni, ma all'estremità dotata di connettori MIDI tornano a essere due cavi distinti e unidirezionali. Le operazioni per una corretta gestione dei messaggi vengono delegate a un'interfaccia MIDI-USB posta tra le estremità del cavo, come mostrato nella Figura 1.17.

Altri protocolli di trasporto per i messaggi MIDI sono RTP-MIDI, che sfrutta le reti Ethernet, e Bluetooth LE MIDI. Per una loro descrizione dettagliata si rimanda al sito midi.org [7].

[6] Un optoisolatore, noto anche come optocoppia o fotoaccoppiatore, è un componente elettronico che permette di trasferire un segnale fra due circuiti mantenendo l'isolamento galvanico tra essi. Esso normalmente si realizza accoppiando otticamente un LED con un elemento fotosensibile.

[7] https://www.midi.org/specifications-old/category/transport-specifications-and-info

Figura 1.16. Cavo di interfacciamento tra porta MIDI e porta giochi.

Figura 1.17. Interfaccia MIDI-USB *Roland UM-ONE Mk2*.

Capitolo 2

I messaggi MIDI

Questo capitolo fornisce una descrizione puntuale dei messaggi MIDI standardizzati nella versione 1.0 del protocollo. Si introduce il concetto di canale MIDI (Paragrafo 2.1) e la differenza tra byte di stato e byte di dati (Paragrafo 2.2). Nel seguito del capitolo, in particolare nei Paragrafi da 2.3 a 2.10, i messaggi vengono trattati secondo la loro organizzazione in famiglie, in accordo con le specifiche MIDI. Vista la sua specificità, il messaggio CONTROL CHANGE viene affrontato in una sezione apposita (Paragrafo 2.5). Nel Paragrafo 2.11 si descrive, infine, la trasmissione *running status*, grazie alla quale si può ottenere un uso più efficiente della banda durante una performance MIDI.

2.1 I canali

La nozione di **canale** è di fondamentale importanza per il protocollo MIDI. MIDI 1.0 supporta 16 canali per l'invio e la ricezione di messaggi, convenzionalmente numerati da 1 a 16 anziché da 0 a 15 secondo una consuetudine più vicina al mondo della musica che a quello dell'informatica e della tecnologia. A livello di codifica interna ai messaggi, il Canale 1 sarà comunque rappresentato dal valore 0, il Canale 2 dal valore 1, e via dicendo. Il limite dei 16 canali è stato recentemente superato dalle specifiche 2.0, che portano il totale disponibile a 256.

Per semplicità il lettore può temporaneamente ritenere che ciascun messaggio, quale, ad esempio, l'accensione o lo spegnimento di una nota, rechi informazione sul canale di appartenenza. Questa affermazione si rivelerà corretta per i messaggi relativi alla performance musicale, ma, come approfondito nel seguito, non tutti i messaggi supportati dal protocollo risulteranno agganciati a un canale. Va inoltre precisato che il concetto di canale introduce una separazione puramente logica, perché dal punto di vista fisico i messaggi scambiati tra due dispositivi e afferenti a canali diversi condividono lo stesso mezzo trasmissivo, coabitando all'interno di un'unica trasmissione seriale.

Una metafora che può favorire la comprensione è rappresentata dal modello di trasmissione tipico del mezzo televisivo: il *broadcasting*. Nell'etere vengono simultaneamente diffusi i segnali di differenti emittenti, le antenne ricevono tali segnali e gli apparecchi TV possono selettivamente sintonizzarsi su una delle emittenti. In modo analogo, su un singolo cavo MIDI vengono trasmessi messaggi relativi a tutti i canali, oltre a messaggi che non presentano informazione di canale. Come i televisori, i dispositivi MIDI possono attuare l'ascolto selettivo di un singolo canale, ma in aggiunta possono gestire anche messaggi in arrivo su più canali.

Un singolo connettore MIDI OUT in un dispositivo mittente consente di inviare tanto messaggi di canale (per la precisione, di qualsiasi canale) quanto messaggi non di canale interfogliati all'interno di una stessa trasmissione seriale. Dal punto di vista del dispositivo destinatario, il connettore MIDI IN riceve tutti i messaggi, anche nel caso in cui il destinatario sia in ascolto selettivo su un dato canale, e il connettore MIDI THRU, se impiegato, li propaga nella loro interezza sulla catena a valle.

Per i messaggi che richiedono di specificare un canale, i dispositivi trasmettitori in genere permettono di selezionarlo attraverso opportuni controlli. Come detto, un caso tipico di messaggi di canale è dato dalle informazioni sulla performance musicale, quali l'accensione e lo spegnimento di una nota. I *controller* a tastiera talvolta presentano 16 pulsanti mutuamente esclusivi associati ai canali, oppure consentono di immettere valori numerici tramite combinazioni di tasti, al fine di determinare il canale cui associare i messaggi MIDI prodotti in risposta alle azioni dello strumentista.

Il protocollo prevede che più dispositivi all'interno di un *setup* MIDI possano trasmettere messaggi sullo stesso canale. Inoltre alcuni strumenti sono in grado di gestire simultaneamente più canali per i messaggi di performance. I *controller* a tastiera multi-parte possono presentare, ad esempio, funzioni di *split* che consentono di raggruppare tasti contigui in zone assegnando a ciascuna di esse un diverso canale; se nel sintetizzatore tali canali vengono gestiti con *patch* differenti, un tastierista può controllare con la mano destra e la sinistra due diversi timbri.

Il canale specifico su cui i dispositivi MIDI si pongono in ascolto dei messaggi [1] è detto **canale base** (*basic channel* o *base channel*). Si tratta di uno dei 16 canali previsti dal protocollo, e in genere può essere selezionato sia attraverso interfaccia fisica sia inviando opportuni comandi MIDI della famiglia *System Exclusive* (si veda il Paragrafo 2.10). Le specifiche suggeriscono di impostare il canale base a 1 all'accensione dei dispositivi. Indipendentemente dal canale base selezionato, uno strumento può ricevere messaggi MIDI su più di un canale, incluso il canale base stesso, che da questo punto di vista non si differenzia dagli altri 15. Il ruolo del canale base verrà approfondito nel Paragrafo 2.7 dedicato alla famiglia di messaggi *Channel Mode*.

Per superare il limite dei 16 canali MIDI sono state prodotte interfacce hardware multi-porta. Poiché ogni porta è in grado di supportare 16 canali MIDI indipendenti, un dispositivo dotato di n porte è in grado di gestire $16 \times n$ canali. Questo approccio si rivela utile, ad esempio, per i sintetizzatori multi-timbrici in grado di gestire con timbriche differenti più di 16 voci, ossia 16 eventi sonori contemporanei; infatti, a un dato istante di tempo e in un dato sintetizzatore, un canale può risultare associato a un'unica *patch*, pertanto eventi sonori simultanei appartenenti a 16 canali diversi non possono presentare più di 16 timbri differenti. Risulta invece possibile avere più note simultanee sullo stesso canale (quindi associate allo stesso timbro), associare più canali nel sintetizzatore a una stessa *patch*, determinare associazioni differenti tra timbri e canali nei vari moduli sonori presenti nel *setup*, e modificare dinamicamente per ciascun sintetizzatore l'associazione timbro-*patch* nel tempo.

2.2 Byte di stato e byte di dati

Nel protocollo MIDI la comunicazione ha luogo attraverso uno scambio di messaggi costituiti da uno o più byte, ossia tramite sequenze di valori binari espressi su 8 bit. Come chiarito precedentemente, da qui in avanti si farà riferimento agli ottetti di bit, anche se sul canale trasmissivo, dal punto di vista elettrico, vengono inviati per ciascun byte 10 bit, che contengono anche un flag di start e uno di stop.

Ogni messaggio è formato da un byte di stato (*status byte*), seguito da un numero variabile di byte di dati (*data byte*). I **byte di stato** hanno principalmente lo scopo di identificare il tipo di messaggio, il che conferisce inoltre un significato specifico ai byte di dati successivi. Ad eccezione dei messaggi appartenenti alla famiglia *Real Time*, l'arrivo di un nuovo byte di stato forza il ricevitore a considerare il nuovo messaggio, anche se la ricezione di quello precedente non è completa. Si tratta, evidentemente, di una situazione indesiderata.

I **byte di dati** codificano valori numerici necessari allo svolgimento dell'operazione determinata dal byte di stato, il cui significato dipende dal tipo di messaggio. Nella maggior parte dei casi il numero di byte di dati relativi a un byte di stato è pari a 1 o 2, ma alcune famiglie di messaggi

[1] Si tratta, più propriamente, dell'ascolto dei messaggi di canale; altri messaggi, essendo potenzialmente rivolti all'intero *setup* MIDI, vengono comunque ricevuti.

comportano l'invio del solo byte di stato (come nel caso dei messaggi *Real Time*) o, al contrario, di un numero significativo di byte di dati (come nel caso dei messaggi *System Exclusive*).

Un byte di stato si distingue da un byte di dati attraverso il suo bit più significativo (*most significant bit*), che nella notazione posizionale comunemente adottata è la cifra binaria più a sinistra nell'ottetto. Il bit più significativo viene posto a 1 per i byte di stato e a 0 per i byte di dati. Pertanto l'intervallo di valori possibili è:

- $[10000000_2, 11111111_2]$ oppure $[80_{16}, FF_{16}]$ per un byte di stato;
- $[00000000_2, 01111111_2]$ oppure $[00_{16}, 7F_{16}]$ per un byte di dati.

Ne risulta che lo spazio utile (*payload*) per codificare il messaggio si limita ai rimanenti 7 bit. Da qui l'ampio utilizzo in MIDI degli insiemi a $2^7 = 128$ valori, ad esempio per rappresentare le altezze (*pitch*), i livelli di intensità (*velocity*), i timbri (*program* o *patch*), e via dicendo.

In generale il protocollo MIDI prevede l'invio seriale del byte di stato seguito dagli eventuali byte di dati corrispondenti. Esistono però due eccezioni notevoli. La prima è costituita dai messaggi della famiglia *Real Time*, costituiti dal solo byte di stato: per via delle esigenze temporali stringenti ad essi collegate, questi messaggi possono essere immessi sul canale trasmissivo anche suddividendo messaggi a più byte. Estrarli dalla sequenza e ricostruire i messaggi eventualmente interrotti è comunque un'operazione tecnicamente semplice e non ambigua, come verrà spiegato nel Paragrafo 2.9.

La seconda eccezione si riferisce alla cosiddetta codifica *running status*, applicabile solo a determinate famiglie di messaggi MIDI. Quando un byte di stato viene ricevuto ed elaborato, il destinatario tiene in memoria lo stato corrente fino a quando non viene ricevuto un byte di stato diverso. Se il mittente si trova a dover ripetere lo stesso byte di stato di messaggio in messaggio, può (facoltativamente) inviare solo il primo e omettere quelli successivi, limitandosi ai byte di dati. La conseguenza è una minore occupazione del canale trasmissivo, problema molto sentito in ambito MIDI. Quando viene applicata la tecnica *running status*, un messaggio completo può essere composto solo da byte di dati, purché sia stato inviato in precedenza un byte di stato. L'argomento verrà approfondito nel Paragrafo 2.11.

Il protocollo MIDI prevede che i dispositivi possano ignorare lo stato dei messaggi che non sono ritenuti significativi per il loro funzionamento. Un chiaro esempio è offerto dal messaggio TUNE REQUEST, che richiede ai *synth* MIDI di eseguire una procedura di ri-accordatura (Paragrafo 2.8.4): si tratta di un'operazione sensata per i sintetizzatori analogici ma non per quelli digitali, che dunque non implementano una risposta a TUNE REQUEST.

2.3 Categorie e famiglie di messaggi

I messaggi MIDI si suddividono in due categorie principali: messaggi di canale e di sistema.

I **messaggi di canale** (*channel messages*) sono destinati ai dispositivi in ascolto sul canale codificato nel byte di stato. A tale scopo un messaggio di canale utilizza i bit meno significativi del proprio byte di stato per identificare il canale su cui viene instradato. Essendo 16 i canali MIDI disponibili, sono sufficienti allo scopo i 4 bit meno significativi del byte di stato. Quest'ultimo quindi si presenta nella forma

$$1xxxnnnn_2$$

ove nnnn rappresenta la codifica in base 2 del canale. Per via delle convenzioni MIDI, si osservi che 0000_2 corrisponde al Canale 1, 0001_2 al Canale 2, e via dicendo fino a 1111_2 per il Canale 16. Le restanti cifre binarie, rappresentate da xxx_2, sono l'identificativo del comando MIDI codificato dal messaggio.

Risulta evidente che, se i messaggi MIDI fossero tutti di canale, ossia richiedessero di dedicare 4 bit all'informazione di canale, il protocollo contemplerebbe al massimo $2^3 = 8$ messaggi differenti, poiché 3 sono le cifre binarie riservate allo scopo all'interno del byte di stato. In realtà, come

specificato più avanti, la combinazione di cifre binarie 111_2 denota un'altra categoria di messaggi, detti di sistema, ove è possibile recuperare i 4 bit meno significativi del byte di stato per specificare 16 ulteriori comandi [2].

All'interno della categoria dei messaggi di canale, si riconoscono due famiglie:

1. **messaggi di canale relativi alle voci** (*Channel Voice messages*). Sono principalmente dedicati alla performance musicale, determinando, ad esempio, l'accensione e lo spegnimento di una nota o l'associazione del timbro a un dato canale. In questo caso, l'informazione di canale serve a poter inviare i messaggi di performance su canali distinti, permettendo di associare, ad esempio, un timbro o una variazione di intensità sonora potenzialmente differenziati canale per canale;

2. **messaggi di canale relativi al modo** (*Channel Mode messages*). Definiscono il modo di funzionamento di uno strumento. In questo caso l'informazione di canale viene utilizzata per controllare tutti e soli i dispositivi nel *setup* che hanno come canale base quello specificato nel messaggio.

La seconda categoria, quella dei **messaggi di sistema** (*system messages*), non richiede di specificare informazione di canale, in quanto i suoi messaggi sono potenzialmente rivolti all'intero *setup* MIDI. Esistono tre famiglie di messaggi di sistema:

3. **messaggi di sistema comuni** (*System Common messages*). Sono rivolti a tutti i dispositivi in ricezione, in maniera indipendente dal canale;

4. **messaggi di sistema in tempo reale** (*System Real Time*). Presentano requisiti temporali stringenti e sono anche utilizzati per implementare una particolare forma di sincronizzazione detta *timing clock*, illustrata nel seguito;

5. **messaggi di sistema esclusivi** (*System Exclusive messages*). Spesso detti semplicemente *SysEx*, sono destinati a dispositivi specifici all'interno del *setup*, e per questo in genere recano informazioni sul produttore e sul modello. I messaggi della famiglia possono contenere migliaia di byte di dati, la cui interpretazione dipende dalle scelte progettuali del singolo dispositivo. Il protocollo MIDI, al riguardo, standardizza solo il messaggio di inizio (SYSEX) e di terminazione (END OF EXCLUSIVE, EOX) della sequenza. Sebbene questa coppia delimiti messaggi della famiglia *System Exclusive*, i messaggi stessi appartengono convenzionalmente alla famiglia *System Common*.

La Figura 2.1 mostra l'articolazione dei messaggi MIDI nelle due categorie e nelle cinque famiglie, che ora verranno descritte nel dettaglio.

2.4 I messaggi *Channel Voice*

I messaggi di canale relativi alle voci, definiti nelle specifiche come *Channel Voice*, sono complessivamente 7 e sono dedicati alla performance musicale in senso stretto. Per quanto detto sopra sulla struttura dei byte di stato, la famiglia esaurisce ben 7 delle 8 combinazioni rese disponibili dai 3 bit riservati all'identificazione del messaggio. La Tabella 2.2 presentata alla fine di questa sezione funge da specchietto riassuntivo.

Nel seguito verranno illustrati nel dettaglio tutti i messaggi *Channel Voice*; fa eccezione CONTROL CHANGE, le cui peculiarità richiedono una trattazione apposita che avrà luogo nel Paragrafo 2.5.

[2] Come si vedrà nel seguito, nel protocollo non tutte le 16 combinazioni che si rendono così disponibili sono associate a messaggi MIDI: alcune risultano indefinite.

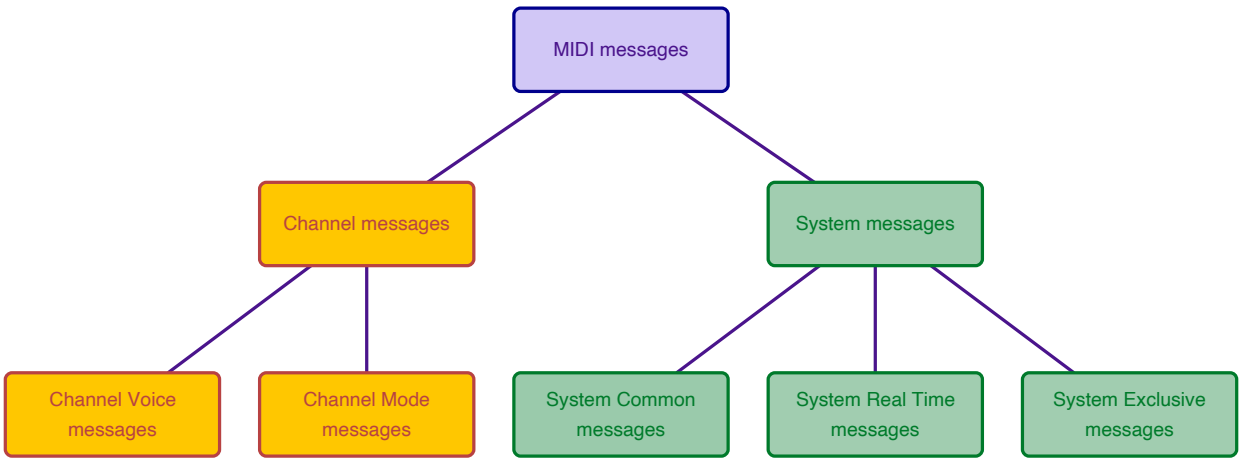


Figura 2.1. Le due categorie e le cinque famiglie di messaggi MIDI.

2.4.1 Il messaggio NOTE-ON

Il messaggio NOTE-ON viene generato dalla pressione di un tasto su un *controller* a tastiera, e corrisponde dal punto di vista logico all'avvio di una nota. Questa definizione, liberamente tradotta a partire dalle specifiche, risente dell'originale legame con gli strumenti a tastiera, ma può essere facilmente estesa ad altre categorie di *controller* in cui la nota viene emessa, ad esempio, per percussione, insufflazione, pizzicamento, e via dicendo.

In un messaggio NOTE-ON, il byte di stato si presenta nella forma $\texttt{1001nnnn}_2$ o $\texttt{9n}_{16}$, ove:

- il bit in posizione più significativa è posto a `1` per identificare l'ottetto come byte di stato;
- i 3 bit successivi, posti a $\texttt{001}_2$, contengono l'identificativo del messaggio NOTE-ON;
- i 4 bit meno significativi sono riservati all'informazione di canale.

Un messaggio NOTE-ON si compone poi di due byte di dati, riservati, rispettivamente, all'informazione di altezza della nota, detta *pitch*, e di intensità, detta *velocity*. Quest'ultimo termine richiama la rapidità nel gesto di pressione del tasto da parte dello strumentista, e nelle specifiche si raccomanda di interpretarne il valore su una scala logaritmica. Per entrambi i byte di dati, il bit più significativo è posto a `0`, pertanto ai valori di pitch e velocity sono riservati i restanti 7 bit. Questo consente di disporre di $2^7 = 128$ valori o livelli nell'intervallo [0, 127].

Riguardo alla rappresentazione dei pitch e alla loro corrispondente frequenza, l'argomento verrà trattato più diffusamente nel Paragrafo 2.4.2. In merito all'intensità, essa può assumere un livello tra 0, che indica assenza di suono, e 127, che nelle specifiche non viene associato a un valore assoluto. Il valore medio, che potrebbe corrispondere in notazione musicale a ***mf***, è 64, ossia 01000000_2 o 40_{16}.

Ad esempio, il messaggio

```
10010000 01000101 01000000
```

codifica un messaggio di NOTE-ON sul Canale 1, che in MIDI si rappresenta come 0000_2. Il pitch è pari a 69, che corrisponde – come si vedrà nel prossimo paragrafo – a un *la* dell'ottava centrale, mentre l'intensità è pari a 64, che rappresenta un livello medio.

Normalmente la ricezione di NOTE-ON da parte di un sintetizzatore avvia la riproduzione della nota con l'altezza e l'intensità specificate nel messaggio, usando il timbro attualmente associato al canale su cui giunge il messaggio. Per altre tipologie di dispositivi, però, la ricezione di NOTE-ON potrebbe corrispondere a eventi differenti, quali l'accensione di luci o la movimentazione di macchine sulla scena.

Si rammenta che, essendo il protocollo MIDI seriale, anche nell'ipotesi di accensione perfettamente simultanea di più note (accordo), i rispettivi messaggi di NOTE-ON verrebbero immessi in serie sul canale trasmissivo.

La Figura 2.2 fornisce una rappresentazione grafica del messaggio di NOTE-ON. Le cifre binarie indicate da `nnnn` nel byte di stato sono riservate al canale, le cifre `kkkkkkk` nel primo byte di dati si riferiscono al pitch, le cifre `vvvvvvv` nel secondo byte di dati rappresentano la velocity.

Figura 2.2. Rappresentazione grafica del messaggio NOTE-ON.

2.4.2 Altezza delle note in MIDI

Lo standard MIDI identifica le altezze delle note del sistema temperato attraverso 128 valori interi, compresi tra 0 e 127. A ogni incremento (o decremento) del pitch corrisponde una nota che dista un semitono dalla precedente in senso ascendente (o discendente).

Preso a riferimento il pitch 60, che corrisponde al *do* dell'ottava centrale (261,63 Hz), al pitch 59 corrisponde il *si* naturale dell'ottava inferiore, e al pitch 61 il *do*♯ dell'ottava stessa. Il pitch del *la* centrale (440 Hz) corrisponde al valore 69.

Applicando questa logica si ricava che tutti i pitch multipli di 12 rappresentano un *do* naturale su differenti ottave. Specificamente, il pitch 0 è attribuito al *do* naturale di frequenza 8,18 Hz, che si trova al di fuori del campo uditivo umano. La tastiera del pianoforte corrisponde ai pitch che vanno da 21 a 108.

Nella documentazione tecnica dei dispositivi MIDI l'altezza delle note viene comunemente indicata in cosiddetta notazione scientifica, sistema proposto dall'*Acoustical Society of America* nel 1939. L'altezza viene completamente identificata da una stringa composta dal nome della nota secondo la terminologia anglosassone (lettere da A a G, cui corrispondono nella solmisazione i nomi da *la* a *sol*), seguita dall'eventuale accidente che ne rappresenta lo stato di alterazione e da un numero arabo che ne indica l'ottava.

La numerazione delle ottave può sollevare dubbi, non essendo univoca in ambito musicale. Tra i pianisti, ad esempio, è comune definire –1 l'ottava più bassa (incompleta) di una tastiera a 88 tasti, il che porta l'ottava centrale a essere la terza ottava. Un altro approccio, del tutto naturale in ambito MIDI, consisterebbe nel dividere 60 (il primo pitch dell'ottava centrale) per 12 (il numero di semitoni nell'ottava), portando così il pitch 0 a essere la prima nota dell'ottava 0, e di conseguenza identificando l'ottava centrale come ottava 5. Ciononostante, in ambito MIDI si registra convergenza nel numerare 4 l'ottava centrale; in tal modo i pitch estremi di una tastiera a 88 tasti sono chiamati A0 (pitch 21) e C8 (pitch 108), e C4 è il *do* dell'ottava centrale. Questa numerazione delle ottave riflette il forte legame del MIDI con gli strumenti a tastiera: C4, infatti, risulta il quarto *do* in una tastiera standard a 88 tasti. Si sottolinea, comunque, che le specifiche MIDI riportano l'associazione tra pitch e frequenze (ad esempio, il pitch 69 corrisponde alla frequenza di 440 Hz), senza specificare che tale altezza vada chiamata A4.

L'intonazione del *la naturale* dell'ottava centrale, detto *la corista*, è stata storicamente oggetto di revisioni e aggiustamenti. Ad esempio, nell'800 in molte nazioni era applicato il cosiddetto standard francese, che poneva il *la* corista alla frequenza di 435 Hz. Una figura chiave nella ridefinizione dell'intonazione del *la* fu Johann Heinrich Scheibler, musicologo autodidatta e inventore nel 1834 del tonometro (in tedesco *Tonmesser*), un dispositivo per misurare le altezze. Egli propose di prendere a riferimento la frequenza di 440 Hz, e la sua idea, immediatamente approvata dal *Gesellschaft Deutscher Naturforscher und Ärzte*, venne poi adottata dall'industria

musicale statunitense nel corso del '900. Nel 1936 l'*American Standards Association* diede indicazioni in tal senso, mentre una vera e propria standardizzazione della frequenza del *la* corista venne effettuata dall'*International Organization for Standardization* (ISO) nel 1955 e riconfermata nel 1975: si tratta dello standard ISO 16[3]. Nel 1971 l'intonazione del *la* corista a 440 Hz fu giuridicamente riconosciuta da una delegazione nominata dal Consiglio d'Europa a cui anche l'Italia si è adeguata.

Una volta fissato il riferimento, è possibile calcolare per via matematica le frequenze di ciascuna nota. Il primo passaggio consiste nel definire l'intervallo di ottava tra due note n_1 e n_2. Dette rispettivamente f_1 e f_2 le frequenze delle due note, esse formano un intervallo di ottava (ascendente) se $f_2/f_1 = 2$. In altri termini, in un intervallo di ottava, la nota superiore ha frequenza doppia rispetto a quella inferiore.

A questo punto è possibile suddividere l'ottava in intervalli più piccoli. Nella teoria occidentale, l'intervallo più stretto è quello di semitono, definito in modo tale che l'ottava risulti suddivisa in 12 semitoni. Il *sistema temperato equabile*, formalizzato alla fine del '600, prevede che i semitoni siano uguali, ossia che l'intervallo di semitono ascendente a partire da una data nota e l'intervallo di semitono discendente a partire dalla nota posta un tono sopra la prima diano origine alla stessa frequenza. Questo significa, ad esempio, che nel sistema temperato la frequenza del *do*♯ e quella del *re*♭ della stessa ottava coincidono.

La definizione di semitono "uguale" può trarre in inganno. Considerando due generiche note n_1 e n_2 che formano un intervallo di semitono ascendente, quello che risulta costante *non* è la differenza $f_2 - f_1$, bensì il rapporto

$$f_2/f_1 = \sqrt[12]{2} \approx 1,0594. \tag{2.1}$$

Quindi, per calcolare la frequenza di una nota posta un semitono sopra una data frequenza, quest'ultima va moltiplicata per $\sqrt[12]{2}$. Peraltro è facile verificare che tale operazione, compiuta 12 volte ossia quanti sono i semitoni all'interno di un'ottava, porta esattamente a raddoppiare la frequenza iniziale:

$$f_2 = (((f_1 \cdot \sqrt[12]{2}) \cdot \sqrt[12]{2}) \cdot \ldots) \cdot \sqrt[12]{2} = (\sqrt[12]{2})^{12} \cdot f_1 = 2f_1 \tag{2.2}$$

A questo punto, applicando iterativamente la formula, è possibile calcolare la frequenza di qualsiasi nota che disti s semitoni, in senso ascendente o discendente, rispetto a un riferimento dato. Fissata per convenzione la frequenza del *la* corista a 440 Hz, la frequenza da calcolare sarà:

$$f_s = (\sqrt[12]{2})^s \cdot f_1 \tag{2.3}$$

Ad esempio, si voglia ottenere la frequenza del *do* centrale (C4) conoscendo quella del *la* corista (A4): lo scostamento in numero di semitoni risulta $s = -9$. Se ne ricava $f_{C4} = (\sqrt[12]{2})^{-9} \cdot f_{A4} = 2^{-9/12} \cdot 440 \approx 261,63$ Hz.

La Tabella 2.1 mostra la corrispondenza tra le frequenze così determinate e i pitch MIDI, limitandosi all'estensione standard di una tastiera a 88 tasti. Si rammenti, però, che i pitch MIDI presentano ulteriori 21 valori nel registro grave, numerati da 0 a 20, e 19 in quello acuto, numerati da 109 a 127. I sintetizzatori digitali applicano normalmente la corrispondenza tra pitch e frequenze mostrata nella tabella, ma, come si vedrà nel seguito, è possibile modificare tali associazioni attraverso opportuni messaggi MIDI.

Esistono formule rapide che, basandosi sulla teoria sopra esposta, consentono di calcolare il pitch MIDI m di qualsiasi nota partendo dalla frequenza in hertz f_m e viceversa. La formula diretta è:

$$m = 12 \cdot \log_2(f_m/440) + 69 \tag{2.4}$$

Al fine di comprendere tale equazione, si osservi innanzitutto che $\log_2(f_m/440)$ dà il numero di ottave (valore non necessariamente intero né positivo) che separa le note di frequenza f_m e 440 Hz.

[3] https://www.iso.org/standard/3601.html

Tabella 2.1. Nome delle note, corrispettiva frequenza e pitch MIDI.

Nota	Frequenza (Hz)	Lunghezza d'onda (cm)	Pitch MIDI	Tastiera a 88 tasti
A0	27,50	1254,55	21	
A#0/B♭0	29,14	1184,13	22	
B0	30,87	1117,67	23	
C1	32,70	1054,94	24	
C#1/D♭1	34,65	995,73	25	
D1	36,71	939,85	26	
D#1/E♭1	38,89	887,10	27	
E1	41,20	837,31	28	
F1	43,65	790,31	29	
F#1/G♭1	46,25	745,96	30	
G1	49,00	704,09	31	
G#1/A♭1	51,91	664,57	32	
A1	55,00	627,27	33	
A#1/B♭1	58,27	592,07	34	
B1	61,74	558,84	35	
C2	65,41	527,47	36	
C#2/D♭2	69,30	497,87	37	
D2	73,42	469,92	38	
D#2/E♭2	77,78	443,55	39	
E2	82,41	418,65	40	
F2	87,31	395,16	41	
F#2/G♭2	92,50	372,98	42	
G2	98,00	352,04	43	
G#2/A♭2	103,83	332,29	44	
A2	110,00	313,64	45	
A#2/B♭2	116,54	296,03	46	
B2	123,47	279,42	47	
C3	130,81	263,74	48	
C#3/D♭3	138,59	248,93	49	
D3	146,83	234,96	50	
D#3/E♭3	155,56	221,77	51	
E3	164,81	209,33	52	
F3	174,61	197,58	53	
F#3/G♭3	185,00	186,49	54	
G3	196,00	176,02	55	
G#3/A♭3	207,65	166,14	56	
A3	220,00	156,82	57	
A#3/B♭3	233,08	148,02	58	
B3	246,94	139,71	59	
C4	261,63	131,87	60	
C#4/D♭4	277,18	124,47	61	
D4	293,66	117,48	62	
D#4/E♭4	311,13	110,89	63	
E4	329,63	104,66	64	
F4	349,23	98,79	65	
F#4/G♭4	369,99	93,24	66	
G4	392,00	88,01	67	
G#4/A♭4	415,30	83,07	68	
A4	440,00	78,41	69	
A#4/B♭4	466,16	74,01	70	
B4	493,88	69,85	71	
C5	523,25	65,93	72	
C#5/D♭5	554,37	62,23	73	
D5	587,33	58,74	74	
D#5/E♭5	622,25	55,44	75	
E5	659,25	52,33	76	
F5	698,46	49,39	77	
F#5/G♭5	739,99	46,62	78	
G5	783,99	44,01	79	
G#5/A♭5	830,61	41,54	80	
A5	880,00	39,20	81	
A#5/B♭5	932,33	37,00	82	
B5	987,77	34,93	83	
C6	1046,50	32,97	84	
C#6/D♭6	1108,73	31,12	85	
D6	1174,66	29,37	86	
D#6/E♭6	1244,51	27,72	87	
E6	1318,51	26,17	88	
F6	1396,91	24,70	89	
F#6/G♭6	1479,98	23,31	90	
G6	1567,98	22,00	91	
G#6/A♭6	1661,22	20,77	92	
A6	1760,00	19,60	93	
A#6/B♭6	1864,66	18,50	94	
B6	1975,53	17,46	95	
C7	2093,00	16,48	96	
C#7/D♭7	2217,46	15,56	97	
D7	2349,32	14,69	98	
D#7/E♭7	2489,02	13,86	99	
E7	2637,02	13,08	100	
F7	2793,83	12,35	101	
F#7/G♭7	2959,96	11,66	102	
G7	3135,96	11,00	103	
G#7/A♭7	3322,44	10,38	104	
A7	3520,00	9,80	105	
A#7/B♭7	3729,31	9,25	106	
B7	3951,07	8,73	107	
C8	4186,01	8,24	108	

Ad esempio, per $f_m = 880$ Hz il valore è 2; per $f_m = 622,25$ Hz, che approssima la frequenza del *re*♭ dell'ottava superiore (si veda la Tabella 2.1), il valore è circa 0,5, ossia metà ottava, il che è coerente con la distanza di 6 semitoni rispetto al riferimento. Tornando all'equazione, la distanza in numero di ottave va moltiplicata per i semitoni che compongono un'ottava, e il risultato va sommato al pitch del riferimento scelto. In questo caso, trattandosi del *la* corista, il pitch risulta 69.

La formula inversa è:

$$f_m = 2^{(m-69)/12} \cdot 440 \tag{2.5}$$

Rifacendosi alla Formula 2.3, si osservi che $2^{(m-69)/12} = (\sqrt[12]{2})^{m-69}$ rappresenta il fattore moltiplicativo da applicare alla frequenza di base per ottenere la frequenza della seconda nota. Avendo fissato il riferimento al valore 69, risulta $s = m - 69$ ossia $m = s + 69$. Ad esempio, è facile verificare che una distanza in semitoni $s = -9$ rispetto al *la* corista individui il pitch $m = 60$ proprio del *do* centrale.

Come ultima osservazione, si noti che l'uso dei pitch MIDI per denotare le altezze comporta perdita di informazione rispetto alla notazione musicale. Limitandosi all'uso di alterazioni doppie per le note naturali, ogni pitch MIDI corrisponde a tre potenziali notazioni diverse in partitura; fa eccezione la coppia *sol*♯/*la*♭, per la quale è impossibile determinare un'ulteriore scrittura basata sull'alterazione doppia di una nota naturale. L'argomento non assume particolare rilievo nella comunicazione tra dispositivi MIDI, il cui funzionamento prescinde dagli aspetti simbolici di partitura, ma verrà ripreso nel Paragrafo 5.1 affrontando gli Standard MIDI Files, che invece costituiscono un formato per rappresentare informazione musicale.

2.4.3 Il messaggio NOTE-OFF

Il messaggio NOTE-OFF viene generato al rilascio di un tasto su un *controller* a tastiera. In analogia con NOTE-ON, anche in questo caso la metafora della tastiera deve essere generalizzata, e il gesto che determina lo spegnimento della nota dipende dall'interfaccia del *controller*.

In un messaggio NOTE-OFF, il byte di stato si presenta nella forma $\texttt{1000nnnn}_2$ o $\texttt{8n}_{16}$, ove:

- il bit in posizione più significativa è posto a `1` per identificare l'ottetto come byte di stato;
- i 3 bit successivi, posti a $\texttt{000}_2$, contengono l'identificativo del messaggio NOTE-OFF;
- i 4 bit meno significativi sono riservati all'informazione di canale.

Anche il messaggio NOTE-OFF presenta due byte di dati, riservati, rispettivamente, all'altezza (*pitch*) della nota da spegnere e alla rapidità nel rilascio del tasto (*velocity*). In entrambi i casi il bit più significativo è posto a `0` e i restanti 7 bit codificano l'informazione numerica. Molti *controller* non sono in grado di generare l'informazione di velocity per NOTE-OFF: in questo caso, le specifiche suggeriscono di utilizzare nel secondo byte di dati il valore di default 64, ossia 01000000_2 e 40_{16}.

Il messaggio NOTE-OFF è pensato per terminare l'esecuzione di una nota precedentemente accesa da un corrispettivo NOTE-ON, ossia un messaggio che presenti lo stesso canale e lo stesso pitch. Eventuali altre note contemporaneamente accese nel canale non vengono influenzate. Essendo poi NOTE-OFF un messaggio di canale, esso non può interagire con messaggi su altri canali; nello specifico, se l'altezza indicata nel pitch di un NOTE-OFF sul canale N_x risultasse correntemente accesa (anche) da un NOTE-ON sul canale N_y con $x \neq y$, lo stato di accensione sul canale N_y non verrebbe influenzato.

Un'alternativa al messaggio NOTE-OFF è un messaggio NOTE-ON avente uguale canale, stesso pitch e velocity pari a 0. Questa variante non permette di specificare una velocità di rilascio, e dunque è indicata solo per quei *controller* che non implementano tale funzionalità. La sostituzione di NOTE-OFF con NOTE-ON, prevista nelle specifiche MIDI, potrebbe sorprendere per via della limitata disponibilità di combinazioni per identificare i messaggi MIDI; con i vincoli dati dall'informazione di canale, infatti, si possono realizzare solo 8 combinazioni, e, di queste, 2

risultano parzialmente sovrapponibili. Tuttavia, l'utilizzo di NOTE-ON con velocity nulla in luogo di NOTE-OFF può comportare benefici notevoli in caso di adozione della tecnica *running status* (Paragrafo 2.11), e quindi è un'alternativa suggerita dalle specifiche stesse in nome di una maggiore efficienza nell'occupazione del canale trasmissivo.

A titolo di esempio, si voglia spegnere il *la* dell'ottava centrale acceso dal NOTE-ON mostrato nel Paragrafo 2.4.1. Il messaggio MIDI corrispondente risulta:

```
10000000 01000101 01000000
```

oppure

```
10010000 01000101 00000000
```

a seconda della tecnica di codifica privilegiata dal dispositivo mittente.

Esistono casi in cui un messaggio NOTE-OFF cerca di spegnere una nota non attiva. Potrebbe trattarsi del risultato di una delle seguenti condizioni di errore: il *controller* non ha generato il messaggio NOTE-ON ma solo il corrispondente NOTE-OFF; il *controller* ha erroneamente generato un NOTE-OFF in eccesso; il *controller* ha correttamente generato i due messaggi, ma il NOTE-ON è andato perso o è stato alterato durante la trasmissione sul canale. La situazione potrebbe però verificarsi anche in condizione di funzionamento normale. Ad esempio, se più *controller* collegati in *daisy chain* operano sullo stesso canale, le azioni dei due performer interagiscono. Dette A e B le due sorgenti e fissato un pitch comune, la sequenza NOTE-ON$_A$ – NOTE-ON$_B$ – NOTE-OFF$_B$ – NOTE-OFF$_A$ porta allo spegnimento della nota alla ricezione del primo dei NOTE-OFF. Peraltro, data la struttura di questi messaggi, sarebbe impossibile per un destinatario ricondurre i singoli NOTE-ON e NOTE-OFF alle rispettive sorgenti. Nel caso giunga a un sintetizzatore un NOTE-OFF che cerca di spegnere una nota non accesa, il modulo sonoro gestisce la situazione semplicemente ignorando il messaggio di NOTE-OFF.

Più critico sarebbe lo smarrimento di un messaggio NOTE-OFF, che implicherebbe la prosecuzione per un tempo indeterminato del suono avviato da un NOTE-ON. Per prevenire questo problema MIDI supporta la tecnica dell'*active sensing*, descritta nel Paragrafo 2.9.3.

La Figura 2.3 fornisce una rappresentazione grafica del messaggio di NOTE-OFF. Le cifre binarie indicate da `nnnn` nel byte di stato sono riservate al canale, le cifre `kkkkkkk` nel primo byte di dati si riferiscono al pitch, le cifre `vvvvvvv` nel secondo byte di dati rappresentano la velocity. La Figura 2.4 mostra la variante basata su NOTE-ON con velocity 0.

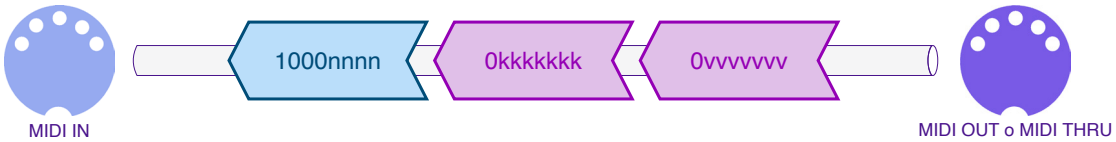


Figura 2.3. Rappresentazione grafica del messaggio NOTE-OFF.

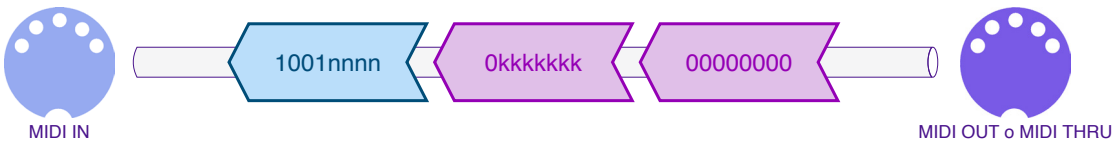


Figura 2.4. Rappresentazione grafica della variante di NOTE-OFF per mezzo di NOTE-ON con velocity nulla.

2.4.4 Il messaggio PROGRAM CHANGE

Il messaggio PROGRAM CHANGE serve a stabilire l'associazione tra timbro e canale attraverso l'invio del numero di programma (detto *program number* o *patch number*). Tutti gli eventi sonori su un dato canale condividono lo stesso numero di programma; per avere una performance multi-timbrica è necessario inviare NOTE-ON/NOTE-OFF su canali differenti (ed eventualmente stabilire ciascuna *patch* attraverso PROGRAM CHANGE).

In un messaggio PROGRAM CHANGE, il byte di stato si presenta nella forma $1100nnnn_2$ o Cn_{16}, ove:

- il bit in posizione più significativa è posto a 1 per identificare l'ottetto come byte di stato;
- i 3 bit successivi, posti a 100_2, contengono l'identificativo del messaggio PROGRAM CHANGE;
- i 4 bit meno significativi sono riservati all'informazione di canale.

Il messaggio reca un unico byte di dati, che serve a identificare uno dei timbri presenti nel sintetizzatore. La sintassi limiterebbe il numero massimo di *patch* a 128, ma nel seguito si vedrà come sia possibile organizzare i timbri in banchi da 128 *patch* ciascuno, estendendo così le potenzialità dei sintetizzatori multi-timbrici.

Si osservi che il messaggio veicola solo il valore identificativo del timbro, senza alcuna informazione sui parametri di sintesi. Dunque nelle performance MIDI si fa affidamento sulle timbriche messe a disposizione dai sintetizzatori, senza poter avere un controllo più raffinato sulla qualità del suono.

Va anche precisato che, in assenza di uno standard di riferimento, ogni sintetizzatore potrebbe associare un timbro diverso e non noto a priori allo stesso valore numerico, rendendo difficilmente prevedibile il comportamento di PROGRAM CHANGE. Questo problema, particolarmente rilevante nelle performance MIDI, è stato risolto attraverso l'adozione dello standard General MIDI, di cui si parlerà nel Capitolo 4.

La Figura 2.5 fornisce una rappresentazione grafica del messaggio di PROGRAM CHANGE. Le cifre binarie indicate da nnnn nel byte di stato sono riservate al canale, quelle indicate da ppppppp nel byte di dati codificano il nuovo numero di programma.

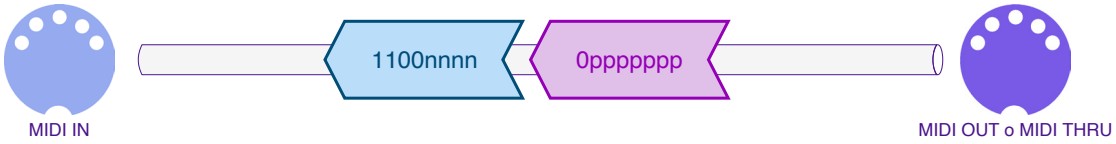


Figura 2.5. Rappresentazione grafica del messaggio PROGRAM CHANGE.

2.4.5 Esempio di NOTE-ON, NOTE-OFF e PROGRAM CHANGE

Si supponga che un performer voglia eseguire su un *controller* MIDI la sequenza melodico-ritmica mostrata nella Figura 2.6, impostando lo strumento per inviare i messaggi sul Canale 5 e volendo associare agli eventi il timbro caratterizzato dal numero di *patch* 12. Il *controller* si trova all'interno di una catena minimale costituita dal dispositivo stesso (mittente) e da un sintetizzatore (destinatario), collegati attraverso un cavo MIDI posto tra il connettore MIDI OUT del primo e il connettore MIDI IN del secondo. Si ipotizzi che le apparecchiature siano già accese, collegate tra loro e pronte a scambiarsi messaggi.

La sequenza di operazioni che l'utente dovrà compiere sul *controller* include:

1. la selezione del canale su cui il *controller* invierà i messaggi;
2. la scelta del timbro associato al canale, che deve essere comunicata al modulo sonoro;

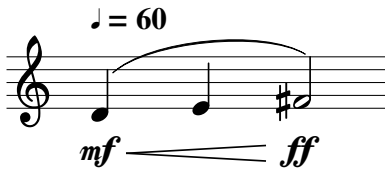


Figura 2.6. La linea melodico-ritmica dell'esempio.

3. la pressione e il rilascio dei tasti (se si tratta di *controller* a tastiera).

Una prima osservazione riguarda il punto 1: la selezione del canale per il *controller* trova riscontro nella costruzione dei messaggi MIDI successivi, ma di per sé non genera un messaggio da immettere sul canale trasmissivo. In altre parole, non è necessario informare il sintetizzatore del fatto che il *controller* ora operi sul Canale 2; il sintetizzatore semplicemente si troverà a gestire messaggi in arrivo sul Canale 2, indipendentemente da quale dispositivo a monte nella catena li stia trasmettendo.

L'assegnazione di un timbro al canale dal punto di vista logico deve essere successiva, in quanto trova riscontro in un messaggio PROGRAM CHANGE, che è di canale. Se le operazioni fossero invocate in ordine inverso e il *controller* inizialmente stesse operando, ad esempio, sul canale Canale 1, il sintetizzatore assocerebbe timbro 12 al Canale 1, per poi ricevere messaggi di performance sul Canale 2, producendo suoni con uno strumento presumibilmente diverso. Talvolta i *controller*, alla variazione del canale, inviano automaticamente l'impostazione del timbro, rendendo in pratica possibile anche l'inversione delle operazioni 1 e 2, con il risultato di generare comunque un PROGRAM CHANGE aggiuntivo. Si tratta peraltro di un comportamento non stabilito dalle specifiche MIDI, dipendendo piuttosto dalle scelte implementative del produttore, per cui non è logicamente corretto farvi affidamento.

Riguardo all'accensione e allo spegnimento delle note, si osservi che i messaggi corrispondenti di NOTE-ON e NOTE-OFF risultano temporizzati in modo "naturale", in quanto vengono immessi sul cavo MIDI, trasmessi e interpretati dal sintetizzatore nel minor tempo possibile rispetto al gesto estemporaneo del performer. Nei messaggi MIDI non vi è una rappresentazione della nozione di tempo, quale potrebbe essere un *timestamp* (assoluto) o un Δt (relativo all'occorrenza dell'evento precedente).

In una condizione ideale, in cui l'esecutore sia estremamente preciso, il canale trasmissivo sgombro, le latenze (intrinseche in qualsiasi circuito elettrico) annullate o equiparate, il monitoraggio dei messaggi MIDI trasmessi sulla catena darebbe il risultato riportato nella Figura 2.7. Per un performer meno preciso si potrebbe verificare la situazione mostrata nella Figura 2.8, in cui si rileva una parziale sovrapposizione tra due note consecutive legate e un piccolo scollamento temporale tra lo spegnimento di una nota e l'accensione di quella successiva. In entrambe le tabelle l'asse temporale è orientato verticalmente e il tempo scorre dall'alto verso il basso.

Riguardo ai valori si osservi che il Canale 2 viene correttamente codificato come 0001_2 (il cui valore è 1) e che le dinamiche vengono poste in corrispondenza con valori di velocity crescenti ma arbitrari. Infine, nella colonna di sinistra, messaggi posti immediatamente uno sotto l'altro implicano una distanza temporale piccola a piacere, ma non nulla.

2.4.6 Il messaggio CHANNEL PRESSURE (Aftertouch)

Il messaggio CHANNEL PRESSURE è una prima forma di *aftertouch* (letteralmente: dopo-tocco). Nella metafora dei *controller* a tastiera, un dispositivo in grado di generare tale messaggio deve essere sensibile alla variazione nella pressione esercitata sul tasto dopo l'accensione della nota. Questo discorso può essere facilmente adattato ad altri modelli di *controller*, prendendo in considerazione, ad esempio, l'andamento dell'insufflazione in uno strumento a fiato, o dello sfregamento delle corde in uno strumento ad arco.

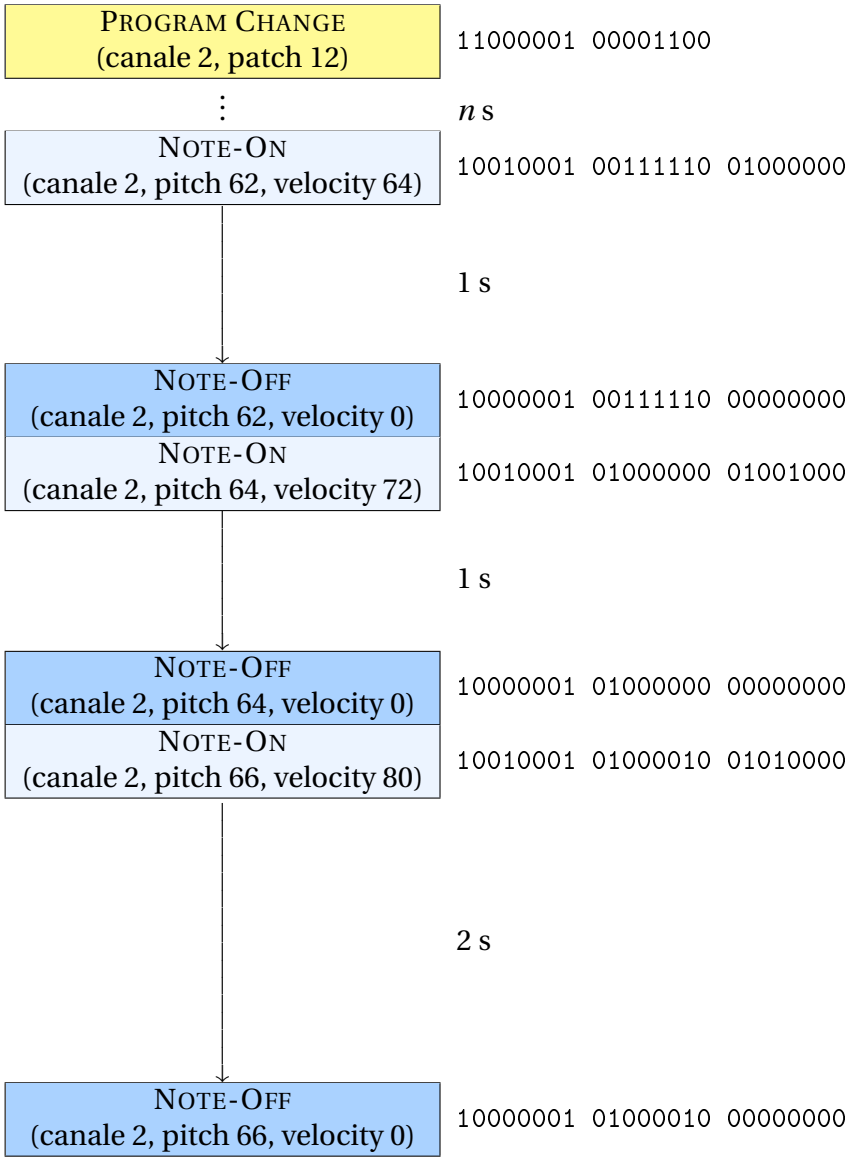


Figura 2.7. Messaggi MIDI generati da un *performer* particolarmente accurato.

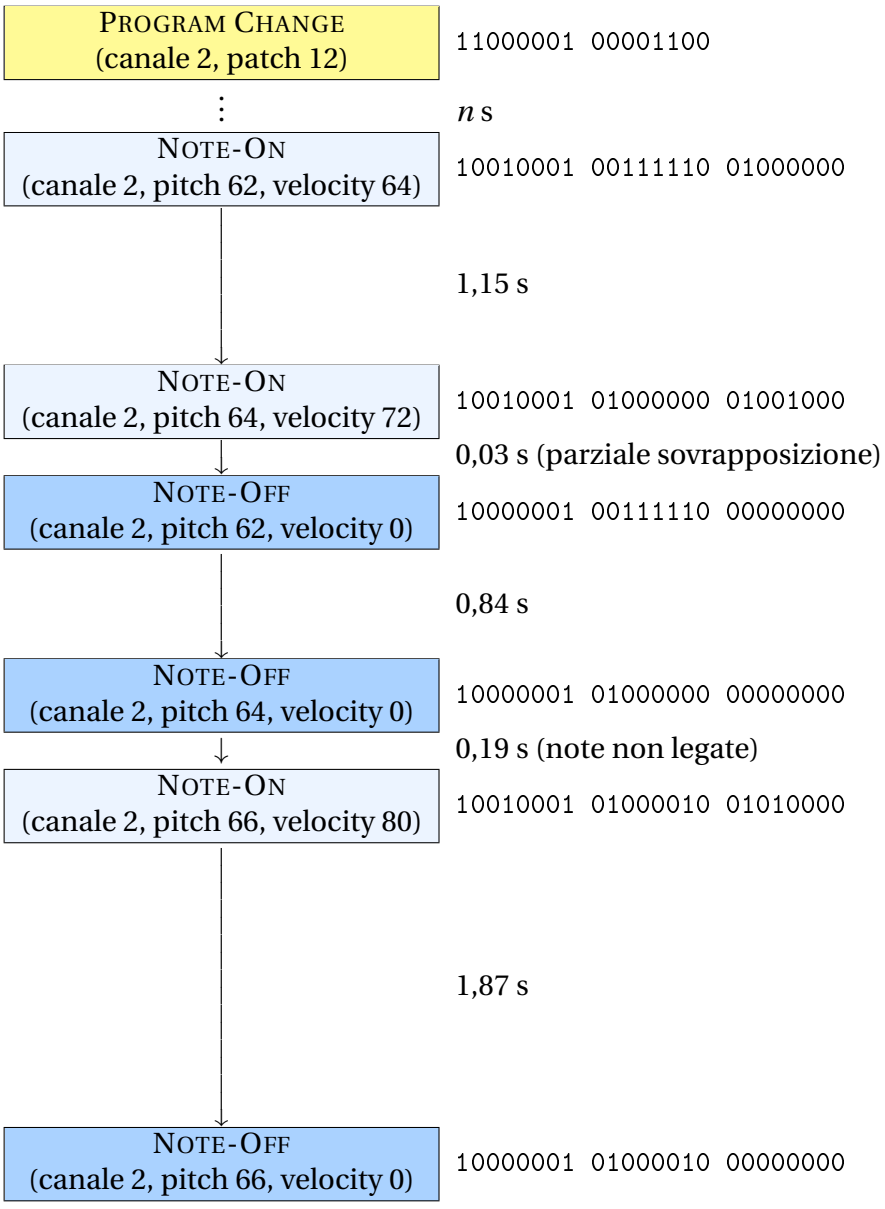


Figura 2.8. Messaggi MIDI generati da un *performer* impreciso.

In un messaggio CHANNEL PRESSURE, il byte di stato si presenta nella forma $1101nnnn_2$ o Dn_{16}, ove:

- il bit in posizione più significativa è posto a 1 per identificare l'ottetto come byte di stato;
- i 3 bit successivi, posti a 101_2, contengono l'identificativo del messaggio CHANNEL PRESSURE;
- i 4 bit meno significativi sono riservati all'informazione di canale.

Il messaggio CHANNEL PRESSURE presenta un solo byte di dati che specifica, attraverso un valore a 7 bit, la variazione di pressione calcolata facendo una media su tutti i tasti premuti, o, per meglio dire, rilevata da tutti i sensori attivati. Il byte di dati tiene dunque conto della pressione aggiuntiva esercitata dallo strumentista dopo l'accensione delle note, che è un valore necessariamente positivo. Un valore pari a 0 viene usato per indicare il ritorno alla situazione iniziale; non avrebbe senso inviare frequenti messaggi CHANNEL PRESSURE sul canale trasmissivo in assenza di variazione. Le specifiche MIDI non fissano in termini assoluti quale sia la variazione di pressione cui far corrispondere il valore massimo, lasciando questa scelta all'implementazione da parte del produttore. Peraltro molti *controller* non sono in grado di rilevare le variazioni di pressione, e dunque non generano messaggi di questo tipo. Analogamente, la documentazione MIDI non specifica come i sintetizzatori debbano rispondere a variazioni di pressione: scelte comuni dei produttori includono l'alterazione dell'intensità sonora, la variazione nella brillantezza del timbro e la modulazione dell'effetto di vibrato.

L'effetto complessivo di CHANNEL PRESSURE viene influenzato dai valori di pressione, potenzialmente differenti tra loro, rilevati sui singoli tasti. Va comunque ricordato che esso si riverbera in modo identico su tutte le note attive sul canale. Ad esempio, si supponga di suonare una triade, e che la pressione sui tasti influenzi nel sintetizzatore il *cutoff* di un filtro VCF (*Voltage Controlled Filter*). In tal modo, aumentando la pressione sui tasti, il suono diventa più brillante, mentre riducendola il suono si fa più ovattato. Una volta avviate le note che compongono la triade, l'applicazione di una maggiore pressione su uno solo dei tasti comporta che tutte le note assumano un timbro più brillante. In altre parole, tramite il messaggio CHANNEL PRESSURE non è possibile disaccoppiare l'effetto tra note che appartengono allo stesso canale; questa potenzialità è legata all'altro messaggio di aftertouch previsto dalle specifiche MIDI, ossia POLYPHONIC KEY PRESSURE (Paragrafo 2.4.7).

CHANNEL PRESSURE è il primo messaggio MIDI a evidenziare in modo significativo i limiti nel *bitrate* del protocollo. Infatti, ipotizzando continue variazioni nella pressione dei tasti da parte dello strumentista, potrebbero venire immessi sul canale trasmissivo un gran numero di messaggi CHANNEL PRESSURE, a maggior ragione se questo avviene su più canali (si pensi a più tastiere, ciascuna impostata su un canale diverso, in configurazione *daisy chain*). Le specifiche non precisano con quale frequenza, o in risposta a quali eventi, un *controller* in grado di generare messaggi CHANNEL PRESSURE li debba immettere sul canale trasmissivo: si tratta di scelte implementative lasciate al produttore. Il problema è noto con il nome di *MIDI choke*, ossia intasamento, ed è la condizione che si verifica quando una sorgente di messaggi MIDI cerca di inviare dati a una frequenza che supera la capacità trasmissiva del canale. Ciò avviene di rado per messaggi NOTE-ON e NOTE-OFF, in quanto la pur limitata banda è in grado di trasmetterne circa 1000 al secondo, e in numero ancora maggiore con l'attivazione del *running status* (Paragrafo 2.11). Messaggi che invece richiedono un frequente invio di valori, come nel caso di variazioni continue nella pressione, risultano problematici.

La Figura 2.9 fornisce una rappresentazione grafica del messaggio di CHANNEL PRESSURE. Le cifre binarie indicate da nnnn nel byte di stato sono riservate al canale, quelle indicate da ppppppp nel byte di dati rappresentano il valore di pressione.

2.4.7 Il messaggio POLYPHONIC KEY PRESSURE (Aftertouch)

Il messaggio POLYPHONIC KEY PRESSURE implementa una seconda forma di *aftertouch* finalizzata a rilevare la variazione della pressione esercitata sui tasti dopo l'accensione della nota. Si tratta

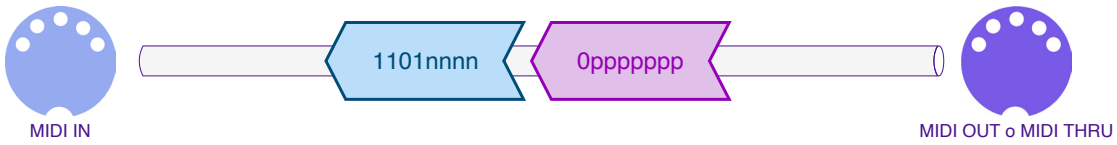


Figura 2.9. Rappresentazione grafica del messaggio CHANNEL PRESSURE.

ancora una volta di un messaggio di canale, ma – a differenza del messaggio CHANNEL PRESSURE descritto nel paragrafo precedente – in questo caso i valori di pressione rilevati sui differenti tasti vengono mantenuti distinti.

In un messaggio POLYPHONIC KEY PRESSURE, il byte di stato si presenta nella forma $\texttt{1010nnnn}_2$ o $\texttt{An}_{16}$, ove:

- il bit in posizione più significativa è posto a `1` per identificare l'ottetto come byte di stato;
- i 3 bit successivi, posti a $\texttt{010}_2$, contengono l'identificativo del messaggio POLYPHONIC KEY PRESSURE;
- i 4 bit meno significativi sono riservati all'informazione di canale.

Il messaggio include due byte di dati, riservati, rispettivamente, all'altezza (*pitch*) della nota su cui viene rilevata la variazione di pressione e all'entità della variazione stessa. In entrambi i casi il bit più significativo è posto a `0` e i restanti 7 bit codificano l'informazione numerica.

Riprendendo l'esempio della triade introdotto nel Paragrafo 2.4.6, dopo l'accensione delle tre note la variazione nella pressione di una sola nota influenza la risposta del sintetizzatore per quell'unica nota, mentre le altre non vengono alterate.

Il messaggio in oggetto, consentendo di inviare con continuità informazioni sulle singole note, rende ancora più evidente il problema del *MIDI choke* introdotto nel precedente paragrafo.

La Figura 2.10 fornisce una rappresentazione grafica del messaggio di POLYPHONIC KEY PRESSURE. Le cifre binarie indicate da `nnnn` nel byte di stato sono riservate al canale, quelle indicate da `kkkkkkk` nel primo byte di dati codificano il pitch cui applicare il valore di pressione quantificato da `vvvvvvv` nel secondo byte di dati.

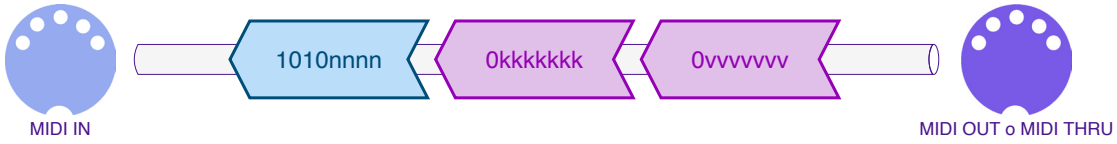


Figura 2.10. Rappresentazione grafica del messaggio POLYPHONIC KEY PRESSURE.

2.4.8 Esempi di CHANNEL PRESSURE e POLYPHONIC KEY PRESSURE

In questo paragrafo vengono presentati due esempi riguardanti il messaggio CHANNEL PRESSURE e il messaggio POLYPHONIC KEY PRESSURE.

Al fine di semplificare il contesto, verrà presa in considerazione una catena MIDI minimale, costituita da un *controller* senza aree di *split*, che dunque invia tutti i messaggi di performance su un unico canale, collegato a un sintetizzatore in ascolto sul canale stesso (Figura 1.8). Gli esempi rimarrebbero validi anche in *setup* più articolati, ad esempio con molteplici *controller* che inviano messaggi di performance su uno stesso canale o su più canali MIDI. In questo caso va ricordato che entrambi i tipi di *aftertouch* sono messaggi di canale, quindi si applicano al solo canale indicato; in aggiunta, nel caso di POLYPHONIC KEY PRESSURE, si applicano alle sole note accese su tale canale e

con un certo pitch. D'altro canto, i messaggi si applicano a tutte le note accese che soddisfano tali requisiti, indipendentemente da quali dispositivi generano gli *aftertouch* e quali le note.

Un'altra premessa necessaria è che il *controller* abbia sensori in grado di rilevare variazioni di pressione e generare di conseguenza gli *aftertouch*, cosa non banale soprattutto per il POLYPHONIC KEY PRESSURE. Ad esempio, tra il 1992 e il 2010 non è stato immesso sul mercato alcun *controller* MIDI in grado di inviare questo tipo di messaggio, solitamente delegato alle potenzialità dei *sequencer*.

Esempio di CHANNEL PRESSURE

Si consideri la situazione rappresentata, dal punto di vista della notazione musicale, nella Figura 2.11: a un istante di tempo t_0 viene suonata una nota, che dura complessivamente 8 s. A 2 s dall'accensione si verifica un aumento progressivo della pressione esercitata sul tasto, che arriva al suo massimo dopo altri 2 s. Nei successivi 2 s la pressione via via diminuisce, per poi stabilizzarsi su quella iniziale fino allo spegnimento della nota, dopo ulteriori 2 s. A livello di messaggi inviati dal *controller* si verifica una situazione simile a quella mostrata nella Figura 2.12.

Se anziché di una nota si fosse trattato di un bicordo, sarebbe stata sufficiente una variazione di pressione su uno solo dei due tasti per inviare analoghi messaggi CHANNEL PRESSURE. Se le variazioni di pressione risultassero differenti sugli n tasti, starebbe alle scelte implementative dei produttori determinare il comportamento da inviare attraverso CHANNEL PRESSURE. Una possibilità sarebbe quella di basarsi sulla variazione di pressione più marcata; un'altra possibilità consisterebbe nel calcolare una media tra le variazioni di pressione.

Esempio di POLYPHONIC KEY PRESSURE

Si consideri la situazione rappresentata, dal punto di vista della notazione musicale, nella Figura 2.13. A un istante di tempo t_0 viene suonato un bicordo, che dura complessivamente 8 s. Per la nota più acuta si verifica la situazione descritta all'esempio precedente; per quella più grave, fin dall'istante di accensione si registra un progressivo aumento della pressione sul tasto, che raggiunge il suo culmine dopo 4 s per poi annullarsi completamente nei restanti 4 s. La sequenza di messaggi MIDI inviati dal *controller* potrebbe essere quella riportata nella Figura 2.14.

In questo scenario la presenza simultanea di altre note accese sul canale non influisce, né tali note vengono influenzate dall'azione sui tasti in questione. Per completezza va detto che un altro *controller* collegato a monte potrebbe stare agendo sullo stesso canale del *controller* corrente e potrebbe avere già acceso una o più delle note in oggetto: in tal caso, le azioni dei due *controller* si accavallerebbero.

2.4.9 Il messaggio PITCH BEND CHANGE

Il messaggio PITCH BEND CHANGE viene usato per alterare l'intonazione della nota (pitch), effettuando interventi di microintonazione con una precisione potenzialmente ben superiore ai cent,[4]. Trattandosi di un messaggio di canale e non consentendo di selezionare singoli pitch attraverso i byte di dati, il suo effetto si riverbera su tutte le note correntemente accese nel canale.

A livello di interfaccia utente, questo messaggio viene spesso associato a una *pitch bend wheel*, un controllo a forma di rotella dotato di richiamo in posizione centrale al fine di annullarne automaticamente l'effetto.

In un messaggio PITCH BEND CHANGE, il byte di stato si presenta nella forma $\texttt{1110nnnn}_2$ o $\texttt{En}_{16}$, ove:

- il bit in posizione più significativa è posto a `1` per identificare l'ottetto come byte di stato;

[4] In musica, il centesimo di semitono, abbreviato in cent, è un'unità di misura degli intervalli musicali. Il cent corrispondente alla centesima parte di un semitono equabile, ovvero alla milleduecentesima parte di un'ottava.

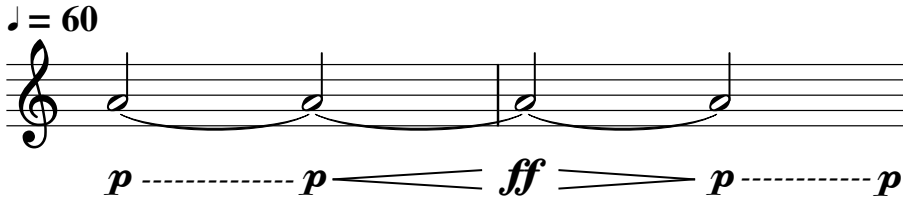


Figura 2.11. Esempio di variazione della pressione sul tasto.

NOTE-ON (Canale 2, pitch 69, velocity 40)	10010001 01000101 00101000	t = 0 s
		circa 2 s
CHANNEL PRESSURE (Canale 2, pressione +2)	11010001 00000010	t = (2+ϵ) s
CHANNEL PRESSURE (Canale 2, pressione +4)	11010001 00000100	
CHANNEL PRESSURE (Canale 2, pressione +6)	11010001 00000110	
⋮		
CHANNEL PRESSURE (Canale 2, pressione +62)	11010001 00111110	
CHANNEL PRESSURE (Canale 2, pressione +64)	11010001 01000000	t = 4 s
CHANNEL PRESSURE (Canale 2, pressione +62)	11010001 00111110	
⋮		
CHANNEL PRESSURE (Canale 2, pressione +4)	11010001 00000100	
CHANNEL PRESSURE (Canale 2, pressione +2)	11010001 00000010	
CHANNEL PRESSURE (Canale 2, pressione 0)	11010001 00000010	t = 6 s
		2 s
NOTE-OFF (Canale 2, pitch 69, velocity 0)	10000001 01000101 00000000	t = 8 s

Figura 2.12. Messaggi MIDI generati da una variazione di pressione sul canale.

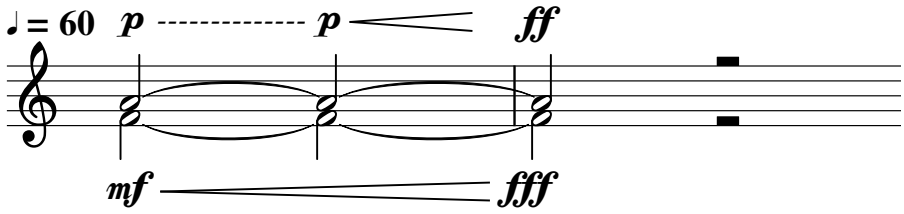


Figura 2.13. Esempio di variazione della pressione su più tasti.

NOTE-ON (Canale 2, pitch 69, velocity 40)	10010001 01000101 00101000	t = 0 s
NOTE-ON (Canale 2, pitch 65, velocity 64)	10010001 01000001 01000000	
POLY KEY PRESSURE (Canale 2, pitch 65, pressione +2)	10100001 01000001 00000010	
POLY KEY PRESSURE (Canale 2, pitch 65, pressione +4)	10100001 01000001 00000100	
⋮		
POLY KEY PRESSURE (Canale 2, pitch 65, pressione +49)	10100001 01000001 00110001	t = 2 s
POLY KEY PRESSURE (Canale 2, pitch 69, pressione +2)	10100001 01000101 00000010	
POLY KEY PRESSURE (Canale 2, pitch 69, pressione +4)	10100001 01000101 00000100	
POLY KEY PRESSURE (Canale 2, pitch 65, pressione +50)	10100001 01000001 00110010	
POLY KEY PRESSURE (Canale 2, pitch 69, pressione +6)	10100001 01000101 00000110	
⋮		
POLY KEY PRESSURE (Canale 2, pitch 69, pressione +62)	10100001 01000101 00111110	
POLY KEY PRESSURE (Canale 2, pitch 69, pressione +64)	10100001 01000101 01000000	t = 4 s
POLY KEY PRESSURE (Canale 2, pitch 65, pressione +98)	10100001 01000001 01100010	
↓		circa 2 s
NOTE-OFF (Canale 2, pitch 69, velocity 0)	10000001 01000101 00000000	t = 6 s
NOTE-OFF (Canale 2, pitch 65, velocity 0)	10000001 01000001 00000000	

Figura 2.14. Messaggi MIDI generati da una variazione di pressione gestita polifonicamente. Ai pitch sono stati assegnati colori diversi.

- i 3 bit successivi, posti a 110_2, contengono l'identificativo del messaggio PITCH BEND CHANGE;
- i 4 bit meno significativi sono riservati all'informazione di canale.

Il messaggio prevede l'invio di due byte di dati, i cui valori a 7 bit rappresentano, rispettivamente, il byte meno significativo (LSB, *least significant byte*) e il più significativo (MSB, *most significant byte*) di un unico valore a 14 bit. Questo consente di suddividere un intervallo in $2^{14} = 16384$ livelli, cui corrispondono 8192 livelli inferiori, il livello di intonazione prefissato e 8191 livelli superiori.

A titolo esemplificativo, si riportano alcuni esempi di messaggi PITCH BEND CHANGE. La sequenza di byte

```
1110nnnn 00000000 01000000
```

si riferisce al valore a riposo, ossia all'intonazione prefissata per i pitch del canale $nnnn_2$. Si nota che il valore codificato nei byte di stato, ricostruito considerando solo i 7 bit di *payload* e l'ordine LSB/MSB, è 10000000000000_2 = 8192. La sequenza di byte

```
1110nnnn 00000001 01000000
```

reca il valore 10000000000001_2 = 8193, il che implica un livello sopra quello predefinito (intonazione lievemente crescente). Il massimo incremento si ottiene con 14 bit tutti posti a 1, fermo restando che entrambi i byte di dati debbano comunque avere uno 0 come bit più significativo. Infine,

```
1110nnnn 01111111 00111111
```

trasporta il valore 11111111111111_2 = 8191 (lo 0 in posizione più significativa è stato omesso). Questo valore corrisponde a un livello sotto quello predefinito (intonazione lievemente calante). Il massimo decremento si ottiene con 14 bit tutti posti a 0.

L'intervallo – superiore e inferiore – da suddividere in livelli intermedi può essere specificato attraverso un apposito messaggio CONTROL CHANGE, detto *Pitch Bend Sensitivity*, illustrato nel Paragrafo 2.5. Un'impostazione comune è quella di suddividere in livelli l'intervallo di semitono, in modo che la massima variazione di *pitch bend* porti la nota a coincidere con quella immediatamente precedente o successiva. In questo caso si ha la possibilità di suddividere in 8192 livelli il passaggio tra i due pitch, dando un senso di continuità nel graduale passaggio. Anche i messaggi PITCH BEND CHANGE, come quelli di *aftertouch*, contribuiscono a intasare il canale trasmissivo.

La Figura 2.15 fornisce una rappresentazione grafica del messaggio di POLYPHONIC KEY PRESSURE. Le cifre binarie indicate da `nnnn` nel byte di stato sono riservate al canale, quelle indicate da `vvvvvvv` nel primo byte di dati e da `xxxxxxx` nel secondo rappresentano, rispettivamente, la parte meno significativa e più significativa di un valore espresso su 14 bit.

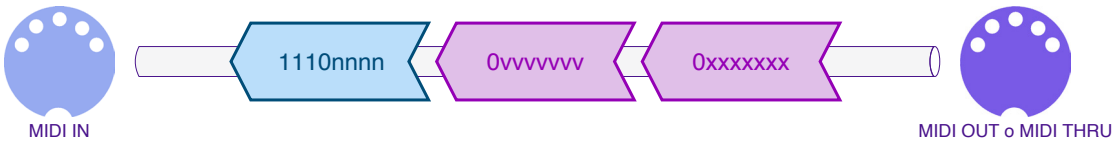


Figura 2.15. Rappresentazione grafica del messaggio PITCH BEND CHANGE.

2.5 Il messaggio CONTROL CHANGE

Il messaggio CONTROL CHANGE viene inviato quando si utilizza un controllo diverso da un tasto (sempre nella metafora pianistica) per modificare il suono di una nota. Esempi in tal senso sono i pedali (*pedal*), le rotelle (*wheel*), le levette (*lever*) e gli interruttori (*switch*).

In un messaggio CONTROL CHANGE, il byte di stato si presenta nella forma $\texttt{1011nnnn}_2$ o $\texttt{Bn}_{16}$, ove:

- il bit in posizione più significativa è posto a `1` per identificare l'ottetto come byte di stato;
- i 3 bit successivi, posti a $\texttt{011}_2$, contengono l'identificativo del messaggio CONTROL CHANGE;
- i 4 bit meno significativi sono riservati all'informazione di canale.

Il messaggio prevede l'invio di due byte di dati, i cui valori a 7 bit rappresentano, rispettivamente, il numero di *controller* (*Control Change number*, o più semplicemente *CC number*) e il relativo valore (*CC value*); il significato di quest'ultimo dipende dal primo byte di dati.

La collocazione di CONTROL CHANGE all'interno della suddivisione in famiglie di Figura 2.1 non è univoca. Infatti, CONTROL CHANGE, più che essere un messaggio, sottintende un insieme di messaggi, alcuni riferibili alle voci e altri ai modi: a seconda del *CC number*, il messaggio può appartenere alla famiglia *Channel Voice* o *Channel Mode*. In particolare, il messaggio CONTROL CHANGE afferisce:

- alla famiglia *Channel Voice* quando il primo byte di dati assume un valore nell'intervallo [0, 119], ossia $[00000000_2, 01110111_2]$ in base 2 e $[00_{16}, 77_{16}]$ in base 16;
- alla famiglia *Channel Mode* quando il primo byte di dati appartiene all'intervallo [120, 127], ossia $[01111000_2, 01111111_2]$ in base 2 e $[78_{16}, 7F_{16}]$ in base 16. Si tratta delle ultime 8 combinazioni di valori disponibili.

Si sottolinea che l'identificativo della funzione svolta viene delegato al primo byte di dati. Ci si trova di fronte a uno stratagemma adottato dalle specifiche MIDI per estendere il numero di identificativi di messaggio a disposizione. Anche se all'interno del byte di stato il messaggio CONTROL CHANGE è identificato da un codice a 3 bit univoco, è grazie ai 7 bit del primo byte di dati (*CC number*) che ne viene specificata la funzione. Il secondo byte di dati è il solo a svolgere il ruolo normalmente attribuito ai byte di dati: codificare il valore da attribuire alla funzione. All'interno della documentazione MIDI esiste una tabella apposita che elenca i codici identificativi dei messaggi CONTROL CHANGE contenuti nel primo byte di dati.

Le assegnazioni di tali codici sono avvenute di comune accordo tra MMA e JMSC, al fine di definire funzioni specifiche per le applicazioni standard in ambito musicale. Tuttavia, non si preclude ai produttori la possibilità di ridefinirli per applicazioni extra-musicali compatibili con il MIDI, come i *controller* di illuminazione o i generatori di fumo, a patto di esplicitare le nuove associazioni nel manuale utente.

La Figura 2.16 fornisce una rappresentazione grafica del messaggio di CONTROL CHANGE, ove `ccccccc` rappresenta il *CC number* e `vvvvvvv` il *CC value*.

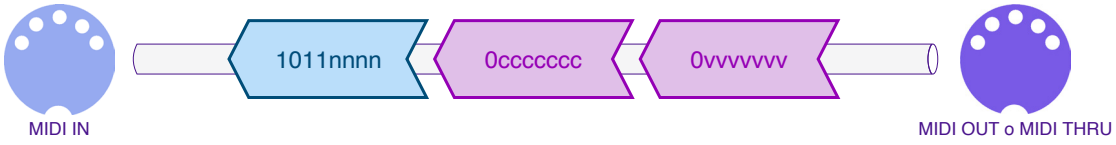


Figura 2.16. Rappresentazione grafica del messaggio CONTROL CHANGE.

La Figura 2.17 mostra invece la collocazione del messaggio all'interno delle famiglie menzionate, introducendo anche l'ulteriore ripartizione che verrà descritta nel prossimo paragrafo.

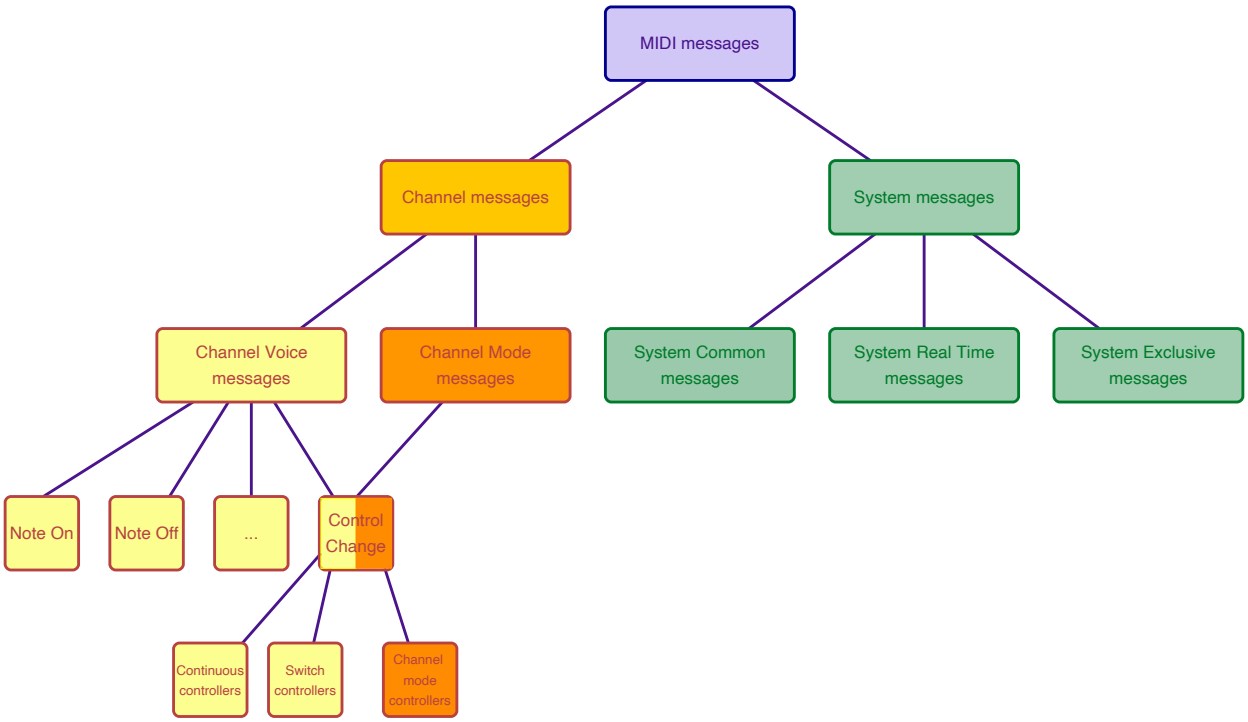


Figura 2.17. Collocazione di CONTROL CHANGE all'interno delle famiglie di messaggi MIDI.

2.5.1 Categorie di CONTROL CHANGE

Nell'insieme di messaggi CONTROL CHANGE è possibile riconoscere diverse categorie:

- ***controller* continui** (*continuous controller*), appartenenti alla famiglia *Channel Voice* e pensati per immettere sul canale trasmissivo informazioni frequenti, tenendo traccia di valori che variano gradualmente e/o con continuità. Per questa categoria, il primo byte di dati ricade nell'intervallo [0, 101], ossia [00000000_2, 01100101_2] in base 2 e [00_{16}, 65_{16}] in base 16. I *controller* continui a loro volta si suddividono in:
  - ***controller* continui ad alta risoluzione**, il cui primo byte di dati si colloca negli intervalli [0, 19] e [32, 51] (si veda il Paragrafo 2.5.2);
  - ***controller* continui a bassa risoluzione**, il cui primo byte di dati si colloca nell'intervallo [70, 95] (si veda il Paragrafo 2.5.3);
  - **numeri di parametro registrati e non registrati**, il cui primo byte di dati si colloca nell'intervallo [96, 101] (si veda il Paragrafo 2.5.4);
- ***controller* a interruttore** (*switch controller*), appartenenti alla famiglia *Channel Voice* e riferibili a comportamenti di tipo acceso/spento. Per questa categoria, il primo byte di dati assume valori nell'intervallo [64, 69] (si veda il Paragrafo 2.5.5);
- **messaggi di canale relativi al modo** (*channel mode messages*), la cui funzione è legata alla modalità di funzionamento dei dispositivi MIDI. Come precisato sopra, il valore del primo byte di dati ricade nell'intervallo [120, 127]. Quest'ultima categoria rappresenta l'intera famiglia di messaggi *Channel Mode*, pertanto verrà trattata a parte nel Paragrafo 2.7.

Alcuni intervalli di valori non sono definiti, e anche all'interno degli intervalli definiti esistono numeri di CONTROL CHANGE non assegnati. A questo argomento è dedicato il Paragrafo 2.5.6.

La categorizzazione sopra proposta rispecchia le funzionalità e le caratteristiche comuni dei blocchi di *CC number*. Nelle specifiche MIDI, invece, la numerazione dei *controller* viene

presentata in modo da coprire con continuità l'intero intervallo di valori ammissibili per il primo byte di dati:

- [0, 31] per l'MSB dei *controller* continui (*controller* ad alta risoluzione);
- [32, 63] per l'LSB dei *controller* nell'intervallo [0, 31];
- [64, 95] per i *controller* aggiuntivi a byte singolo (*controller* a bassa risoluzione);
- [96, 101] per funzioni di incremento/decremento e parametri;
- [102, 119] per *controller* non specificati a byte singolo.

Nella pratica le due classificazioni mostrate sono equivalenti, differendo principalmente per l'inclusione o meno dei valori non assegnati.

2.5.2 Controller continui ad alta risoluzione

I *controller* continui ad alta risoluzione, come si evince dal nome, devono garantire un numero adeguato di livelli per rappresentare valori a granularità fine. Il secondo byte di dati codifica valori nell'intervallo [0, 127], ove in generale 0 implica nessun effetto e 127 il massimo effetto. Esistono però alcuni casi particolari, tra cui i *controller* che presentano posizione centrale, riuniti nel gruppo dei cosiddetti *controller effect* (*Balance*, *Pan*, *Expression*) e quelli in cui il valore numerico funge da selettore (*Bank Select*). L'argomento verrà approfondito nel seguito.

Sebbene la finalità sia garantire alta risoluzione, la struttura del messaggio CONTROL CHANGE rende disponibile un unico byte di dati per esprimere valori, e questo consentirebbe solo $2^7 = 128$ livelli. Al fine di ottenere l'alta risoluzione si deve ricorrere a uno stratagemma: inviare una coppia di messaggi CONTROL CHANGE in qualche modo associati tra loro, in cui il primo messaggio veicola il valore grezzo (*coarse*), rappresentando il Most Significant Byte (MSB), e il secondo messaggio il valore raffinato (*fine*), ossia il Least Significant Byte (LSB). Alla ricezione del secondo messaggio, il destinatario può ricostruire il valore completo a 14 bit, cui corrispondono $2^{14} = 16384$ livelli distinti.

Permane il problema di come abbinare due messaggi a un'unica funzione. Infatti, poiché il byte di stato CONTROL CHANGE è identico, è compito del primo byte di dati identificare tanto la funzione quanto il ruolo di MSB o LSB per il byte che segue. A tale scopo nella tabella dei *CC number* è stata stabilita una corrispondenza tra l'intervallo [0, 31] e [32, 63], per cui il *CC number* in posizione n all'interno del primo intervallo corrisponde alla parte MSB e il *CC number* in posizione $n + 32$ alla parte LSB della stessa funzione [5].

Si consideri, ad esempio, il CONTROL CHANGE per l'effetto di pan, che determina il posizionamento dello strumento sul fronte stereo. Ad esso viene associato il *CC number* 10 per la parte MSB e il *CC number* $10 + 32 = 42$ per la parte LSB. Volendo sfruttare la granularità più fine, con $2^{14} = 16384$ possibili posizioni, il valore 00000000000000_2 (ossia 0_{16}) è associato alla posizione completamente a sinistra, 10000000000000_2 (ossia 2000_{16}) alla posizione centrale e 11111111111111_2 (ossia $3FFF_{16}$) alla posizione completamente a destra.

Questo approccio per i *controller* continui ad alta risoluzione presenta un ulteriore vantaggio. Non tutti i dispositivi atti a generare messaggi MIDI sono in grado di gestire la granularità fine, ad esempio per via di limiti nella sensoristica, e non tutti i ricevitori sono in grado di interpretarla. Lo scambio di valori MSB e LSB viene completamente sfruttato solo se mittente e destinatario sono entrambi in grado di gestire la granularità fine. Nel caso in cui il mittente sia capace di generare MSB e LSB ma il destinatario supporti solo valori grezzi, MSB e LSB verranno entrambi immessi sul canale trasmissivo, ma il destinatario prenderà in considerazione solo la parte MSB, che è

[5] Nella definizione degli intervalli di valori ammissibili per i *controller* continui ad alta risoluzione, si può notare una discrepanza rispetto a quanto riportato nel Paragrafo 2.5.1. Questo perché le specifiche assegnano funzioni all'intervallo [0, 19], marcano come indefiniti i CC number [20, 31], ma denotano l'intero intervallo [32, 63] come LSB per il corrispettivo MSB nell'intervallo [0, 31].

quella che ha maggior peso nel distinguere i valori. Nell'esempio del *pan*, i valori nell'intervallo [0, 16383] che si otterrebbero dalla valutazione di LSB e MSB vengono rimappati sull'intervallo [0, 127] derivante dalla sola lettura dell'MSB: ciò comporta perdita di precisione, ma il posizionamento sul fronte stereo non ne viene stravolto. Nel caso in cui il mittente sia impossibilitato a generare il messaggio LSB, i valori immessi sul canale trasmissivo consentiranno una minore precisione, ma verranno comunque correttamente interpretati dal destinatario. Nell'esempio del *pan*, per il posizionamento completamente a destra verrà inviato il solo messaggio

```
1011nnnn 00001010 01111111
```

il cui valore corrisponde a 127. Ammettendo che il destinatario operi ad alta risoluzione e sostituendo il valore di LSB mancante con 7 zeri, il valore ricostruito su 14 bit risulta essere 16256, molto prossimo al valore massimo di 16383.

In questa sede non sarà possibile definire tutti i *controller* continui ad alta risoluzione, compito che si demanda alle specifiche, per cui nel seguito ci si limiterà a descrivere i più significativi.

Esempi di *controller* continui ad alta risoluzione

I *controller* che rientrano in questa categoria introducono effetti che si riverberano sull'intero canale, influenzando anche le note già accese. L'entità dell'effetto è determinata dal valore del byte di dati (se ci si limita alla bassa risoluzione dell'MSB) o dei byte di dati (se si sfrutta l'alta risoluzione attraverso l'uso di MSB/LSB).

Alcuni esempi notevoli:

- *Modulation* (CC 1 e 33), progettato per effetti di modulazione la cui realizzazione dipende dal timbro e dal modello di sintesi. Normalmente è utilizzato per realizzare il vibrato su pitch, intensità e brillantezza del suono;
- *Breath Controller* (CC 2 e 34), originariamente pensato per la generazione di messaggi MIDI attraverso l'emissione di fiato, e spesso associato a messaggi aftertouch e/o di modulazione. Le specifiche prevedono valori crescenti per il secondo byte di dati all'aumentare della pressione;
- *Foot Controller* (CC 4 e 36), appositamente introdotto per valori generati su una scala continua da parte di un pedale. L'effetto dipende da come è programmato il dispositivo ricevente, e potrebbe coincidere con *Modulation* o *Breath Controller*. Si osservi che pedali di cui si vuole tracciare il solo stato premuto/rilasciato vengono invece gestiti attraverso i *controller* a interruttore (Paragrafo 2.5.5);
- *Portamento Time* (CC 5 e 37), progettato per controllare la frequenza di calcolo dei valori intermedi nell'interpolare le frequenze di due note suonate consecutivamente sullo stesso canale. L'uso di 0 come valore del *controller* implica l'annullamento dell'effetto;
- *Volume* (CC 7 e 39), pensato per definire il volume globale del canale, cui si somma l'informazione di velocity delle singole note.

Balance, Pan, Expression

Si tratta di messaggi CONTROL CHANGE che rientrano nel gruppo di controllo degli effetti (*controller effect*). Essi presentano una posizione centrale, codificata tramite il valore centrale del byte di dati, o tramite il valore centrale ricostruito a partire dalla coppia di byte di dati in un contesto a granularità fine. In particolare:

- *Balance* (CC 8 e 40) determina il bilanciamento nel volume tra due sorgenti sonore distinte. Un esempio è offerto dai campioni stereo associati a un dato timbro. Considerando il solo MSB, il valore 0 implica sorgente sinistra a pieno volume, 64 (ossia 40_{16}) volume bilanciato, e 127 (ossia $7F_{16}$) sorgente destra pieno volume;

- *Pan* (CC 10 e 42) governa invece il posizionamento di una singola sorgente sonora sul fronte stereo. In questo caso, 0 implica posizionamento tutto a sinistra, 64 (ossia 40_{16}) al centro, e 127 (ossia $7F_{16}$) tutto a destra;

- *Expression* (CC 11 e 43) è una forma di accentuazione del volume in termini percentuali rispetto a quanto specificato sul canale dai CC 7 e 39 descritti nel precedente paragrafo.

Bank Select

Bank Select è un *controller* utilizzato per caricare un banco di suoni all'interno di un sintetizzatore. Come spiegato nel Paragrafo 2.4.4, il messaggio PROGRAM CHANGE permette di assegnare a un canale uno su 128 timbri disponibili, ma questo limite rappresenterebbe una limitazione molto forte alle potenzialità multitimbriche di un modulo sonoro. Per questo motivo viene introdotto il concetto di **banco di suoni** (*sound bank*), che racchiude un numero massimo di 128 timbri selezionabili tramite PROGRAM CHANGE. Un sintetizzatore può disporre di più banchi di suoni, e il banco corrente viene caricato tramite il messaggio *Bank Select*.

A questo tipo di CONTROL CHANGE vengono assegnati i *CC number* 0 (ossia 00_{16}) e 32 (ossia 20_{16}), con funzioni, rispettivamente, di MSB e LSB. Questo consente di disporre di un massimo di 16384 banchi, ciascuno dotato di 128 timbri.

L'operazione di selezione del banco richiede di far seguire un messaggio PROGRAM CHANGE a *Bank Select MSB* o alla coppia *Bank Select MSB* e *LSB*. L'introduzione sul canale trasmissivo di messaggi intermedi potrebbe, infatti, causare interpretazioni errate della funzione. Una volta selezionato il banco, possono essere inviati ulteriori messaggi di PROGRAM CHANGE per associare le *patch* in esso incluse ai canali.

In MIDI, i banchi vengono convenzionalmente numerati a partire da 1. Quindi, ad esempio:

- l'invio di MSB = 00_{16} e LSB = 00_{16} comporta la selezione del banco 1;
- l'invio di MSB = 00_{16} e LSB = $7F_{16}$ opera la selezione del banco 128;
- l'invio di MSB = 01_{16} e LSB = 00_{16} effettua la selezione del banco 129;
- l'invio di MSB = $7F_{16}$ e LSB = $7F_{16}$, infine, attua la selezione del banco 16384.

Mentre nei normali *controller* ad alta risoluzione i valori del secondo byte di dati determinano l'intensità dell'effetto (ad es., il volume del canale), e nei cosiddetti *controller effect* la posizione rispetto a un riferimento centrale (ad es., il *pan* sul canale sinistro o destro), per *Bank Select* si tratta di identificativi numerici che operano la selezione del banco.

2.5.3 Controller continui a bassa risoluzione

I *controller* continui a bassa risoluzione sono pensati per inviare frequenti aggiornamenti dei corrispettivi valori, ma con granularità non raffinata. Risulta dunque adeguato allo scopo il secondo byte di dati di un singolo messaggio. Anche in questo caso, 0 rappresenta il limite inferiore dell'intervallo (effetto del *controller* nullo) e 127 quello superiore (massima entità dell'effetto). Nelle specifiche, i corrispettivi *CC number* vengono associati a funzioni piuttosto generiche, quali *Sound Controller 1-10, General Purpose Controller 5-8, Effect 1-5 Depth*, e via dicendo.

2.5.4 Numeri di parametro registrati e non registrati

Nell'evidente impossibilità di supportare tutte le funzioni di controllo di tipo musicale ed extra-musicale, e soprattutto di preconizzare eventuali usi futuri, le specifiche MIDI 1.0 hanno introdotto un meccanismo standard per impostare parametri di *controller* originariamente non previsti.

In particolare, CC 98 e CC 99 sono riservati ai cosiddetti *Non-Registered Parameter Number* (*NRPN*), rispettivamente alla parte LSB e MSB del valore; CC 100 e CC 101 sono invece associati ai *Registered Parameter Number* (*RPN*), rispettivamente alla parte LSB e MSB del valore.

La differenza tra *RPN* e *NRPN* è che i numeri di parametro registrati esiste un elenco concordato tra MMA e JMSC e riportato all'interno dei documenti di specifica. Al contrario, i numeri di parametro non registrati possono essere assegnati secondo necessità dai singoli produttori. Poiché ogni dispositivo può definire un particolare *NRPN*, è possibile che macchine diverse interpretino lo stesso codice in due modi differenti, creando situazioni di ambiguità o conflitto.

È necessario un approfondimento sul valore associato ai messaggi *NRPN* e *RPN*. Come di consueto, il valore del *controller* è rappresentato dal secondo byte di dati, ma, in questo caso, esso non è associato all'entità dell'effetto, bensì identifica un numero di parametro all'interno di un elenco. Per attribuire un valore numerico al parametro, quindi, ci si deve affidare ad altri messaggi: *Data Entry* (CC 6 e CC 38), *Data Increment* (CC 96) e *Data Decrement* (CC 97).

Data Entry, che rientra nella categoria dei *controller* ad alta risoluzione, serve a specificare (attraverso il suo secondo byte di dati) il valore numerico associato al parametro il cui numero è determinato (attraverso il loro secondo byte di dati) dai messaggi NRPN e RPN. *Data Increment* implica l'incremento di un'unità rispetto al valore corrente, impostato di default o specificato da un precedente *Data Entry*; *Data Decrement* esegue il decremento di un'unità. Agendo per definizione con passo 1, *Data Increment* e *Data Decrement* non presentano valori significativi nel secondo byte di dati; trattandosi di CONTROL CHANGE, però, questi messaggi devono rispettarne la composizione su 3 byte.

Per chiarezza, si confronti la struttura di questi messaggi con quella dei *controller* continui. Nei *controller* continui la funzione è espressa nel primo byte di dati e l'entità dell'effetto è quantificata attraverso il secondo byte di dati. Per i *controller* di tipo numero di parametro il primo byte di dati contiene il generico identificatore *RPN* o *NRPN*, e il secondo rappresenta l'esatta funzione, con riferimento a una tabella di valori stabilita da MMA e JMSC o dal singolo produttore; questo determina l'esaurimento dello spazio a disposizione in un messaggio CONTROL CHANGE, per cui l'entità dell'effetto deve essere necessariamente delegata ad altri messaggi.

Esempio: Pitch Bend Sensitivity

Un esempio chiarificatore sull'uso dei numeri di parametro riguarda la risoluzione frequenziale di *pitch bend*, ossia la cosiddetta *pitch bend sensitivity*.

Il messaggio PITCH BEND CHANGE, affrontato nel Paragrafo 2.4.9, consente un aggiustamento graduale dell'intonazione del pitch attraverso 16384 valori intermedi. Quello che le specifiche non formalizzano è l'intervallo musicale da suddividere in 16384 livelli. Allo scopo è stato introdotto il *Registered Parameter Number 0* (espresso su 14 bit come 00000000000000_2), considerato di utilità generale e dunque concordato tra produttori.

Per determinare la risoluzione frequenziale di *pitch bend*, i primi messaggi a essere inviati sono:

```
1011nnnn 01100010 00000000
1011nnnn 01100011 00000000
```

ossia una coppia di CONTROL CHANGE sul canale `nnnn` (byte di stato) che identificano, rispettivamente, la parte LSB e MSB di *Registered Parameter Number* (primo byte di dati) attribuendo a entrambi il valore 0 (secondo byte di dati).

A seguire, va codificato il valore che corrisponde all'estensione dell'intervallo da suddividere. Per questo specifico *RPN* la parte MSB di *Data Entry* rappresenta l'estensione in semitoni, e la parte LSB di *Data Entry* l'ulteriore estensione in cent, ossia in centesime parti di semitono. Dai vincoli sull'intervallo di valori si comprende facilmente che l'incremento (o il decremento) complessivo possa oscillare tra un minimo di 0 semitoni e 0 cent a un massimo di 127 semitoni e 127 cent [6].

Volendo suddividere in 16384 livelli intermedi un intervallo complessivo di un tono centrato sulla frequenza originaria, consentendo dunque una microintonazione che ha come limiti il semitono precedente e successivo, i messaggi *Data Entry* da inviare risultano:

[6] Dalla definizione di cent come centesima parte del semitono, in generale il valore in cent dovrebbe ricadere nell'intervallo [0, 99]. Un incremento di 100 cent implica infatti un incremento di semitono.

```
1011nnnn 00000110 00000001
1011nnnn 00100110 00000000
```

ossia una coppia di CONTROL CHANGE sul canale `nnnn` (byte di stato) che identificano, rispettivamente, la parte MSB e LSB di *Data Entry* (primo byte di dati), attribuendo 1 come estensione in semitoni e 0 come ulteriore estensione in cent (secondo byte di dati).

Esempio: Fine e Coarse Tuning

Altri esempi d'uso dei numeri di parametro registrati sono il *Fine Tuning* e il *Coarse Tuning*, il cui scopo è quello di controllare l'accordatura globale del canale. Gli *RPN* corrispondenti sono 1 (ossia 00_{16} 01_{16}) e 2 (ossia 00_{16} 02_{16}). Si sottolinea come, anche in questo caso, si tratti di valori registrati, e dunque concordati tra produttori.

Il *Fine Tuning* utilizza tanto MSB quanto LSB, in modo da disporre di 16384 possibili valori: tanti sono i livelli in cui viene suddiviso l'intervallo di un tono centrato sulla frequenza originaria. In altre parole, è possibile variare l'intonazione nell'intervallo [-100, +100) cent con una granularità di 100/8192 cent [7].

Il *Coarse Tuning*, invece, utilizza il solo MSB, e dunque 128 valori nell'intervallo [-64, 63] che si riferiscono a una granularità in semitoni. I valori minimi e massimi, espressi in cent, risultano dunque -6400 e +6300.

In entrambi i casi il valore trasportato da *Data Entry* è la deviazione in cent rispetto al riferimento originario. Ad esempio, considerando il *la* dell'ottava centrale (pitch 69) come riferimento, si tratta del numero di cent di distanza del nuovo La rispetto alla frequenza standard pari a 440 Hz. La frequenza dei restanti pitch viene modificata di conseguenza, in quanto *Fine Tuning* e *Coarse Tuning* sono messaggi di canale e lavorano sulla cosiddetta intonazione "master".

Ad esempio, si voglia inviare un messaggio di *Fine Tuning* con deviazione pari a 1 semitono più 1 cent in senso ascendente, ossia 51 cent. I primi messaggi sono:

```
1011nnnn 01100010 00000010
1011nnnn 01100011 00000000
```

ossia una coppia di CONTROL CHANGE sul canale `nnnn` (byte di stato) che identificano, rispettivamente, la parte LSB e MSB di *Registered Parameter Number* (primo byte di dati). Il valore 2, che nelle specifiche corrisponde a *Coarse Tuning*, viene rappresentato su 14 bit e ripartito tra i secondi byte di dati dei due messaggi.

Segue una coppia *Data Entry MSB/LSB* che deve opportunamente rappresentare l'incremento di 51 cent rispetto al riferimento. Innanzitutto, ricordando che per il *Fine Tuning* l'intervallo di 100 cent viene suddiviso in 8192 valori e calcolando la proporzione, si ottiene la corrispondenza approssimata tra 51 cent e il numero di livelli $n = 8192 \times 51/100 = 4177.92 \cong 4178$. Inoltre è necessario rammentare che l'intervallo di 16384 livelli risulta simmetrico rispetto al riferimento, quindi la deviazione di entità 0 si traduce in un valore pari a 8192, cui va aggiunto il valore precedentemente calcolato pari a 4178. In definitiva, il valore da ripartire sulla coppia MSB/LSB è 12370, che si esprime su 14 bit come 11000001010010_2. I messaggi *Data Entry* da inviare risultano pertanto:

```
1011nnnn 00000110 01100000
1011nnnn 00100110 01010010
```

MIDI consente anche di modificare in modo puntuale la frequenza delle singole note anziché l'accordatura dell'intero canale, ma allo scopo vanno utilizzati messaggi SYSTEM EXCLUSIVE, come mostrato nel Paragrafo 2.10.

[7] Come denotato dall'uso di parentesi diverse, l'intervallo è chiuso a sinistra e aperto a destra, in quanto il valore minimo rappresentabile è $(100/8192) \times -8192 = -100$, mentre il valore massimo è $(100/8192) \times 8191 = 99.99$.

2.5.5 Controller a interruttore

Nella categoria dei *controller* a interruttore (*switch controller*) rientrano dispositivi quali pedali, pulsanti e interruttori, di cui interessa solo lo stato binario premuto/rilasciato, acceso/spento, ecc. Per caratterizzarne lo stato, sarebbe sufficiente un singolo bit di dati usato come flag; essendo però fissa la struttura di CONTROL CHANGE, è necessario inviare comunque un messaggio completo con due byte di dati, di cui il secondo completamente dedicato alla codifica dell'informazione binaria. La convenzione è usare l'intervallo [0, 63] per lo stato *Off* e [64, 127] per lo stato *On*.

I *controller* a interruttore previsti dalle specifiche MIDI sono i seguenti:

- *Damper Pedal On/Off (Sustain)* (CC 64). Svolge la funzione del pedale situato più a destra nei pianoforti, detto "forte" o "risonanza" o "3 corde". L'effetto di tale pedale, quando premuto, è prolungare il suono di tutte le note già attive in precedenza o avviate prima del suo rilascio. Si tratta di un messaggio di canale, per cui interagisce con i soli messaggi di NOTE-ON e NOTE-OFF sullo stesso canale. Il concetto di nota attiva al momento della pressione del pedale significa un NOTE-ON non ancora terminato da un NOTE-OFF; se giungesse al sintetizzatore il NOTE-OFF corrispondente mentre il pedale è premuto, questo verrebbe ignorato fino al rilascio del pedale. Lo stesso comportamento si applica a note attivate da un NOTE-ON (che viene istantaneamente preso in considerazione) e disattivate da un NOTE-OFF (che viene invece temporaneamente ignorato) quando lo stato del pedale è abbassato;

- *Portamento On/Off* (CC 65). Abilita o disabilita il portamento, ossia l'effetto di passaggio graduale tra due note suonate in modo consecutivo sullo stesso canale. Per ovvie ragioni, l'effetto può essere implementato solo alla ricezione del successivo NOTE-ON;

- *Sostenuto On/Off* (CC 66). Corrisponde al cosiddetto pedale "tonale" dei pianoforti, il cui scopo è prolungare unicamente i suoni attivi prima dell'abbassamento del pedale. Quindi il pedale ritarda solo i messaggi di NOTE-OFF nello stesso canale, successivi alla sua pressione e relativi a NOTE-ON precedenti al suo abbassamento;

- *Soft Pedal On/Off* (CC 67). Corrisponde al cosiddetto pedale "una corda" dei pianoforti, il cui effetto è smorzare l'intensità del suono;

- *Legato Footswitch* (CC 68). Abilita o disabilita l'effetto di legato tra due note consecutive accese sul canale. In questo caso, valori maggiori o uguali a 64 introducono l'effetto di legato, mentre valori minori rappresentano la situazione "normale", ossia senza effetto. Quando il pedale è abbassato, lo strumento entra in modalità monofonica, quindi l'arrivo sul canale specificato di un NOTE-ON prima dello spegnimento di una nota precedente tramite NOTE-OFF ne provoca l'immediata sospensione;

- *Hold 2* (CC 69). Ha un comportamento simile a *Damper Pedal* (CC 64), ma con un effetto di smorzamento naturale dei suoni.

2.5.6 Controller non definiti

Le specifiche MIDI 1.0 sottolineano che, a causa del numero limitato di *CC number* disponibili, sarebbe impossibile assegnare un codice a ogni possibile effetto musicale e non musicale, presente e futuro. Alla luce di tale affermazione appare quanto meno strana la presenza di un gran numero di *CC number* marcati come *undefined*. In realtà questi codici sono a disposizione dell'utente per definire comportamenti specifici dei dispositivi nel *setup* MIDI, controllando, ad esempio, effetti originariamente non previsti, veicolando valori di parametri che non rientrano nelle competenze di altri *controller*, e via dicendo.

La parola *undefined* può inoltre sottendere il concetto di "non ancora definito", ossia disponibile per applicazioni future. La storia del protocollo MIDI 1.0 ha mostrato che, di fatto, non è mai emersa la necessità di definire nuovi *controller* riassegnando i *CC number* non definiti.

Svariate applicazioni hanno sfruttato, piuttosto, i *Non-Registered Parameter Number*, descritti nel Paragrafo 2.5.4.

L'elenco completo dei numeri di *controller* non associati a funzioni è: 3, 9, 14, 15, da 20 a 31, da 85 a 90 e da 102 a 119. Si tratta complessivamente di 40 *CC number*.

2.6 Riassunto dei messaggi *Channel Voice*

Prima di affrontare una nuova famiglia di messaggi MIDI, è opportuno riassumere quanto visto finora riguardo alla famiglia *Channel Voice*. La Tabella 2.2 offre un quadro sinottico dei 7 messaggi trattati.

byte di stato		1° byte	2° byte	descrizione
base 16	base 2	di dati	di dati	
8n	1000nnnn	0kkkkkkk	0vvvvvvv	NOTE-OFF kkkkkkk: pitch vvvvvvv: velocity
9n	1001nnnn	0kkkkkkk	0vvvvvvv	NOTE-ON kkkkkkk: pitch vvvvvvv 0: velocity vvvvvvv = 0: NOTE-OFF
An	1010nnnn	0kkkkkkk	0vvvvvvv	POLYPHONIC KEY PRESSURE kkkkkkk: pitch vvvvvvv: valore di pressione
Bn	1011nnnn	0ccccccc	0vvvvvvv	CONTROL CHANGE ccccccc: identificativo del CC vvvvvvv: valore del CC
Cn	1100nnnn	0ppppppp		PROGRAM CHANGE ppppppp: numero di programma
Dn	1101nnnn	0vvvvvvv		CHANNEL PRESSURE vvvvvvv: valore di pressione
En	1110nnnn	0vvvvvvv	0xxxxxxx	PITCH BEND CHANGE vvvvvvv: parte LSB del valore xxxxxxx: parte MSB del valore

Tabella 2.2. Riassunto dei messaggi *Channel Voice*.

2.7 I messaggi *Channel Mode*

I messaggi di canale relativi al modo (*Channel Mode*) determinano la modalità di funzionamento dei dispositivi MIDI, e in particolare come questi debbano gestire i successivi messaggi *Channel Voice*. I cosiddetti modi MIDI risultanti si applicano tanto ai trasmettitori (*controller*, *sequencer*, ecc.) quanto ai ricevitori (sintetizzatori).

Dal punto di vista sintattico, i messaggi della famiglia *Channel Mode* si presentano come CONTROL CHANGE con *CC number* nell'intervallo [120, 127], per un totale di 8 messaggi. L'informazione di canale serve a indirizzare le direttive sul modo di funzionamento a tutti e soli i dispositivi nel *setup* che presentano come canale base quello specificato nel byte di stato. Pertanto, la modalità di funzionamento viene determinata da messaggi di canale e coinvolge tutti e soli i dispositivi "in ascolto" su tale canale. Va però sottolineato che gli strumenti MIDI, in alcune modalità operative, possono trasmettere e ricevere messaggi *Channel Voice* anche sui restanti canali MIDI. Il discorso verrà ripreso nei prossimi paragrafi.

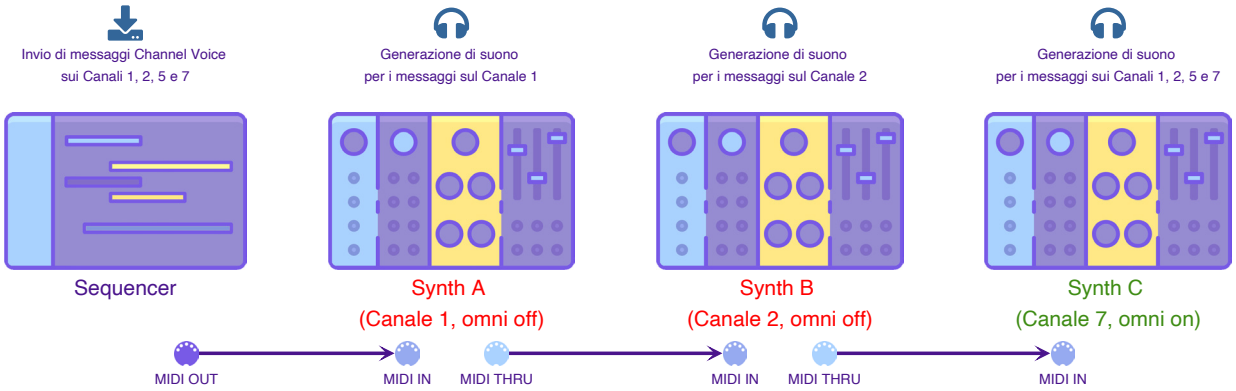


Figura 2.18. Comportamento dei sintetizzatori a seconda del valore della proprietà *omni*.

Trattandosi di casi particolari del messaggio CONTROL CHANGE, il byte di stato si presenta per l'intera famiglia nella forma $\texttt{1011nnnn}_2$ o $\texttt{Bn}_{16}$, ove:

- il bit in posizione più significativa è posto a `1` per marchiare l'ottetto come byte di stato;
- i 3 bit successivi, posti a $\texttt{011}_2$, identificano il messaggio CONTROL CHANGE, che in questo caso viene spesso definito CHANNEL MODE;
- i 4 bit meno significativi sono riservati all'informazione di canale.

2.7.1 La proprietà *omni*

Una prima proprietà che determina il modo di funzionamento dei dispositivi MIDI è ***omni***. Essa ammette i valori mutuamente esclusivi *omni on* e *omni off*.

Quando impostato a *omni off*, un dispositivo risponde a messaggi *Channel Voice* sul solo canale base[8]. Ad esempio, se il canale base è impostato a N e giunge un messaggio *Channel Voice* sul canale M con $M \neq N$, il dispositivo lo ignora; analogamente, un messaggio *Channel Mode* sul canale M non ne modifica il funzionamento.

Nello scenario *omni on*, invece, il dispositivo riceve e gestisce messaggi *Channel Voice* su tutti i 16 canali MIDI; un messaggio *Channel Mode* su un canale differente da N, però, continuerebbe a essere ignorato.

I valori alternativi per la proprietà *omni* vengono determinati dal primo byte di dati del messaggio CONTROL CHANGE, posto a 124 (01111100_2, $7C_{16}$) per *omni off* e a 125 (01111101_2, $7D_{16}$) per *omni on*. Per quanto riguarda il secondo byte di dati, la cui presenza è richiesta dalla sintassi di CONTROL CHANGE, il suo valore è indifferente ai fini del messaggio: convenzionalmente, ecco viene posto a 0.

Un caso tipico di utilizzo di *omni off* è un *setup* con più sintetizzatori in *daisy chain*, ciascuno specializzato nella produzione di un timbro e, dunque, associato ai soli eventi in arrivo su un dato canale; questo richiede un'opportuna configurazione del canale base su ciascun sintetizzatore. Si consideri l'esempio mostrato nella Figura 2.18, in cui un *sequencer* invia messaggi sui Canali 1, 2, 5 e 7 alla catena MIDI. Nel *setup* si trovano tre sintetizzatori multitimbrici, potenzialmente in grado di gestire i messaggi in arrivo su tutti i canali. Dei tre moduli sonori, però, i primi due – aventi Canale base 1 e 2, rispettivamente – sono impostati su *omni off*: questo significa che gestiranno solo messaggi *Channel Voice* in arrivo sul proprio canale base. Il terzo sintetizzatore, invece, è stato impostato su *omni on* e dunque è in grado di gestire messaggi in arrivo su tutti i canali. Per cambiare in *omni on* il funzionamento del secondo sintetizzatore, si dovrebbe inviare un messaggio così costruito:

[8] Si osservi che un dispositivo potrebbe presentare più canali base, anche se non si tratta di uno scenario consueto.

```
10110001 01111101 00000000
```

ossia un CONTROL CHANGE sul Canale 2 con primo byte di stato impostato a 125. Si osservi che questo modificherebbe il valore della proprietà *omni* di qualsiasi altro dispositivo nella catena con Canale base 2.

Lo stato consigliato dalle specifiche MIDI all'avvio dei dispositivi è *omni on*: ciò consente, ad esempio, di collegare un *controller* a un sintetizzatore e realizzare una performance senza doversi preoccupare della selezione del canale base sui dispositivi.

La Figura 2.19 fornisce una rappresentazione grafica dei messaggi CONTROL CHANGE relativi a questa proprietà, spesso definiti OMNI OFF e OMNI ON.

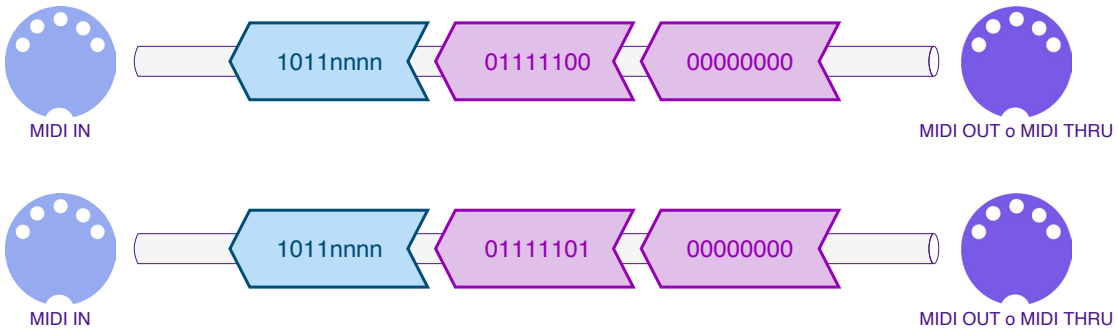


Figura 2.19. Rappresentazione grafica dei messaggi CONTROL CHANGE per *omni off* (sopra) e *omni on* (sotto).

2.7.2 La proprietà *mono*/*poly*

Una seconda proprietà che determina il modo di funzionamento dei dispositivi MIDI è *mono*/*poly*. I valori mutuamente esclusivi sono:

- *mono on*, che corrisponde a *poly off* o, più semplicemente, a *mono*;
- *mono off*, che corrisponde a *poly on* o, più semplicemente, a *poly*.

Quando impostato su *mono*, uno strumento risponde a messaggi *Channel Voice* in un contesto che, canale per canale, risulta monofonico. Questo significa che l'accensione di una nuova nota è incompatibile con l'esecuzione corrente di una precedente nota sullo stesso canale. La strategia per risolvere la situazione di incompatibilità non viene riportata nelle specifiche, ma lo scenario tipico prevede che il nuovo evento interrompa quello corrente; un'alternativa consiste nell'ignorare il nuovo evento. La proprietà *mono on* può abilitare effetti di transizione (portamento, legato, alterazione della fase di attacco della nota nei suoni campionati, ecc.), rendendo il passaggio tra note consecutive più graduale. Si tratta di scelte implementative lasciate ai produttori dei sintetizzatori e non codificate all'interno delle specifiche.

Un dispositivo in modalità *poly*, invece, è in grado di supportare la polifonia sul singolo canale. Questo si realizza per tutti i canali che lo strumento è in grado di gestire, il che dipende dalla proprietà *omni on*/*off*, come detto nel paragrafo precedente.

Il valore della proprietà viene determinato dal primo byte di dati del messaggio CONTROL CHANGE, posto, rispettivamente, a 126 (01111110_2, $7E_{16}$) per *mono* o 127 (01111111_2, $7F_{16}$) per *poly*. Per quanto riguarda il secondo byte di dati, il suo valore è indifferente ai fini di *poly*, mentre assume per *mono* un significato che verrà chiarito nel Paragrafo 2.7.3.

La modalità di funzionamento *mono* è particolarmente efficace per gli strumenti che non sono in grado di produrre polifonia, come gran parte dei fiati. Impostando, ad esempio, un flauto MIDI su *mono*, esso non potrebbe produrre più di un suono alla volta, nemmeno se l'interfaccia fisica

dello strumento MIDI lo consentisse (ad esempio, usando una tastiera per controllare il timbro del flauto). Va però precisato che, all'interno di un *setup* articolato, la questione presenta termini più complessi. I messaggi *Channel Mode* sono di canale, quindi inviare un messaggio CONTROL CHANGE con valore *mono on* sul canale N implica che tutti i dispositivi in ascolto sul canale base N entrino in tale modalità, e che messaggi di performance prodotti da *controller* differenti su uno stesso canale possano risultare mutuamente esclusivi.

La Figura 2.20 fornisce una rappresentazione grafica dei messaggi CONTROL CHANGE relativi a questa proprietà, spesso definiti MONO ON e MONO OFF o, in modo equivalente, POLY OFF e POLY ON.

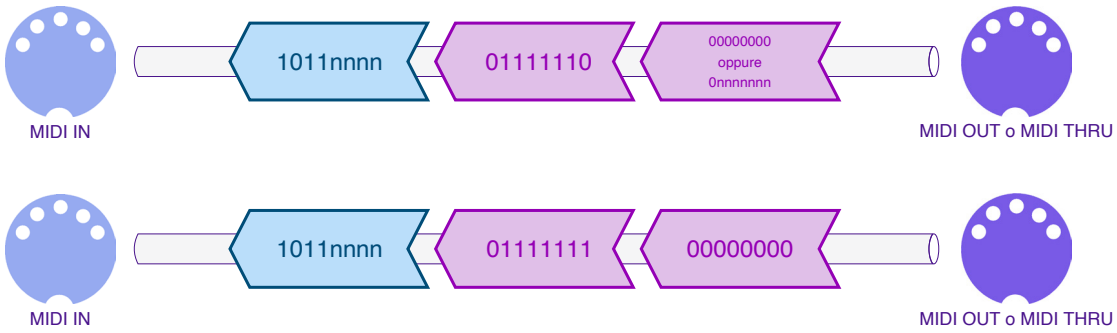


Figura 2.20. Rappresentazione grafica dei messaggi CONTROL CHANGE per *mono* (sopra) e *poly* (sotto).

2.7.3 Modi MIDI

Dalla combinazione dei valori delle due proprietà precedentemente menzionate, *omni off*/*omni on* e *mono*/*poly*, emergono quattro **modi MIDI** (Figura 2.21):

- Modo 1 (*omni on*, *poly*);
- Modo 2 (*omni on*, *mono*);
- Modo 3 (*omni off*, *poly*);
- Modo 4 (*omni off*, *mono*).

Tali modi sono mutuamente esclusivi: un ricevitore o un trasmettitore, a un dato istante, possono trovarsi in uno solo tra i modi elencati. Un dispositivo in grado tanto di generare quanto di ricevere messaggi MIDI, però, può avere trasmettitore e ricevitore impostati su modi MIDI e

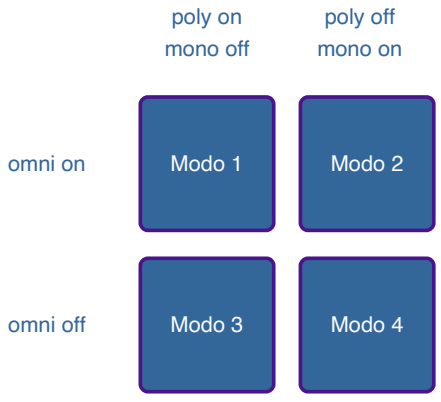


Figura 2.21. I quattro modi MIDI derivanti dalla combinazione di *omni on*/*off* e *mono*/*poly*.

canali base distinti. Va inoltre precisato che le specifiche MIDI non impongono ai produttori di implementare tutte le combinazioni.

È evidente l'importanza di impostare correttamente i vari componenti del *setup* prima di una performance, non solo determinando lo schema di collegamento più opportuno tra gli strumenti, ma anche scegliendo accuratamente il canale base e i valori delle proprietà che ne influenzano il modo di funzionamento. Poiché MIDI, almeno nella sua prima versione, è un protocollo di trasmissione unidirezionale, i dispositivi a monte non sono in grado di conoscere le impostazioni di quelli a valle. Ad esempio, in assenza di pre-impostazioni manuali opportune, è possibile che i primi inviino messaggi in modo polifonico a ricevitori che operano in modo monofonico, o che *controller* differenti si contendano un singolo canale risultando mutuamente esclusivi. Sebbene questo non generi problemi da un punto di vista tecnico, la performance musicale potrebbe considerevolmente risentirne.

All'accensione, i dispositivi MIDI presentano un modo operativo di default. Per specificare completamente un nuovo modo, è necessario inviare una coppia di messaggi CONTROL CHANGE: uno tra OMNI ON e OMNI OFF e uno tra MONO ON e POLY ON. Alla ricezione dei singoli messaggi gli strumenti passano al modo selezionato. Ad esempio, per un dispositivo che opera in Modo 3 (*omni off, poly*), il passaggio al Modo 2 (*omni on, mono*) non è istantaneo, in quanto i messaggi OMNI ON e MONO ON non possono essere inviati simultaneamente; a seconda del loro ordine di ricezione, si attraverserà temporaneamente il Modo 1 o il Modo 4.

Il mancato supporto di un determinato modo da parte di un dispositivo MIDI si traduce nella sua incapacità di gestire alcuni messaggi. Nell'esempio sopra riportato di uno strumento MIDI inizialmente in Modo 3, la corretta gestione di un messaggio OMNI ON comporta il passaggio al Modo 1, e l'incapacità di gestire il successivo MONO ON determina il mancato supporto del Modo 2, lasciando il dispositivo in Modo 1.

Il modo MIDI è spesso selezionabile anche tramite appositi controlli messi a disposizione dell'utente, ad esempio sul pannello frontale. In questo caso, è necessario prestare particolare attenzione alla sovrapposizione tra le impostazioni dettate attraverso il protocollo MIDI (ad esempio inviate da un *sequencer*) e quelle inserite manualmente durante la performance.

Voci dei dispositivi MIDI

A questo punto della trattazione è opportuno chiarire il concetto di **voce** per un dispositivo MIDI. Il numero di voci rappresenta la polifonia più estesa gestibile dal dispositivo, ossia il massimo numero di note che possono risultare contemporaneamente accese, indipendentemente dal canale di trasmissione e dal timbro associato.

Ad esempio, il *controller Yamaha CP33* supporta una polifonia massima di 64 note, il che significa un massimo di 64 tasti contemporaneamente premuti; il sintetizzatore *Yamaha MU80*, dotato di due connettori MIDI IN e dunque in grado di gestire 16 + 16 canali MIDI indipendenti, supporta una polifonia a 64 note, il che implica al massimo 64 suoni attivi in un dato istante temporale.

Il concetto di polifonia a n voci (o note) è ortogonale a quello di canale: esse potrebbero concentrarsi in un unico canale, come pure risultare ripartite su più canali.

Tipicamente le note vengono assegnate alle voci in maniera dinamica, il che consente di gestire le risorse disponibili nel sistema in modo ottimale. Un esempio di allocazione statica consisterebbe nel dividere il numero totale di voci (ad esempio, $n = 64$) per il numero di canali ($c = 16$), ottenendo così la polifonia massima per canale ($n_c = 4$). Si tratterebbe, però, di una scelta poco efficiente: nell'esempio sopra riportato non sarebbe possibile eseguire per intero un accordo da 5 note sul Canale 1, nonostante tutti gli altri canali siano quiescenti e non impegnino le restanti 60 voci.

Modo 1 (*omni on, poly*)

Un dispositivo che opera in **Modo 1** gestisce i messaggi relativi alle voci di tutti i canali (*omni on*) e li assegna polifonicamente alle voci disponibili (*poly*). Lo scenario viene illustrato nella

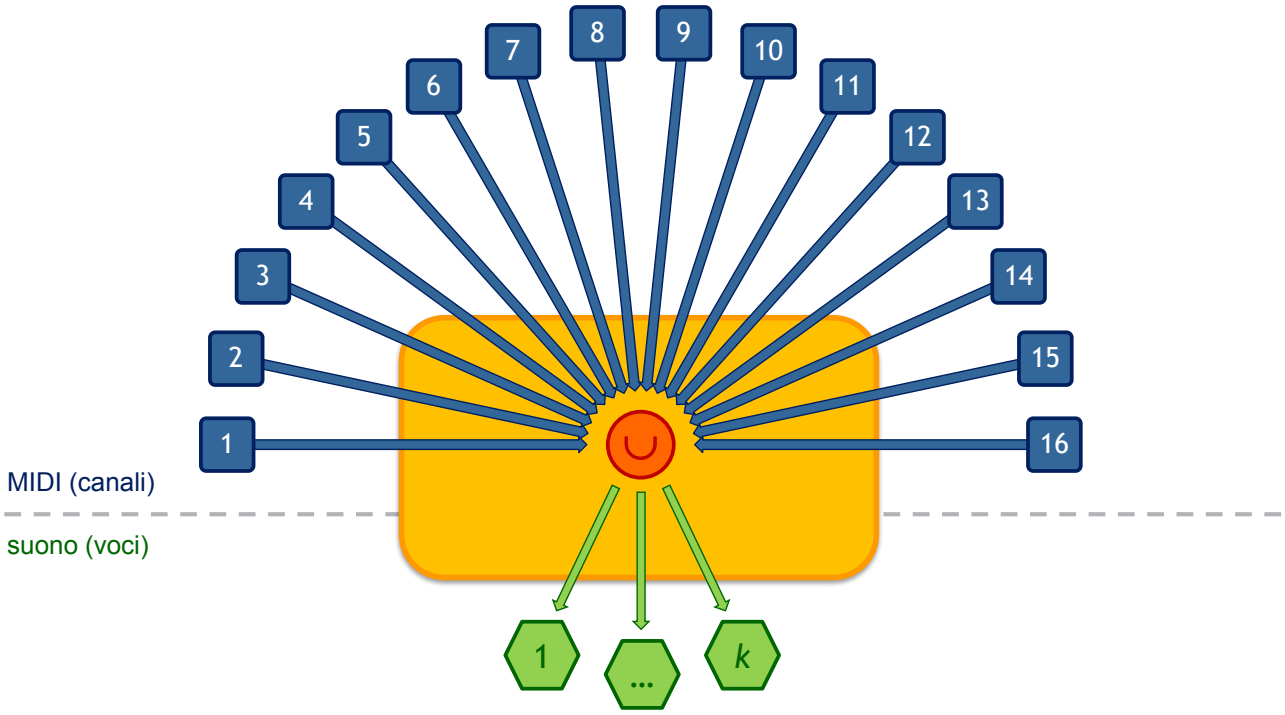


Figura 2.22. Rappresentazione grafica del Modo 1.

Figura 2.22, ove il nodo centrale rappresenta graficamente l'unione dei messaggi provenienti da differenti canali. Si rammenta che il concetto di modo si applica tanto alla componente di ricezione (ricevitore) quanto alla componente di invio (trasmettitore) dei messaggi MIDI da parte di un dispositivo.

Per un ricevitore impostato in Modo 1, i messaggi vengono recepiti da tutti i canali e sono trattati in maniera indistinta, ossia indipendente dal canale. Per questo motivo il Modo 1 trova ampia applicazione nei sintetizzatori monotimbrici [9] chiamati a rispondere ai messaggi di performance in arrivo su tutti i canali; in questo caso, infatti, si tratta di un'impostazione efficace per garantire che qualsiasi messaggio di performance in arrivo dalla catena a monte venga gestito. Viceversa, il Modo 1 non rappresenta la scelta più comune nei sintetizzatori multitimbrici, nei quali verrebbe selezionato un singolo numero di programma cui assegnare la produzione di tutti i suoni, indipendentemente dal canale. L'impostazione del *program number* viene normalmente salvata in un'apposita area di memoria del dispositivo detta *patch memory*.

Per un trasmettitore impostato in Modo 1, tutti i messaggi di performance vengono convogliati sul canale base del dispositivo in maniera polifonica. Questo vanificherebbe, ad esempio, l'effetto delle aree di *split* in una tastiera dotata di tale funzionalità: anche se le aree fossero associate a canali differenti, i messaggi confluirebbero sul solo canale base del *controller*.

Modo 2 (*omni on, mono*)

Un dispositivo che opera in **Modo 2** gestisce i messaggi relativi alle voci di tutti i canali (*omni on*) e li assegna all'unica voce resa disponibile (*mono*). Lo scenario viene illustrato nella Figura 2.23, ove il nodo centrale rappresenta graficamente l'unione dei messaggi provenienti da differenti canali.

Un ricevitore in questa modalità riceve i messaggi di performance da tutti i canali, ma li tratta in modo da controllare una singola voce (monofonia). Indipendentemente dal numero di canali MIDI

[9] Si sottolinea la differenza tra monotimbrico/multitimbrico e monofonico/polifonico. Un dispositivo monotimbrico è in grado di gestire un unico timbro, a differenza di uno strumento multitimbrico; un dispositivo monofonico è capace di gestire una sola voce, ossia l'accensione di un'unica nota alla volta, a differenza di uno strumento polifonico. Esistono, dunque, esempi di sintetizzatori simultaneamente monotimbrici e polifonici.

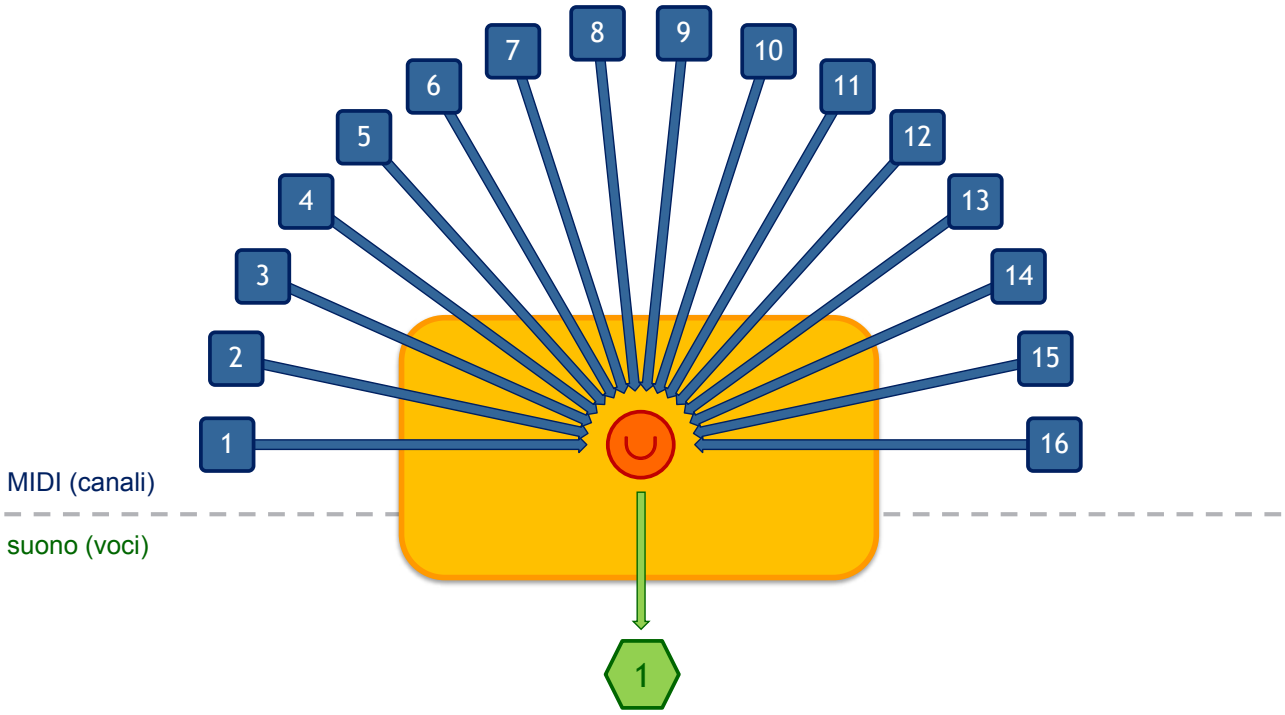


Figura 2.23. Rappresentazione grafica del Modo 2.

attivi o dalla eventuale polifonia su uno di essi, il ricevitore gestirà l'accensione di una sola nota alla volta. Nel caso dei sintetizzatori, l'implementazione tipica di questo comportamento è spegnere l'eventuale nota attiva alla ricezione di un nuovo messaggio NOTE-ON; un'alternativa meno comune consiste nell'ignorare nuove accensioni delle note finché la voce non viene "liberata" da un messaggio NOTE-OFF.

Un trasmettitore in Modo 2 invia tutti i messaggi, una volta ricondotti a una singola voce, sul proprio canale base. Anche in questo scenario, esistono varie strategie implementative per selezionare la nota da accendere o mantenere accesa in caso di conflitto. Tale problema non si pone se il *controller* cui è associato il trasmettitore è intrinsecamente pensato per operare monofonicamente, come nel caso di molti strumenti a fiato.

Il Modo 2 viene raramente implementato, soprattutto a livello di ricevitore, in quanto recepire messaggi sui 16 canali per controllare una singola voce monofonicamente appare ben poco significativo. Un esempio in tal senso, per quanto forzato, è dato da una performance in cui più *controller* in configurazione *daisy chain*, ciascuno con un proprio canale base, collaborano nel controllare uno strumento monotimbrico e monofonico: il Modo 2 garantisce che il sintetizzatore riceva messaggi relativi alla performance da tutti i canali, ma non accenda mai più di una nota alla volta. Questo stesso esempio risulta più significativo se i vari *controller* pilotano anche altri sintetizzatori, impostati su canali base diversi tra loro.

Modo 3 (*omni off, poly*) e Multi Mode

Un dispositivo che opera in **Modo 3** gestisce i messaggi relativi alle voci di un singolo canale (*omni off*) e li assegna polifonicamente alle voci disponibili (*poly*). Lo scenario viene illustrato nella Figura 2.24, ove il nodo centrale rappresenta graficamente la selezione dei messaggi provenienti da un singolo canale.

Per un ricevitore impostato in Modo 3, i messaggi relativi alle voci vengono ricevuti sul solo canale base del dispositivo e assegnati polifonicamente alle voci disponibili. Per un trasmettitore, i messaggi relativi a tutte le voci vengono inviati polifonicamente sul solo canale base.

Contrariamente a quanto si possa immaginare, il Modo 3 è il più utilizzato da parte dei sintetizzatori multitimbrici quando si desidera sfruttare la loro capacità di assegnare un timbro

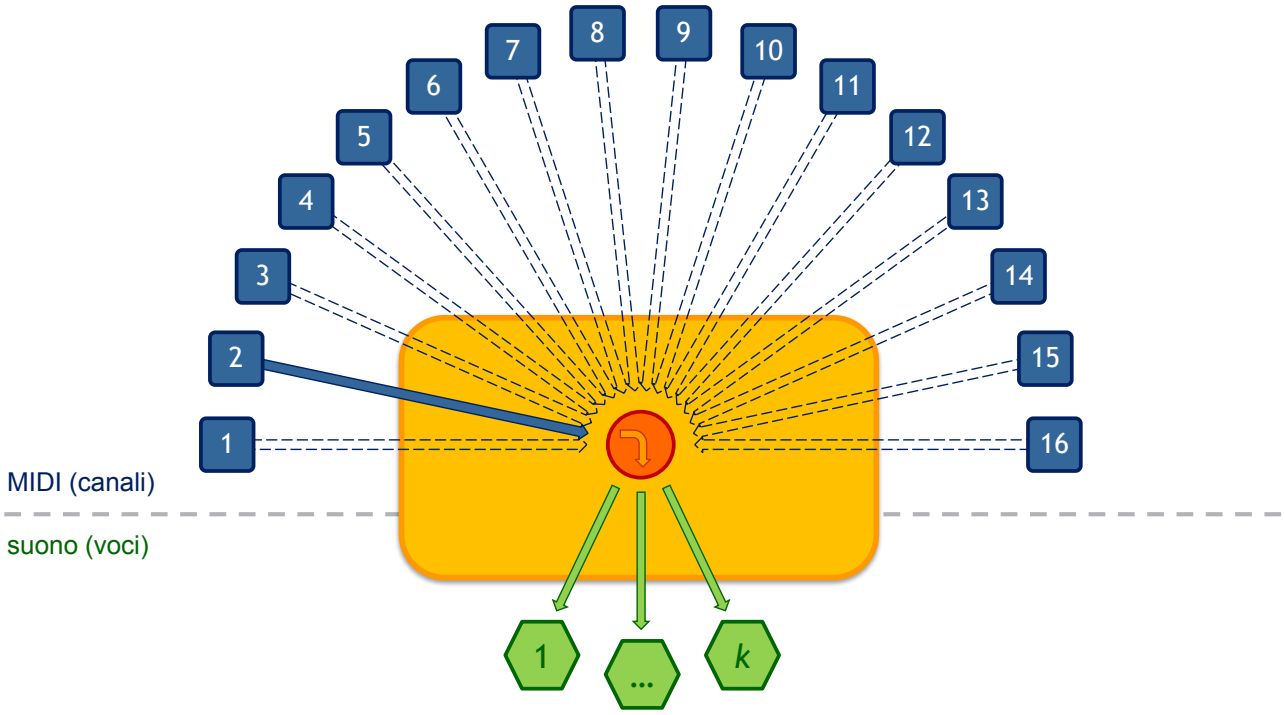


Figura 2.24. Rappresentazione grafica del Modo 3.

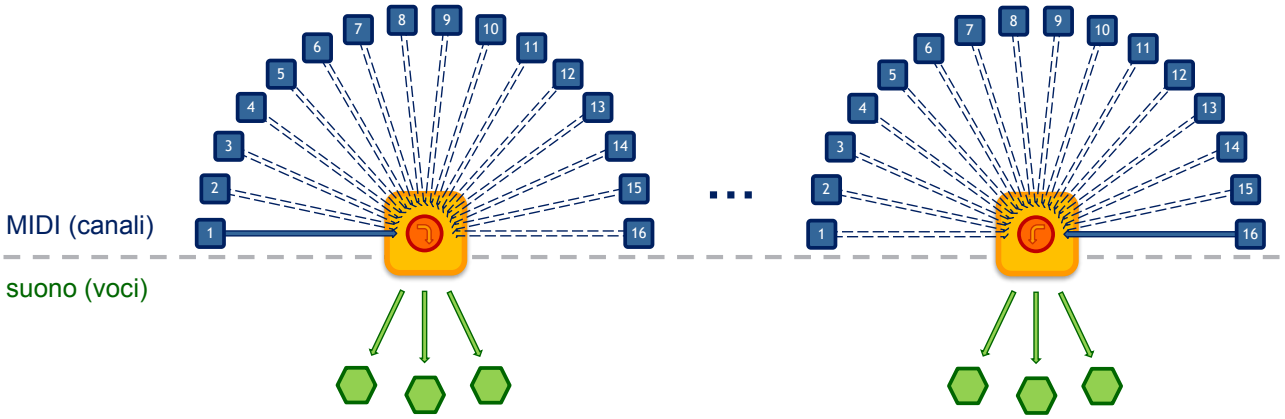


Figura 2.25. Rappresentazione grafica del Multi Mode. Si tratta di una replica del Modo 3 canale per canale.

differente ai messaggi in arrivo su canali differenti. Se l'impostazione polifonica del Modo 3 appare del tutto naturale, sorprende invece l'impostazione *omni off*, che comporta la ricezione dei messaggi sul solo canale base anziché sui 16 canali. In realtà, sebbene i sintetizzatori multitimbrici si presentino esternamente come un solo dispositivo, internamente si comportano come un set di moduli sonori indipendenti, ciascuno impostato su un canale base differente e operante in Modo 3. Questa reinterpretazione del Modo 3, che consente al dispositivo di rispondere contemporaneamente a più canali MIDI indipendenti, ognuno dei quali polifonico, è comunemente detta **Multi Mode** e viene rappresentata graficamente nella Figura 2.25.

Modo 4 (*omni off, mono*) e Mono Mode

Un dispositivo che opera in **Modo 4** gestisce i messaggi relativi alle voci di un singolo canale (*omni off*) e li assegna a una sola voce (*mono*). Lo scenario viene illustrato nella Figura 2.26, ove il nodo centrale rappresenta graficamente la selezione dei messaggi provenienti da un singolo canale.

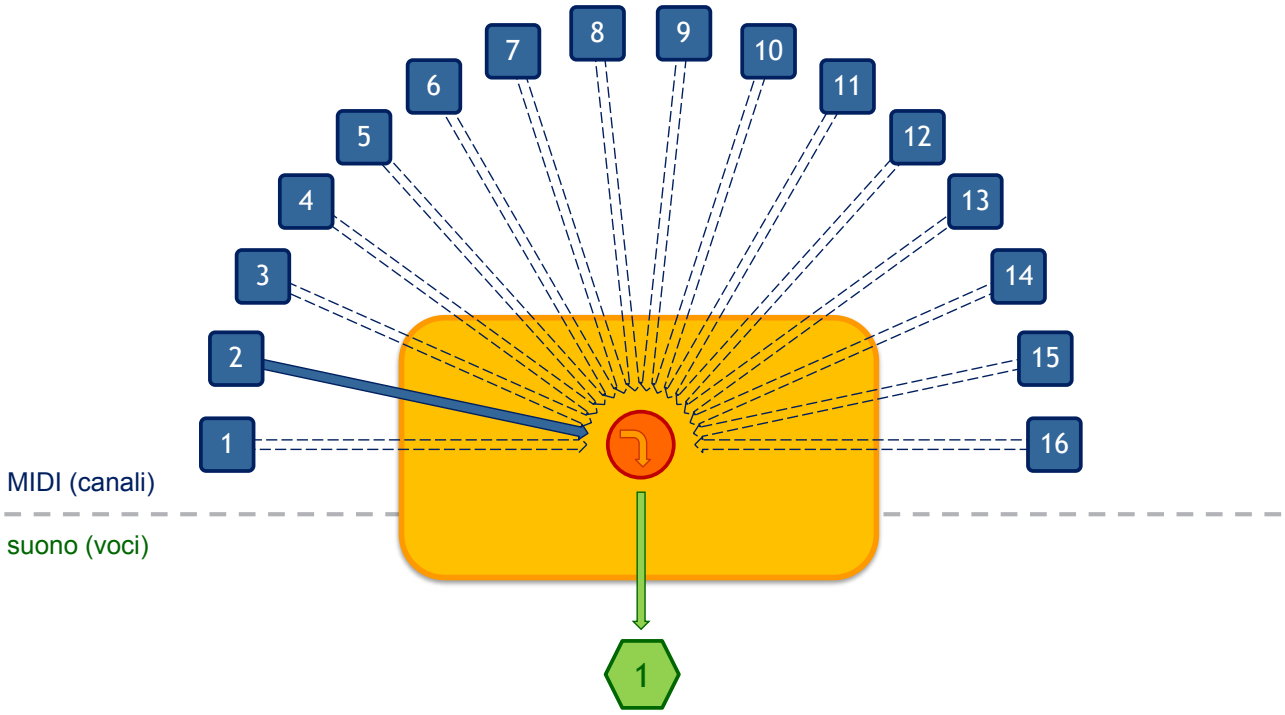


Figura 2.26. Rappresentazione grafica del Modo 4.

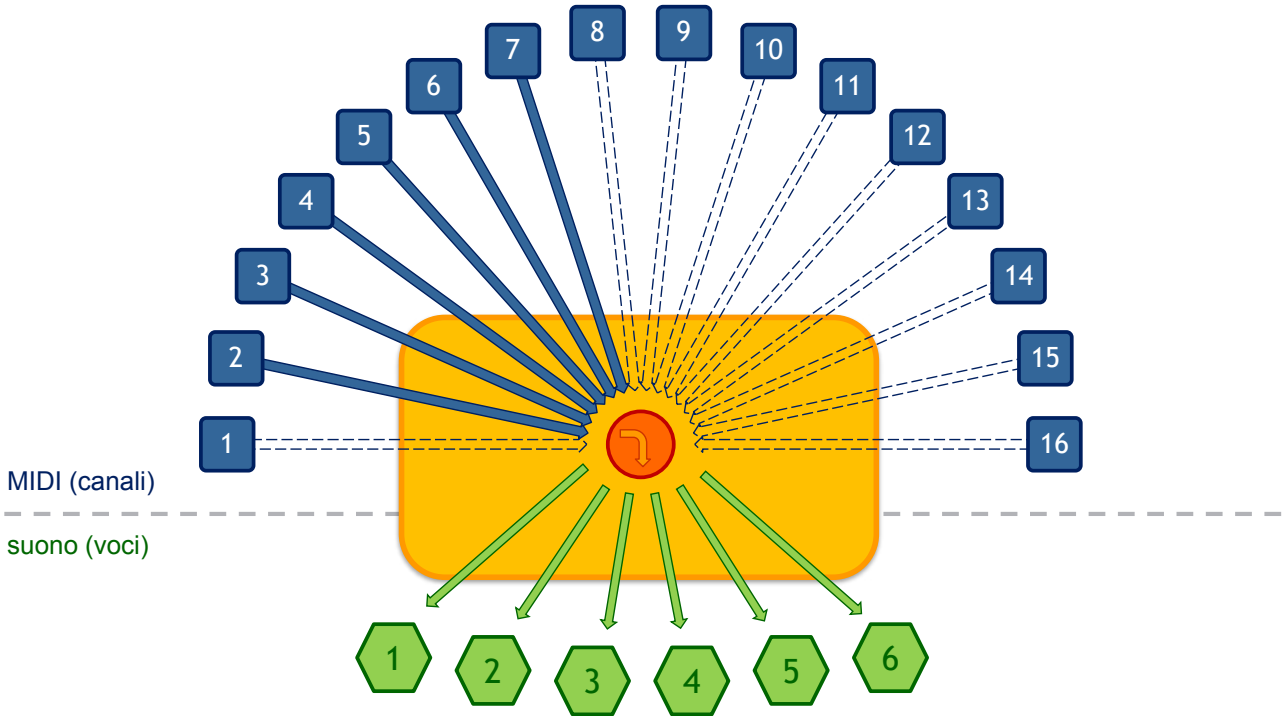


Figura 2.27. Rappresentazione grafica del Mono Mode. Si tratta di una replica del Modo 4, in cui messaggi di canali diversi vengono instradati monofonicamente su voci diverse.

Un ricevitore impostato in Modo 4 gestisce monofonicamente messaggi MIDI mantenendo distinta la loro appartenenza a canali differenti. In accordo con quanto illustrato sopra, sarebbe lecito aspettarsi che il dispositivo si ponga in ascolto sul solo canale base, ma i produttori hanno pensato a una declinazione differente della proprietà *omni off*. Allo scopo il Modo 4 sfrutta il valore codificato attraverso il secondo byte di dati del messaggio MONO ON (si veda il Paragrafo 2.7.2), valore viene invece ignorato nel Modo 2.

Per un ricevitore operante in Modo 4, detto N il canale base, i messaggi relativi alle voci vengono

ricevuti sui canali da N a $N+M-1$, e assegnati monofonicamente alle voci da 1 a M. Il numero di voci M viene specificato attraverso il secondo byte di dati del messaggio MONO ON. Il massimo numero di voci coincide con 16: si tratta del caso in cui il canale base N è 1 e M assume il massimo valore consentito [10]. In altre parole, M fissa il numero di canali MIDI a cui il dispositivo risponde. L'invio di $M=1$ determina un comportamento da strumento monofonico, anche per un dispositivo multitimbrico, come nello spirito originario del Modo 4. Il valore speciale $M=0$ viene utilizzato per istruire il ricevitore nell'usare tutte le voci a disposizione per i messaggi in arrivo sui canali da N a 16. Poiché si opera in contesto monofonico, viene allocata una voce per canale.

Per un trasmettitore in Modo 4, i messaggi relativi alle voci da 1 a M vengono inviati monofonicamente sui canali da N a $N+M-1$. In questo caso ha luogo l'invio di una voce per canale.

Anche il Modo 4 presenta una versione estesa che, per la sua importanza in uno specifico dominio applicativo, ha assunto un nome a sé stante: **Mono Mode** (Figura 2.27). Questa modalità, utilizzata normalmente per le chitarre MIDI, consente di utilizzare contemporaneamente più canali, uno per ciascuna corda del *controller*. Grazie all'indipendenza tra i canali in uso, il Mono Mode consente un migliore tracciamento della performance, la possibilità di trasposizione e di pitch bend indipendente per ciascun canale, l'assegnazione di timbri diversi (tramite *patch* distinte) alle singole corde. Considerando invece l'aspetto monofonico, il Mono Mode può essere adottato per introdurre effetti di portamento e legato. L'uso del Mono Mode può essere esteso ad altri *controller* con caratteristiche simili, ad esempio gli strumenti MIDI ad arco.

2.7.4 Il messaggio ALL NOTES OFF

ALL NOTES OFF è un messaggio appartenente alla famiglia *Channel Mode* che implementa un metodo efficiente per spegnere tutte le voci attive. Dal punto di vista sintattico, esso si presenta come un CONTROL CHANGE il cui primo byte di dati è posto a 123 (01111011_2, $7B_{16}$), mentre il secondo byte di dati ha un valore indifferente ai fini della funzione, convenzionalmente fissato a 0.

Non essendo necessario che ogni ricevitore sappia gestire il messaggio, le specifiche suggeriscono di inviare ove possibile i singoli messaggi NOTE-OFF prima di inviare ALL NOTES OFF. Inoltre il messaggio non è pensato per essere trasmesso periodicamente come parte del normale funzionamento del *setup* MIDI; esso, piuttosto, dovrebbe essere usato solo per indicare che l'intero sistema MIDI è "a riposo".

Un caso particolare riguarda le tastiere con funzioni di sintesi, tipicamente in grado di mandare in esecuzione anche le note accese tramite messaggi NOTE-ON in arrivo dalla catena MIDI a monte. Per questa categoria di dispositivi l'effetto di ALL NOTES OFF dovrebbe disattivare le sole note accese via MIDI, non quelle attive per la pressione locale dei tasti. Se uno strumento di questo tipo non è in grado di distinguere tra tastiera locale e messaggi MIDI in arrivo, le specifiche MIDI richiedono di ignorare ALL NOTES OFF.

Il messaggio deve essere ignorato anche dai ricevitori operanti in modalità OMNI ON. Infatti i dispositivi nel Modo 1 e 2 ricevono e gestiscono messaggi potenzialmente provenienti da diversi canali; essendo ALL NOTES OFF un messaggio di canale, la sua ricezione dovrebbe disattivare solo le note accese in quanto inviate su tale canale, il che sarebbe complesso da un punto di vista implementativo e poco comprensibile a livello di performance. Quando il ricevitore è impostato in Modo 3 ALL NOTES OFF annulla i messaggi NOTE-ON solo sul canale base. Quando il ricevitore è impostato in Modo 4 ALL NOTES OFF annulla solo i messaggi NOTE-ON sul canale su cui è stato ricevuto il messaggio.

Infine, va notato che ALL NOTES OFF, anche quando correttamente gestito, potrebbe non provocare lo spegnimento dell'emissione sonora da parte di un sintetizzatore: è il caso, ad

[10] Non si tratta, ovviamente, del massimo valore di M rappresentabile attraverso il byte di dati dedicato, pari, come di consueto, a 127, ma piuttosto del massimo valore che abbia significato nella definizione del comportamento del Modo 4. Infatti, poiché l'espressione rappresenta un numero di canale, si deve verificare la condizione $N+M-1 \leq 16$; ponendo $N=1$, che è il valore minimo ammissibile per N e pertanto massimizza il possibile valore di M, risulta $M \leq 16$.

esempio, di suoni prolungati attraverso l'uso dell'Hold pedal. Peraltro il comportamento è coerente rispetto alla ricezione di NOTE-OFF durante la pressione del pedale. Per provocare l'interruzione dell'emissione sonora esiste un altro messaggio – ALL SOUND OFF – descritto nel prossimo paragrafo.

Gli altri messaggi della famiglia *Channel Mode*, oltre a svolgere la propria specifica funzione, introducono sul canale lo stesso effetto di ALL NOTES OFF.

La Figura 2.28 fornisce una rappresentazione grafica del messaggio.

Figura 2.28. Rappresentazione grafica del messaggio ALL NOTES OFF.

2.7.5 Il messaggio ALL SOUND OFF

ALL SOUND OFF è un messaggio appartenente alla famiglia *Channel Mode* il cui scopo è interrompere l'emissione sonora nei dispositivi attivi sul canale. Alla ricezione, non solo viene operato uno spegnimento "logico" delle note accese, come in ALL NOTES OFF, ma si richiede di annullare il volume dei suoni nel più breve tempo possibile.

Dal punto di vista sintattico, esso si presenta come un CONTROL CHANGE il cui primo byte di dati è posto a 120 (01111000_2, 78_{16}), mentre il secondo byte di dati ha un valore indifferente ai fini della funzione, convenzionalmente fissato a 0.

Questo messaggio non va interpretato come una scorciatoia per sostituire ALL NOTES OFF, i singoli NOTE-OFF, o *controller* quali *hold pedal* rilasciato, *master volume* impostato a 0, ecc.

Sebbene originariamente destinato ai moduli sonori, il messaggio può essere applicato ad altri domini e a dispositivi diversi, ad esempio per spegnere tutte le luci in una console compatibile con MIDI o per cancellare il buffer audio di un riverbero o di un ritardo digitale controllato via MIDI.

La Figura 2.29 fornisce una rappresentazione grafica del messaggio.

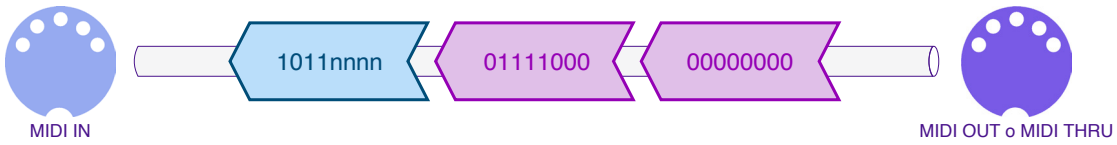


Figura 2.29. Rappresentazione grafica del messaggio ALL SOUND OFF.

2.7.6 Il messaggio RESET ALL CONTROLLERS

RESET ALL CONTROLLERS è un messaggio appartenente alla famiglia *Channel Mode* il cui scopo è istruire i dispositivi in ricezione a ripristinare un ideale stato iniziale. In particolare, vanno reimpostati i valori dei *controller* continui e a interruttore, di pitch bend, di pressione sul canale. Ad esempio, la *modulation wheel* va riportata a 0 e il controllo di *pitch bend* in posizione centrale.

Dal punto di vista sintattico, RESET ALL CONTROLLERS si presenta come un CONTROL CHANGE il cui primo byte di dati è posto a 121 (01111001_2, 79_{16}), mentre il secondo byte di dati ha un valore indifferente, convenzionalmente fissato a 0.

Come nel caso di ALL NOTES OFF, e per motivazioni analoghe, il messaggio deve essere ignorato dai dispositivi in modalità OMNI ON.

La Figura 2.30 fornisce una rappresentazione grafica del messaggio.

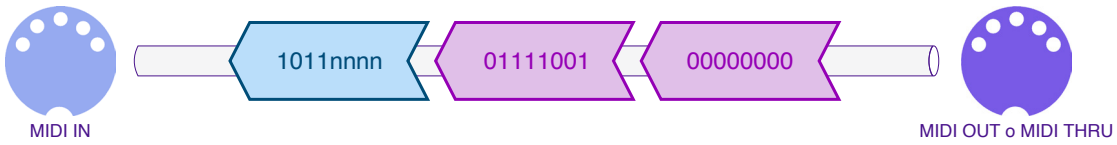


Figura 2.30. Rappresentazione grafica del messaggio RESET ALL CONTROLLERS.

2.7.7 MIDI Panic

Avendo introdotto i messaggi ALL NOTES OFF (Paragrafo 2.7.4), ALL SOUND OFF (Paragrafo 2.7.5) e RESET ALL CONTROLLERS (Paragrafo 2.7.6), è ora possibile definire la funzione *MIDI Panic* messa a disposizione da alcuni dispositivi hardware e software, tra cui molti *sequencer* e DAW. Il suo scopo è risolvere in modo rapido e intuitivo, soprattutto in un contesto di *live performance*, un evidente problema nel *setup* MIDI, che spesso si palesa nel prolungamento indefinito di note accese.

Le specifiche MIDI non menzionano esplicitamente tale terminologia, dunque la scelta di introdurre una funzione *MIDI Panic* e la strategia di implementazione relativa dipendono esclusivamente dai produttori. In generale ci si basa sull'invio di uno o più CONTROL CHANGE: CC 120 (All Sounds Off) e/o CC 123 (ALL NOTES OFF), e in alcuni casi anche CC 121 (RESET ALL CONTROLLERS). L'implementazione più semplice si limita al solo CC 123.

2.7.8 Il messaggio LOCAL CONTROL

Il messaggio LOCAL CONTROL chiude la rassegna della famiglia *Channel Mode*. Il suo scopo è interrompere o riattivare il controllo interno tra la tastiera (o altra forma di *controller*) e il modulo di generazione del suono in un dispositivo che include entrambi i componenti. Il campo di applicabilità del messaggio, quindi, si limita a una particolare categoria di strumenti MIDI.

Sintatticamente, LOCAL CONTROL si presenta come un CONTROL CHANGE il cui primo byte di dati è posto a 122 (01111010_2, $7A_{16}$), mentre il secondo byte di dati serve ad accendere o spegnere il controllo locale. Un messaggio di LOCAL CONTROL OFF viene veicolato ponendo il secondo byte di dati a 0. In questo caso, le informazioni di performance generate dal *controller* vengono inviate unicamente al connettore MIDI OUT, mentre i circuiti di generazione del suono sono controllati dai soli dati in arrivo su MIDI IN. Il normale funzionamento viene ripristinato da LOCAL CONTROL ON, che corrisponde al valore 127 (01111111_2, $7F_{16}$) per il secondo byte di dati.

Le specifiche MIDI raccomandano la modalità LOCAL CONTROL On all'accensione dei dispositivi, e sottolineano l'importanza di poter modificare tale impostazione, non solo tramite messaggi MIDI, ma anche dal pannello di controllo fisico dello strumento.

La Figura 2.31 presenta una rappresentazione grafica del messaggio.

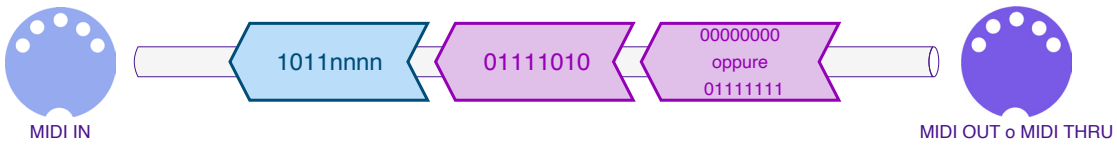


Figura 2.31. Rappresentazione grafica del messaggio LOCAL CONTROL, spento dal valore 0 e acceso dal valore 127 del secondo byte di dati.

2.8 I messaggi *System Common*

I messaggi di sistema comuni (*System Common*) si rivolgono all'intero *setup* MIDI, indipendentemente dal canale base dei dispositivi, e per questo vengono detti messaggi comuni

di sistema.

Considerando il byte di stato, rimane un'unica combinazione disponibile per i 3 bit normalmente dedicati all'identificazione della funzione: 111_2. Va però detto che l'intera categoria dei messaggi di sistema – ossia le famiglie *System Common*, *System Real Time* e *System Exclusive* – non necessita dell'informazione di canale: per questo motivo i 4 bit di ordine inferiore all'interno del byte di stato possono essere utilizzati per discernere potenzialmente tra 16 messaggi diversi. In altre parole, l'intervallo di valori [1111000_2, 11111111_2], oppure [$F0_{16}$, FF_{16}], è riservato a tutti i messaggi di sistema che – dal punto di vista logico – possono afferire alle famiglie *System Common*, *System Real Time* e *System Exclusive*.

I messaggi della famiglia *System Common* includono: MTC QUARTER FRAME, SONG SELECT, SONG POSITION POINTER, TUNE REQUEST, SYSTEM EXCLUSIVE ed END OF EXCLUSIVE (EOX). Si tratta in tutto di 6 messaggi il cui byte di stato si colloca nell'intervallo [11110000_2, 11110111_2], ossia [$F0_{16}$, $F7_{16}$]. All'interno di tale intervallo due combinazioni risultano non definite: 11110100_2, ossia $F4_{16}$, e 11110101_2, ossia $F5_{16}$.

2.8.1 Il messaggio MTC QUARTER FRAME

Il messaggio MTC QUARTER FRAME serve a sincronizzare dispositivi MIDI con altra strumentazione, in particolare apparecchiature audio, video e cinematografiche. Si tratta di uno dei due meccanismi di sincronizzazione attuabili attraverso messaggi MIDI, detto *MIDI Time Code* (MTC); l'altro approccio è basato sul messaggio TIMING CLOCK, che verrà descritto nel Paragrafo 2.9.1.

Il *MIDI Time Code* identifica istanti temporali in modo assoluto, misurando il tempo trascorso dall'origine (ad esempio, dall'inizio della *song*MIDI) in ore, minuti, secondi, fotogrammi e sottoparti. L'informazione così codificata può essere facilmente messa in relazione con l'*SMPTE* [11] *time code*, un insieme di standard per l'identificazione univoca di fotogrammi video o su pellicola cinematografica. Il time code si presenta nella forma:

```
hh:mm:ss:ff
```

ove `hh` sta per ore, `mm` per minuti, `ss` per secondi e `ff` per fotogrammi. Tralasciando l'informazione sui fotogrammi, troppo specifica per l'utente finale, si tratta dell'identificazione temporale comunemente usata nei player multimediali, nei videoregistratori e nei lettori di dischi ottici.

Mentre le prime tre componenti del *time code* sono espresse in unità di tempo assolute (ore, minuti e secondi), l'ultima (fotogrammi) fa necessariamente riferimento al numero di fotogrammi al secondo, detto *frame rate* e misurato in *frames per second* (fps). Esistono differenti standard al riguardo:

- 24 fps. Standard in uso in ambito cinematografico;
- 25 fps. Formato per la televisione a colori nello standard PAL (in uso in Europa, Uruguay, Argentina, Australia) e SECAM (originario della Francia, tuttora presente in alcuni paesi africani e asiatici);
- 29,97 fps, detto anche 30 fps drop frame. *Frame rate* per la televisione a colori nello standard NTSC (in uso nei paesi delle Americhe, quali USA, Canada, Messico e Colombia);
- 30 fps. Standard in contesto HDTV.

Il codice SMPTE in formato digitale richiede 80 bit, inclusi 32 bit per i dati utente. I messaggi MTC QUARTER FRAME hanno l'obiettivo di veicolare l'informazione temporale assoluta dello standard SMPTE su un *setup* MIDI.

La struttura di MTC QUARTER FRAME è costituita da un byte di stato, posto a 11110001_2 o equivalentemente a $F1_{16}$, seguito da un solo byte di dati (Figura 2.32). Poiché quest'ultimo consente

[11] SMPTE è l'acronimo di Society of Motion Picture and Television Engineers.

rappresentazioni numeriche su soli 7 bit, molto al di sotto di quanto richiesto dal codice SMPTE, le specifiche MIDI prevedono l'invio di più messaggi MTC QUARTER FRAME al fine di scomporre il marcatore temporale all'interno dei vari byte di dati. La tecnica richiama quanto visto per i *controller* continui ad alta risoluzione (Paragrafo 2.5.2), ma in questo caso, si tratta di suddividere quattro valori anziché uno: ore, minuti, secondi e fotogrammi; è inoltre necessario codificare lo standard di riferimento in uso per il *frame rate*.

All'interno del byte di dati devono trovare posto sia il valore numerico del marcatore ("di cosa si tratta"), sia la codifica della parte di valore trasportata dal messaggio ("quanto vale"). Per questo motivo il byte di stato viene a sua volta internamente ripartito come $\texttt{0nnndddd}_2$, ove $\texttt{nnn}_2$ è l'identificativo a 3 cifre della porzione di valore trasmesso, mentre $\texttt{dddd}_2$ è una parte del valore che ne rappresenta il *nibble*[12] più significativo o meno significativo. Riguardo agli identificativi su 3 cifre binarie:

- `000` per il *nibble* meno significativo del valore del fotogramma corrente;
- `001` per il *nibble* più significativo del fotogramma;
- `010` per il *nibble* meno significativo del valore del secondo corrente;
- `011` per il *nibble* più significativo del secondo;
- `100` per il *nibble* meno significativo del valore del minuto corrente;
- `101` per il *nibble* più significativo del minuto;
- `110` per il *nibble* meno significativo del valore dell'ora corrente;
- `111` per l'informazione restante sull'ora e sul *frame rate*. In questo caso, il byte di dati può essere ulteriormente ripartito in $\texttt{01110rrd}_2$, ove:
  - lo 0 iniziale identifica il byte di dati;
  - i tre bit successivi sono posti a 1, come identificativo di questa specifica parte del *time code*;
  - lo 0 successivo è un bit non utilizzato;
  - le cifre binarie `rr` costituiscono un codice per i quattro valori supportati del *frame rate*;
  - il solo bit meno significativo partecipa, in qualità di bit più significativo, alla ricostruzione del valore delle ore parzialmente espresso dal messaggio precedente.

  Riguardo ai codici in uso per `rr`, $\texttt{00}_2$ è associato a 24 fps, $\texttt{01}_2$ a 25 fps, $\texttt{10}_2$ a 30 fps drop frame, infine $\texttt{11}_2$ a 30 fps.

I messaggi MTC QUARTER FRAME devono essere inviati in questo ordine per consentire al ricevitore di ricostruire correttamente la marcatura temporale. Si tratta, quindi, di 8 messaggi aventi lo stesso byte di stato e byte di dati costruiti, come mostrato sopra, con una prima parte (su

[12] Viene detto *nibble* un raggruppamento di 4 bit, per cui un byte risulta formato da 2 *nibble*. In base 16, ogni *nibble* è rappresentabile con una singola cifra esadecimale.

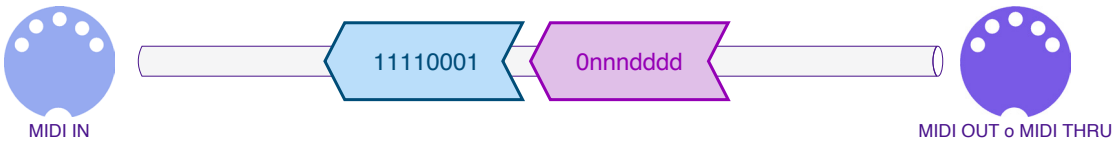


Figura 2.32. Rappresentazione grafica del messaggioMTC QUARTER FRAME.

3 bit) dal valore, variabile ma crescente, e da una seconda parte (su 4 bit) che reca l'informazione di marcatura temporale vera e propria.

Una prima considerazione riguarda l'intervallo di valori esprimibili per minuti, secondi e frame attraverso lo stratagemma della scomposizione in 4 + 4 bit. I valori ricostruiti sono a 8 bit, quindi si collocano nell'intervallo [0, 255], il che consente una piena rappresentazione dei minuti in un'ora, dei secondi in un minuto e dei frame in un secondo (considerando che il massimo *frame rate* è pari a 30). Non sarebbe stato possibile destinare un solo messaggio per componente del *time code*, perché 4 bit avrebbero limitato l'intervallo a [0, 15]. Un discorso a parte meritano le ore, la cui rappresentazione ha luogo su 1 + 4 bit: si tratta comunque dell'intervallo [0, 31], ampiamente sovradimensionato per gli scopi di marcatura temporale di un supporto multimediale.

Una seconda considerazione concerne la frequenza di invio di tali messaggi. Infatti, volendo ottenere un inseguimento preciso del valore temporale corrente, alla riproduzione di ogni nuovo fotogramma (quindi in un tempo assoluto che dipende dal *frame rate*, ma che oscilla tra 33 e 42 ms circa) andrebbero immessi sul canale trasmissivo ben 8 messaggi. In realtà, all'interno di tale intervallo il protocollo richiede l'invio di soli 4 messaggi, motivo per cui il messaggio è detto "quarter frame"; ne consegue che una marcatura completa per un'esatta locazione (ossia un dato fotogramma) potrà avvenire ogni 2 fotogrammi.

MTC QUARTER FRAME è un messaggio che implementa una forma di sincronizzazione, ossia un processo in cui il tempo gioca un ruolo fondamentale, per cui le latenze dovrebbero essere minimizzate. Se il flusso di dati MIDI è prossimo a saturare la capacità del canale trasmissivo, i messaggi MTC potrebbero arrivare in ritardo rispetto all'istante desiderato, introducendo un effetto *jitter*. Per questo motivo si suggerisce di utilizzare porte MIDI e un canale trasmissivo dedicati ai soli dati MTC.

La tecnica MTC consente di sincronizzazione un *sequencer* o una DAW con altri dispositivi in grado di interpretare il *time code* MIDI. Questo primo scenario è vincolato alla possibilità, per mittente e destinatario, di scambiare dati MIDI, il che richiede la compatibilità con il protocollo e l'appartenenza a uno stesso *setup* MIDI. Un altro scenario d'uso prevede, invece, l'aggancio di strumenti MIDI ad apparecchiature dotate di *time code* SMPTE, come, ad esempio, un videoregistratore; in questo caso si rende necessario l'impiego di un convertitore SMPTE/MTC.

MTC può essere utilizzato, ad esempio, per sincronizzare il playback di una *song* MIDI con un riproduttore di filmati o un *editor* video esterno. Nella comunicazione tra un'apparecchiatura SMPTE e un dispositivo MIDI, possibile con l'intervento di un apposito convertitore, ciascuno dei due attori può svolgere il ruolo di controllo (*master*) rispetto all'altro (*slave*). È facile immaginare come SMPTE possa fornire un segnale di temporizzazione che pilota i dispositivi MIDI. Si pensi, ad esempio, al *time code* SMPTE generato da un lettore Betacam SP [13], poi convertito in MTC per controllare una performance in una DAW. Il caso contrario, ossia dispositivi SMPTE controllati via MIDI e in grado di rispondere a un codice generato dalla conversione di MTC, richiede l'invio di messaggi SYSTEM EXCLUSIVE (Paragrafo 2.10) e l'adozione di un apposito protocollo detto MIDI Machine Control (Paragrafo 3.1).

Esempio

Si voglia ricostruire l'esatta marcatura temporale in formato hh:mm:ss:ff e il tipo di codifica SMPTE partendo dalla seguente sequenza di messaggi MTC QUARTER FRAME:

```
11110001 00001010
11110001 00010001
11110001 00100010
11110001 00110000
11110001 01000011
```

[13] Betacam identifica una famiglia di formati video professionali su nastro magnetico sviluppata da Sony a partire dal 1982. Betacam SP ("Superior Performance") è stato uno standard analogico assai diffuso per il video di classe broadcast.

```
11110001 01010010
11110001 01100001
11110001 01110110
```

La ricostruzione del valore SMPTE è la seguente:

- per le ore, preponendo il bit meno significativo del byte di stato dell'ultimo messaggio al *nibble* di ordine inferiore del byte di stato del messaggio precedente, si ottiene $00001_2 = 1$;
- per i minuti, considerando i *nibble* meno significativi dei messaggi corrispondenti, si ottiene $00100011_2 = 35$;
- procedendo allo stesso modo per i secondi, si ha $00000010_2 = 2$;
- infine, per i fotogrammi, $00011010_2 = 26$.

In definitiva, l'invio degli 8 messaggi sopra riportati si traduce nel *time code* 01:35:02:26. Riguardo al numero di fotogrammi al secondo, fondamentale per poter calcolare l'istante temporale in termini assoluti, l'informazione si ricava dal terzultimo e dal penultimo bit del byte di dati dell'ultimo messaggio. Al codice 11_2 corrisponde il *frame rate* di 30 fps.

2.8.2 Il messaggio SONG SELECT

Il messaggio SONG SELECT si utilizza con dispositivi hardware o software che contengono brani (detti anche *MIDI songs*, letteralmente: canzoni) o sequenze MIDI. Tra le categorie di strumenti interessate vanno citati i *sequencer* e le *drum machine*. Il messaggio in oggetto, in particolare, istruisce il dispositivo nel caricare il brano specificato.

Dal punto di vista sintattico, SONG SELECT si compone di:

- un byte di stato prefissato, posto a 11110011_2 o, equivalentemente, a $F3_{16}$;
- un singolo byte di dati atto a specificare il numero di brano all'interno di una collezione. Dall'intervallo di valori rappresentabile su 7 bit, si evince che la numerosità massima della collezione sia limitata a 128 brani.

A livello di interfaccia utente la maggior parte dei dispositivi visualizza i numeri di brano a partire da 1 anziché da 0, ma va ricordato che il primo brano in una collezione viene sempre selezionato dal byte di dati posto a 0.

Quando un dispositivo riceve un messaggio SONG SELECT, carica il brano partendo dall'istante 0; questo comportamento può essere modificato dall'invio di un successivo messaggio SONG POSITION POINTER, descritto nel prossimo paragrafo. Le specifiche prevedono che il messaggio SONG SELECT possa essere inviato solo quando il sistema non è in riproduzione.

La Figura 2.33 fornisce una rappresentazione grafica del messaggio.

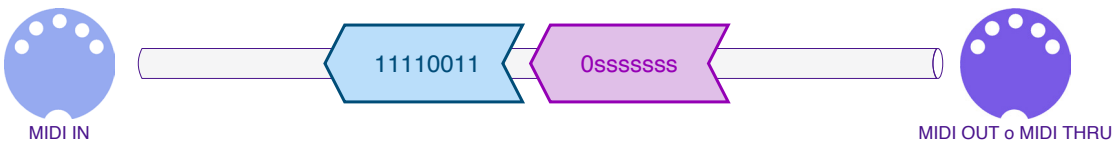


Figura 2.33. Rappresentazione grafica del messaggio SONG SELECT.

2.8.3 Il messaggio SONG POSITION POINTER

Il messaggio SONG POSITION POINTER serve a posizionare il puntatore all'interno del brano correntemente selezionato. Il valore che esso trasporta è correlato, secondo la formula mostrata sotto, al numero di *MIDI clocks* trascorsi tra l'inizio del brano e il punto desiderato. Tale concetto verrà spiegato in maggiore dettaglio nel Paragrafo 2.9.1; per il momento, il lettore può considerare

i *MIDI clocks* come una forma di misura del tempo rapportata al metronomo anziché a un'unita di misura assoluta.

Dal punto di vista sintattico, il messaggio è strutturato nel modo seguente:

- un byte di stato prefissato, posto a 11110010_2 o, equivalentemente, a $F2_{16}$;
- due byte di dati, che fungono, rispettivamente, da LSB e MSB per poter rappresentare su 14 bit lo scostamento della posizione del puntatore rispetto all'istante 0.

Il valore della distanza si misura in pulsazioni MIDI (*MIDI beats*), ove una pulsazione corrisponde a 6 *MIDI clocks*. La sincronizzazione si basa sull'invio periodico di messaggi TIMING CLOCK (Paragrafo 2.9.1) che vengono propagati sul *setup* MIDI a partire da una sorgente. Poiché si tratta di messaggi della famiglia *System Real Time*, SONG POSITION POINTER può essere utilizzato solo con dispositivi che riconoscono i messaggi MIDI in tempo reale.

Anche se MIDI non costituisce un formato per la rappresentazione dell'informazione di partitura, alla scansione metrica principale si fa convenzionalmente corrispondere la figurazione ritmica di un quarto, detta anche semiminima. Come si vedrà nel seguito, in una unità di tempo corrispondente al quarto vengono inviati 24 *MIDI clocks*. Questo permette di calcolare la granularità temporale nel posizionamento del puntatore ottenibile con SONG POSITION POINTER. Infatti, considerando il quarto come pulsazione principale, ricordando che questo è suddiviso in 24 *MIDI clocks* e che un *MIDI beat* equivale a 6 *MIDI clocks*, si ottiene una granularità di 24 / 6 = 4 suddivisioni del quarto, ossia una risoluzione al sedicesimo o alla semicroma.

Il limite dato dai 14 bit consente di rappresentare valori nell'intervallo [0, 16383], quindi la massima distanza punterebbe al 16384° sedicesimo, che equivale al 4096° quarto. Si tratterebbe dell'inizio di battuta 1024 in un tempo di 4/4 o di battuta 2048 in un tempo di 2/4. Per avere un metro di paragone, il primo movimento della *Sinfonia n°9* di Ludwig Van Beethoven è in tempo di 2/4 ed è costituito da 547 battute.

Il messaggio SONG POSITION POINTER può essere inviato solo quando il sistema non è in riproduzione. La risposta a un messaggio di questo tipo, così come la risposta a SONG SELECT, può richiedere del tempo, e il fatto che il playback non sia attivo rappresenta ovviamente un vantaggio. Però, immediatamente dopo la ricezione di SONG POSITION POINTER, potrebbe sopraggiungere un messaggio CONTINUE (spiegato nel seguito), il cui scopo è avviare la riproduzione, e nel frattempo il comando di riposizionamento potrebbe non aver concluso le operazioni. In tal caso, le specifiche MIDI chiedono di tener conto dei *MIDI clocks* intercorsi. Ad esempio, se: i) viene ricevuto SONG POSITION POINTER con il valore 4, pari a $4 \times 6 = 24$ *MIDI clocks*, ii) arriva un messaggio CONTINUE, e iii) il completamento dell'operazione richiede 3 *MIDI clocks*, allora il dispositivo di playback dovrebbe tenerne conto impostando il puntatore interno al *MIDI clock* 27 (sebbene questo non sia selezionabile attraverso la risoluzione offerta da SONG POSITION POINTER).

La Figura 2.34 visualizza il messaggio in forma grafica.

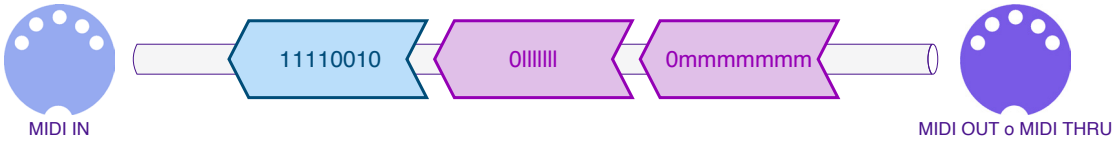


Figura 2.34. Rappresentazione grafica del messaggio SONG POSITION POINTER.

Esempi

Si consideri l'invio del seguente messaggio SONG POSITION POINTER:

```
11110010 00001000 00000000
```

Il valore espresso su 14 bit si traduce in 8 *MIDI beats*, ossia $8 \times 6 = 48$ *MIDI clocks*. Considerando che 24 *MIDI clocks* vengono fatti corrispondere per convenzione alla figura ritmica da un quarto e che la prima pulsazione ha luogo al *MIDI clock* 0, questo messaggio posiziona il puntatore all'inizio del terzo quarto del brano. Con le informazioni a disposizione non è possibile calcolare in quale battuta ricada tale posizione: a titolo di esempio, potrebbe trovarsi all'interno di battuta 1, se il tempo fosse 3/4 o 4/4, o essere la prima pulsazione di battuta 2 in tempo di 2/4.

Come esempio ulteriore, si voglia inviare un messaggio SONG POSITION POINTER per posizionare il *sequencer* alla seconda pulsazione di battuta 33 in tempo di 4/4. In tal caso, il calcolo da fare si compone dei seguenti passaggi. Poiché ogni quarto equivale a 4 sedicesimi e il tempo è 4/4, ogni battuta dura $4 \times 4 = 16$ sedicesimi. L'*offset* che porta a inizio di battuta 33 richiede di "consumare" $32 \times 16 = 512$ sedicesimi. A questi vanno aggiunti 4 sedicesimi di ulteriore *offset* per tener conto della prima pulsazione di battuta 33. In definitiva, la posizione desiderata è quella del sedicesimo 516. Ricordando che il primo sedicesimo del brano viene numerato 0, essa ricade all'inizio della seconda pulsazione di battuta 33. Si tratta ora di tradurre tale valore nel formato richiesto da SONG POSITION POINTER: 516 = 1000000100_2, che ripartito tra LSB e MSB diventa $0000100\ 0000100_2$. Il messaggio SONG POSITION POINTER da inviare è dunque:

```
11110010 00000100 00000100
```

.

In entrambi gli esempi, per poter calcolare la posizione in unità di tempo assolute sarebbe necessario un dato mancante, ossia il valore a cui è impostato il metronomo (*beats per minutes*, bpm): solo questo permetterebbe di stabilire la durata di una pulsazione in secondi.

2.8.4 Il messaggio TUNE REQUEST

Il messaggio TUNE REQUEST viene generalmente utilizzato per richiedere ai sintetizzatori analogici presenti nel *setup* MIDI di calibrare nuovamente i propri oscillatori interni. Il dispositivo ricevente, per poter gestire correttamente il messaggio, deve implementare un'apposita procedura di taratura. Per ovvi motivi questo messaggio non trova applicazione nei sintetizzatori digitali.

Trattandosi di una funzione completamente specificata attraverso il solo byte di stato, per la prima volta ci si confronta con un messaggio MIDI privo di byte di dati. Come mostrato nella Figura 2.35, si tratta dell'ottetto 11110110_2, o, analogamente, $F6_{16}$.

Figura 2.35. Rappresentazione grafica del messaggio TUNE REQUEST.

2.8.5 Il messaggio SYSTEM EXCLUSIVE

Il messaggio SYSTEM EXCLUSIVE (SYSEX) determina l'inizio di una comunicazione detta esclusiva, in quanto destinata a essere interpretata solo da determinati modelli di dispositivi costruiti da specifici produttori. L'argomento verrà trattato diffusamente nel Paragrafo 2.10. Va osservato che, pur dando avvio a un messaggio della famiglia *System Exclusive*, il messaggio SYSTEM EXCLUSIVE rientra dal punto di vista logico nella famiglia *System Common*.

L'informazione di avvio della comunicazione esclusiva è codificata dal byte di stato 11110000_2, ossia $F0_{16}$. Quella che segue è una sequenza potenzialmente lunga di byte di dati: essi rappresentano il contenuto della comunicazione esclusiva vera e propria, e dunque assumono

un significato diverso a seconda delle scelte implementative del produttore. Il primo o i primi tre byte di dati di SYSTEM EXCLUSIVE, però, sono sempre riservati al codice identificativo del costruttore, e per i byte immediatamente successivi c'è convergenza su una sorta di standard. Queste osservazioni verranno riprese nel Paragrafo 2.10.

La struttura del messaggio è mostrata nella Figura 2.36.

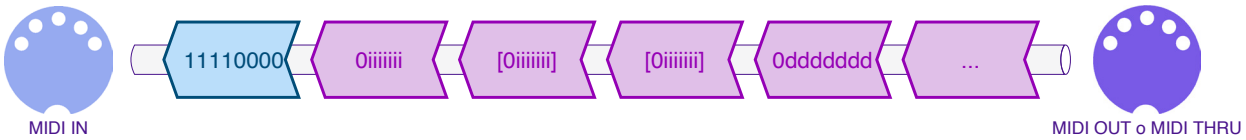


Figura 2.36. Rappresentazione grafica del messaggio SYSTEM EXCLUSIVE.

2.8.6 Il messaggio END OF EXCLUSIVE (EOX)

Il messaggio END OF EXCLUSIVE (EOX) determina la conclusione di un messaggio esclusivo, concludendo la sequenza di byte di dati del messaggio SYSTEM EXCLUSIVE precedente. Anche il messaggio in oggetto appartiene, dal punto di vista logico, alla famiglia *System Common*.

L'uso di una coppia di messaggi delimitatori rappresenta un meccanismo del tutto nuovo all'interno del protocollo MIDI, motivato però da una chiara esigenza: una comunicazione esclusiva è costituita da un numero variabile di byte, non determinabile a priori, mentre qualsiasi altro messaggio MIDI presenta un numero ben preciso di byte (1 byte di stato + n byte di dati, con n fissato dal tipo di messaggio). Si è dunque reso necessario contraddistinguere in maniera standard non solo l'avvio ma anche la fine di una comunicazione *System Exclusive*.

All'informazione EOX codificata dal byte di stato non è necessario aggiungere alcun valore; per questo motivo il messaggio si riduce al solo byte di stato 11110111_2, ossia $F7_{16}$.

Per evitare di bloccare il funzionamento del sistema il trasmettitore impegnato in una comunicazione esclusiva dovrebbe inviare il messaggio END OF EXCLUSIVE immediatamente dopo la sua conclusione, in modo che il ricevitore possa tornare al suo normale funzionamento. Si noti che un messaggio SYSTEM EXCLUSIVE si rivolge a uno specifico dispositivo all'interno del *setup*, quindi ha un livello di indirizzamento molto più raffinato rispetto ai messaggi di canale. Questo argomento verrà ripreso nel Paragrafo 2.8.5.

Le specifiche stabiliscono, inoltre, che qualsiasi byte di stato ponga fine a un messaggio esclusivo [14], quindi qualsiasi altro messaggio MIDI potrebbe svolgere la funzione di END OF EXCLUSIVE. Ciononostante, il protocollo richiede di inviare esplicitamente un messaggio END OF EXCLUSIVE per concludere questo tipo di trasmissione dati.

La struttura del messaggio viene mostrata nella Figura 2.37.

Figura 2.37. Rappresentazione grafica del messaggio END OF EXCLUSIVE.

[14] Fanno eccezione i messaggi della famiglia *Real Time* che, presentando vincoli temporali stringenti, possono collocarsi tra altri messaggi o addirittura tra i byte appartenenti a un singolo messaggio.

2.9 I messaggi *System Real Time*

I messaggi di sistema in tempo reale (*System Real Time*) vengono utilizzati per la sincronizzazione tra componenti MIDI basati su *clock*. In ambito informatico un sistema che opera in tempo reale ha l'obiettivo di completare il proprio task nel tempo stabilito: non deve essere necessariamente veloce, bensì altamente predicibile. Anche in questa accezione la componente temporale rimane di fondamentale importanza.

Come noto, il canale trasmissivo MIDI è lento e può essere intasato di altri messaggi. Per ridurre il problema della latenza, il protocollo riserva a questa famiglia alcuni accorgimenti: innanzitutto la dimensione dei messaggi è minimale, limitandosi al solo byte di stato; in secondo luogo, i messaggi in tempo reale hanno la possibilità di interfogliarsi anche tra byte appartenenti ad altri messaggi MIDI. Peraltro il loro riconoscimento da parte del ricevitore è semplice, in quanto si tratta di sequenze di bit prefissate, senza alcun grado di libertà; quando il ricevitore realizza l'arrivo di un ottetto di bit che coincide con l'identificativo di un messaggio in tempo reale (o, per meglio dire, con il messaggio stesso), è in grado di estrapolarlo dalla parte restante della comunicazione MIDI e di ricostruire l'eventuale messaggio interrotto. Per questa loro proprietà i messaggi della famiglia System Real Time sono gli unici a non determinare la conclusione di una comunicazione esclusiva (Paragrafi 2.8.5 e 2.8.6).

Dal punto di vista sintattico, il nibble di ordine alto del byte di stato risulta essere, ancora una volta, 1111_2. Per quanto riguarda il successivo *nibble*, rimangono disponibili le combinazioni di bit non occupate dalla famiglia *System Common*. In definitiva, il byte di stato per questa famiglia di messaggi si colloca nell'intervallo $[11111000_2, 11111111_2]$, ossia $[F8_{16}, FF_{16}]$. Come detto, nessun messaggio presenta byte di dati.

La famiglia *System Real Time* si compone complessivamente di 7 messaggi: TIMING CLOCK, START, STOP, CONTINUE, ACTIVE SENSING e SYSTEM RESET. All'interno dell'intervallo di valori riservato alla famiglia, una combinazione non risulta definita: si tratta di 11111101_2, ossia FD_{16}.

2.9.1 Il messaggio TIMING CLOCK

Il messaggio TIMING CLOCK implementa una forma di sincronizzazione alternativa a quanto mostrato nel Paragrafo 2.8.1, riservata ai soli dispositivi MIDI e basata sul tempo metronomico (misurato in bpm) anziché sul tempo assoluto (misurato in secondi).

Si tratta, nello specifico, di immettere sul canale trasmissivo 24 colpi di *clock* per pulsazione, detti *ticks*. Alla pulsazione si fa corrispondere convenzionalmente la figura ritmica del quarto, nota anche come semiminima. I *ticks* devono essere spaziati nel tempo nella maniera più uniforme possibile. La loro frequenza in termini assoluti dipende, ovviamente, dal tempo metronomico del brano: a 60 bpm si contano $60 \times 24 = 1440$ *ticks* al minuto (circa un impulso ogni 41,67 ms), a 100 bpm risultano 2400 *ticks* per minuto (un impulso ogni 25 ms), a 120 bpm sono 2880 *ticks* per minuto (uno ogni circa 20,83 ms), e via dicendo.

L'idea è quella di creare, attraverso l'invio di messaggi TIMING CLOCK, una sorta di griglia al cui interno inquadrare temporalmente i restanti messaggi MIDI, allineandoli rispetto alla risoluzione temporale così determinata. La griglia suddivide regolarmente la pulsazione da un quarto in 24 parti, il che corrisponde a una suddivisione in 12 parti dell'ottavo, in 6 parti del sedicesimo, in 3 parti del trentaduesimo. Il valore ritmico più piccolo che si riesce a quantizzare correttamente è il sessantaquattresimo all'interno di una terzina, ma non il sessantaquattresimo all'interno di una duina.

Per comprendere il meccanismo, si immagini uno strumentista in grado di eseguire un brano perfettamente a tempo. Due eventi NOTE-ON distanziati temporalmente dalla durata di un quarto resterebbero tali anche dopo il riallineamento alla griglia, in quanto cadrebbero all'interno di celle rispettivamente al *tick* n e $n+24$. Anche l'esecuzione completa (accensione e spegnimento) di una nota da un sedicesimo non introdurrebbe variazioni ritmiche, in quanto il NOTE-OFF si troverebbe al *tick* $n+6$ rispetto al corrispettivo NOTE-ON. Considerando una nota da un sessantaquattresimo,

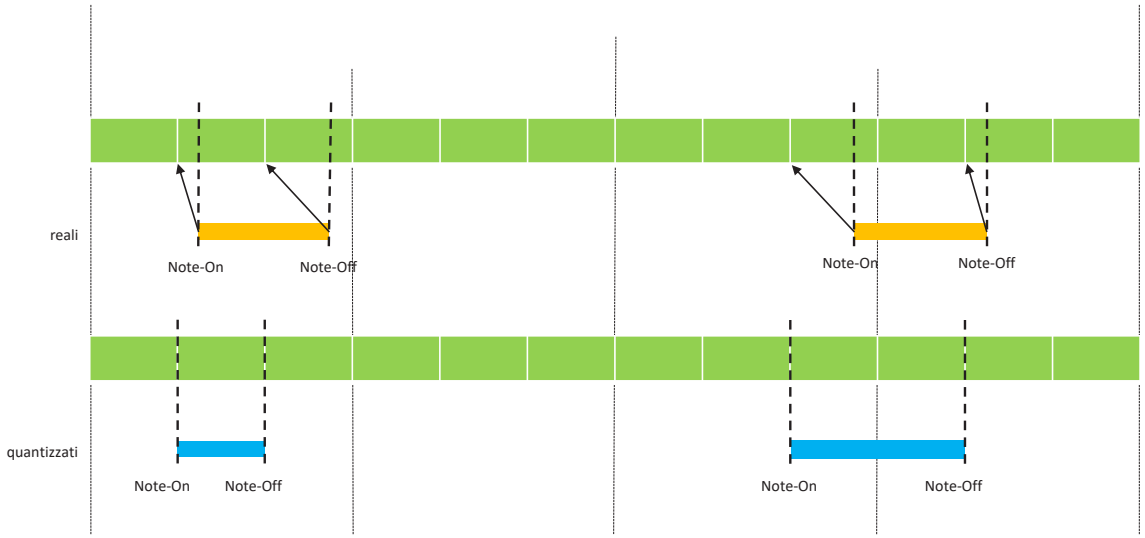


Figura 2.38. Esempio di allineamento di eventi MIDI distanti un sessantaquattresimo rispetto alla quantizzazione prevista da TIMING CLOCK.

invece, il suo NOTE-OFF disterebbe 1,5 *ticks* dal NOTE-ON, e questo intervallo potrebbe portare i due eventi:

- all'interno di due celle temporali contigue. Ad esempio, se NOTE-ON arrivasse al *tick* $n + 1/3$, NOTE-OFF si troverebbe a $n + 1/3 + 3/2 = n + 11/6 = (n + 1) + 5/6$, ricadendo dunque nella cella n+1;

- in due celle temporali suddivise da una cella intermedia. Ad esempio, se NOTE-ON arrivasse al *tick* $n + 2/3$, NOTE-OFF si troverebbe a $n + 2/3 + 3/2 = n + 13/6 = (n + 2) + 1/6$, ricadendo dunque nella cella n+2.

Un esempio degli scenari descritti è mostrato nella Figura 2.38.

Proseguendo nel ragionamento, i messaggi NOTE-ON e NOTE-OFF per una nota da un centoventottesimo potrebbero risultare all'interno della stessa cella, di fatto annichilendo l'evento sonoro, oppure in due celle contigue, assumendo quindi la durata di un sessantaquattresimo sotto terzina.

Si osservi che, nell'attuare questo meccanismo di temporizzazione, assume un ruolo fondamentale il dispositivo che funge da *master*, ossia lo strumento che invia il segnale di sincronizzazione sotto forma di periodici messaggi TIMING CLOCK. I restanti dispositivi nel *setup* MIDI a valle fanno riferimento a tale segnale, assumendo dunque il ruolo di *slave*.

La struttura del messaggio TIMING CLOCK è rappresentata dal solo byte di stato 11111000_2, ossia $F8_{16}$, come mostrato nella Figura 2.39.

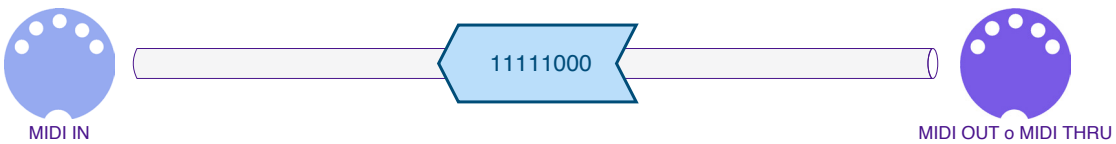


Figura 2.39. Rappresentazione grafica del messaggio TIMING CLOCK.

2.9.2 I messaggi START, STOP e CONTINUE

START, STOP e CONTINUE sono tre messaggi della famiglia *System Real Time* relativi all'esecuzione di una sequenza di messaggi MIDI, motivo per il quale si rivolgono tipicamente a una specifica categoria di dispositivi: i *sequencer*. Nel dettaglio:

- START avvia l'esecuzione della sequenza corrente. Il messaggio è rappresentato dal byte di stato 11111010_2, o, equivalentemente, FA_{16};
- STOP interrompe l'esecuzione della sequenza corrente. Il messaggio è rappresentato dal byte di stato 11111100_2, o, equivalentemente, FC_{16};
- CONTINUE riavvia l'esecuzione della sequenza corrente dal punto in cui è stata interrotta o riposizionata (in stato di pausa). Il messaggio è rappresentato dal byte di stato 11111011_2, o, equivalentemente, FB_{16}.

Le specifiche raccomandano coerenza tra lo stato di riproduzione in cui si trova il *sequencer* e il messaggio che gli viene inviato. Ad esempio, i messaggi START e CONTINUE dovrebbero essere ignorati quando la sequenza è già in esecuzione. Analogamente, il messaggio STOP dovrebbe essere ignorato se il dispositivo non è in fase di riproduzione.

La scelta della sequenza corrente (intesa come *MIDI song*) all'interno di una collezione e il riposizionamento del puntatore di lettura all'interno del brano selezionato sono delegati a due messaggi *System Common* già trattati: SONG SELECT (Paragrafo 2.8.2) e SONG POSITION POINTER (Paragrafo 2.8.3).

La Figura 2.40 mostra la rappresentazione grafica dei tre messaggi in oggetto.

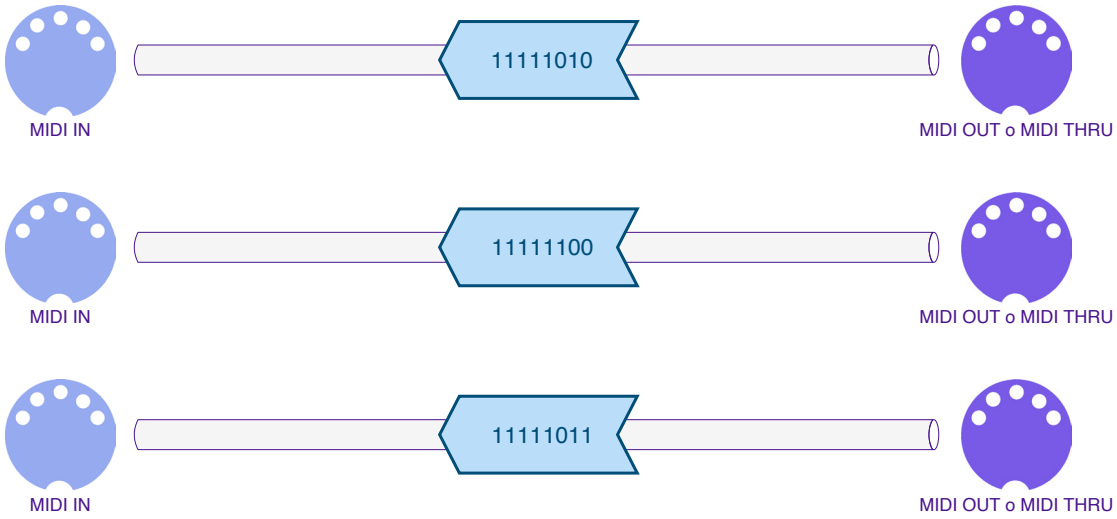


Figura 2.40. Rappresentazione grafica, dall'alto verso il basso, dei messaggi START, STOP e CONTINUE.

Esempio

Si voglia sincronizzare un *sequencer* con un altro dispositivo la cui riproduzione deve avvenire a tempo, come una *drum machine* che esegue pattern ritmici di accompagnamento per la sequenza MIDI. Entrambi gli strumenti devono partire allo stesso istante e suonare allo stesso metronomo. In questo scenario il *sequencer* funge da *master*, inviando messaggi di temporizzazione e istruendo i dispositivi a valle sulle azioni da intraprendere, mentre la *drum machine* è lo *slave*.

Il *master*, all'avvio della propria sequenza, deve essere in grado di avvisare tempestivamente lo *slave* affinché questo operi di conseguenza, ad esempio avviando a sua volta l'esecuzione del

pattern ritmico di accompagnamento. A tale scopo il *master*, che già attua un meccanismo di sincronizzazione del *setup* MIDI attraverso l'invio periodico di messaggi TIMING CLOCK, immette sul canale trasmissivo un messaggio START. La ricezione di START da parte dello *slave* pone il dispositivo in "play mode", uno stato di attesa in cui si prepara all'avvio della riproduzione. All'arrivo del successivo messaggio TIMING CLOCK, lo *slave* dà avvio alla riproduzione del pattern ritmico [15].

Da questo punto in avanti la serie di TIMING CLOCK ricevuti dallo *slave* funge da griglia cui si allineano gli eventi MIDI. Infatti, sapendo che la pulsazione da un quarto viene ripartita in 24 colpi di *clock*, lo *slave* può sincronizzare la riproduzione di eventi che distano, ad esempio, un sedicesimo (6 colpi di *clock*), un ottavo (12 colpi di *clock*), e via dicendo. Questo processo è robusto rispetto alla variazione di bpm nel *master*: essa si rifletterebbe in una variazione nella frequenza di invio di messaggi TIMING CLOCK da parte del *master* stesso, per cui verrebbe riconosciuta e correttamente gestita dallo *slave*.

Il *master* può fermare la sequenza in esecuzione nello *slave* inviando un messaggio STOP. Ad esempio, questa azione potrebbe avvenire in risposta alla pressione del pulsante "Pausa" sull'interfaccia grafica del *sequencer*: l'effetto diretto è interrompere localmente l'esecuzione della sequenza MIDI, ma l'azione si deve riverberare anche sul comportamento degli strumenti *slave* nel *setup*, che altrimenti procederebbero con l'esecuzione dei propri pattern. Il *master* può comunque continuare a inviare messaggi TIMING CLOCK; lo *slave*, ora in stato di stop, li riceverà senza provocare l'avanzamento dell'esecuzione.

L'utente potrebbe far riprendere l'esecuzione in un punto diverso rispetto all'inizio del brano o all'istante in cui il brano è stato messo in pausa. In questo caso, il *master* istruirà lo *slave* inviando un messaggio SONG POSITION POINTER, la cui risoluzione temporale – vale la pena ricordare – è il sedicesimo. Il motivo per cui SONG POSITION POINTER può essere inviato solo quando l'esecuzione è sospesa deriva dal tempo di risposta per determinare il punto di ripresa della performance: lo *slave*, durante l'operazione, potrebbe perdere uno o più colpi di *clock*, compromettendo la sincronizzazione con il *master*.

Per avviare la riproduzione dal punto selezionato, il *master* non deve inviare un messaggio START, che riporterebbe il puntatore all'inizio del brano, bensì un messaggio CONTINUE. Anche in questo caso è il TIMING CLOCK immediatamente successivo a determinare l'avvio vero e proprio dell'esecuzione.

CONTINUE svolge la stessa funzione anche a valle di uno STOP non seguito da SONG POSITION POINTER. In tal caso, lo *slave* deve riprendere dalla posizione raggiunta immediatamente prima dello STOP, tipicamente conservata in memoria come valore corrente di *song position*.

2.9.3 Il messaggio ACTIVE SENSING

Il messaggio ACTIVE SENSING (letteralmente: rilevamento attivo) può essere inviato ripetutamente nel *setup* MIDI allo scopo di comunicare ai dispositivi che la connessione non è caduta. Si tratta di un meccanismo di protezione rispetto all'interruzione indesiderata del canale, ma il suo uso rimane facoltativo.

Si rammenta che la comunicazione MIDI è unidirezionale, per cui un dispositivo a valle non ha modo di interrogare uno strumento a monte, implementando così un meccanismo di consegna affidabile dei messaggi. Si pensi al problema del possibile smarrimento di NOTE-OFF lungo il canale trasmissivo: la nota deve essere mantenuta accesa in quanto previsto dalla performance, oppure si tratta dell'effetto dell'interruzione nella comunicazione? Quest'ultimo caso è tutt'altro che infrequente: banalmente, è sufficiente un cavo inavvertitamente scollegato da una porta MIDI. In questo scenario un sintetizzatore non può sapere se i NOTE-OFF non siano ancora arrivati in quanto mai inviati dal *controller* a monte (in tal caso, il prolungamento delle note attive sarebbe corretto), oppure se i NOTE-OFF siano stati inviati ma non arriveranno mai (e quindi il

[15] La maggior parte dei *master* consente un intervallo di 1 ms tra START e i successivi messaggi TIMING CLOCK per dare allo *slave* la possibilità di prepararsi alla riproduzione.

prolungamento delle note attive sia errato, con una condizione di errore che si protrae per un tempo indefinito).

L'abilitazione del meccanismo di *active sensing* permette di risolvere il problema. Il messaggio funziona infatti come un segnale di *keep-alive*. Una volta ricevuto un primo messaggio ACTIVE SENSING, il ricevitore si aspetta un nuovo messaggio MIDI (di qualsiasi tipo) entro l'intervallo di 300 ms e, in caso contrario, deduce che la connessione sia stata interrotta. In realtà le specifiche suggeriscono di tenere in considerazione i ritardi di propagazione e gestione del messaggio, raccomandando di immettere un nuovo ACTIVE SENSING dopo 270 ms dalla trasmissione dell'ultimo messaggio. Per quanto concerne il ricevitore, le specifiche suggeriscono una tolleranza di circa 10%, valutando lo stato della connessione dopo circa 330 ms.

Nel caso l'*active sensing* sia attivo ma nell'intervallo di tempo non venga rilevato alcun messaggio, il ricevitore ritiene che la trasmissione risulti interrotta, e dunque disattiva tutte le voci e torna al normale funzionamento, con *active sensing* disattivato. Nell'esempio già citato lo smarrimento dei NOTE-OFF dovuto all'involontaria interruzione delle comunicazione comporterebbe – nel caso peggiore – un ritardo di 300 ms (o 330 ms, tenendo conto delle tolleranze) nello spegnimento del suono.

La composizione del messaggio ACTIVE SENSING si riconduce, come per tutti i messaggi in tempo reale, al solo byte di stato, fissato, in questo caso, a $\texttt{11111110}_2$, o analogamente $\texttt{FE}_{16}$ (Figura 2.41).

Figura 2.41. Rappresentazione grafica del messaggio ACTIVE SENSING.

Esempio

Un esempio di scambio di messaggi tra trasmettitore e ricevitore, con *active sensing* disattivato e attivato, è mostrato rispettivamente nelle Figure 2.42 e 2.43. Il tempo di propagazione dei messaggi lungo la catena MIDI può essere considerato trascurabile, dell'ordine dei nanosecondi, considerata la tecnologia del doppino ritorto schermato e la lunghezza del cavo. Per attivare la tecnica *active sensing*, nel secondo scenario il trasmettitore non fa altro che emettere messaggi di questo tipo.

In entrambi i casi a un dato istante il canale trasmissivo si interrompe, e dunque i messaggi successivi emessi dal trasmettitore non possono giungere al ricevitore. Questa situazione, che nell'esempio in questione porta allo smarrimento di un messaggio NOTE-OFF, nel primo scenario comporta un prolungamento indefinito del suono emesso dal ricevitore; nel secondo scenario, invece, il mancato arrivo di un qualsivoglia messaggio nell'intervallo di 300 ms dalla ricezione dell'ultimo messaggio (nella fattispecie, un ACTIVE SENSING) induce il ricevitore a ritenere che si sia verificato un problema nel canale trasmissivo, gestito tramite lo spegnimento del suono una volta scaduto il *timeout*.

2.9.4 Il messaggio SYSTEM RESET

Il messaggio SYSTEM RESET, noto anche come RESET, chiede ai dispositivi in ricezione di reimpostare il sistema ai valori usati all'accensione. Le specifiche suggeriscono di non inviarlo automaticamente, soprattutto all'avvio del sistema, ma di condizionarne la trasmissione a un'azione manuale ed esplicita da parte dell'utente.

Per questo messaggio, il byte di stato è $\texttt{11111111}_2$, o analogamente $\texttt{FF}_{16}$ (Figura 2.44).

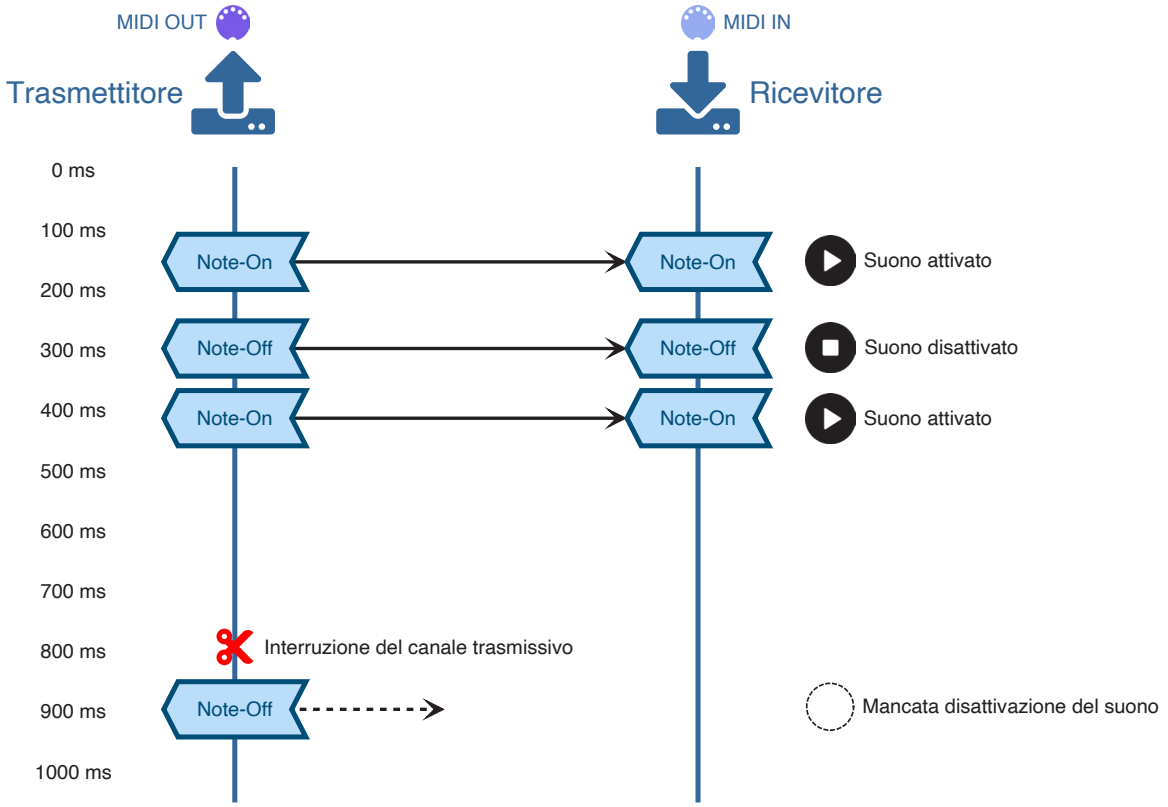


Figura 2.42. Esempio di scambio di messaggi in caso di interruzione del canale trasmissivo senza *active sensing*.

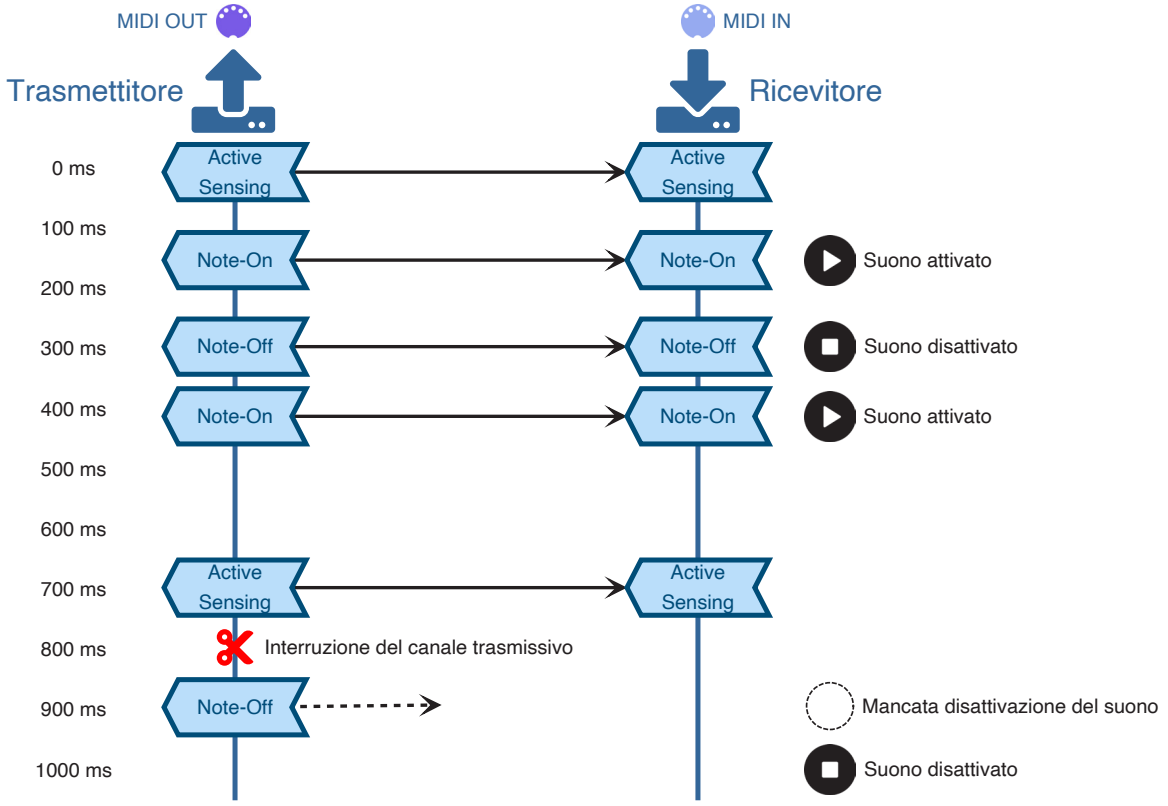


Figura 2.43. Esempio di scambio di messaggi in caso di interruzione del canale trasmissivo con *active sensing*.

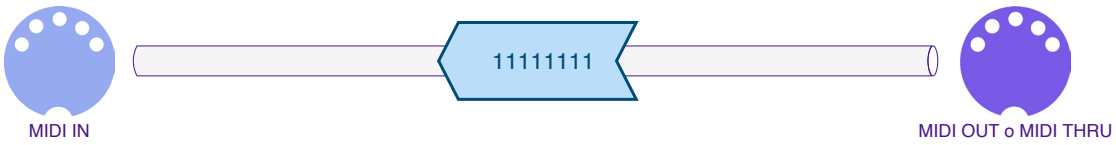


Figura 2.44. Rappresentazione grafica del messaggio SYSTEM RESET.

2.10 I messaggi *System Exclusive*

I messaggi di sistema esclusivi (*System Exclusive*) rappresentano uno stratagemma per portare un meccanismo di programmazione aperta all'interno di un protocollo completamente standardizzato. La coppia di messaggi SYSTEM EXCLUSIVE (Paragrafo 2.8.5) ed END OF EXCLUSIVE (Paragrafo 2.8.6) rappresenta un modo standard, previsto dalle specifiche MIDI, per avviare e terminare una sequenza di byte dal significato non standard, lasciato alle scelte progettuali e implementative dei singoli produttori.

Come sottolinea il nome stesso, i messaggi *System Exclusive* sono di sistema anziché di canale, quindi rivolti potenzialmente a tutti i dispositivi all'interno del *setup* MIDI, indipendentemente dal loro canale base. Ciononostante, essi includono un meccanismo multi-livello per identificare in modo univoco i destinatari, arrivando – ove richiesto – alla granularità del singolo prodotto, inteso come un dato modello realizzato da un determinato costruttore.

Le funzioni di questa famiglia di messaggi sono le più disparate, dall'abilitazione o disabilitazione di una particolare modalità di funzionamento (ad esempio, il General MIDI) all'impostazione di specifici valori per i parametri di sintesi.

Uno degli usi più comuni riguarda il cosiddetto *bulk dump*. In tutte le periferiche programmabili le impostazioni vengono salvate in forma numerica nella memoria del dispositivo. Normalmente le periferiche MIDI permettono un *dump* (scaricamento) di tutte o alcune delle impostazioni contenute nella loro memoria sotto forma di messaggi SYSEX. Un *dump*, quindi, può essere un modo per eseguire copie di backup delle impostazioni del proprio strumento MIDI; inviando il dump alla periferica MIDI, se ne ripristinano le impostazioni. Un uso alternativo di un dump è la sua memorizzazione all'interno di un *sequencer*, in modo da poter modificare lo stato di un dispositivo in modo istantaneo.

All'interno della famiglia *System Exclusive* si riconoscono due tipologie di messaggi: i *SysEx del produttore*, sotto il totale controllo del costruttore del dispositivo (Paragrafo 2.10.1), e i *SysEx universali*, concordati tra i vari costruttori (Paragrafo 2.10.2).

2.10.1 Messaggi *SysEx* del produttore

Il meccanismo dei *SysEx* del produttore consente ai costruttori di definire nuovi messaggi fortemente orientati alle caratteristiche dei propri dispositivi: parametri per le *patch*, dati relativi ai campioni, salvataggio e ripristino della configurazione di un *sequencer* o di uno strumento MIDI, e via dicendo.

Il codice identificativo dei produttori (*manufacturer's ID code*, o semplicemente *manufacturer ID*) viene attualmente assegnato da MMA o AMEI [16], ed è rappresentato dal primo byte di dati o dai primi tre byte di dati in un messaggio SYSTEM EXCLUSIVE. Nel secondo caso i codici estesi hanno come primo byte 00_{16}, mentre l'identificativo vero e proprio occupa i successivi due byte. Si ricorda che tali valori vengono comunque veicolati come byte di dati, quindi solo 7 bit per byte possono essere effettivamente destinati alla rappresentazione del codice identificativo.

L'elenco dei produttori viene periodicamente aggiornato, in modo che nuovi attori possano richiedere un proprio codice per aderire al protocollo. Tra le molte aziende coinvolte [17], si

[16] In origine, si trattava di un accordo tra MMA e JMSC.

[17] https://www.midi.org/specifications-old/item/manufacturer-id-numbers

segnalano

- per l'area nordamericana: Sequential Circuits (01_{16}), Moog Music (04_{16}), Fender (08_{16}), Apple (11_{16}), Steinberg Media Technologies GmbH ($3A_{16}$), Time/Warner Interactive (00 00 01_{16}), IBM (00 00 $3A_{16}$), Microsoft (00 00 41_{16}), Atari Corporation (00 00 58_{16}), AT&T Bell Laboratories (00 00 $5C_{16}$), Nvidia (00 00 $7A_{16}$), Google (00 02 $0D_{16}$), Universal Audio (00 02 18_{16}), Steinway (00 02 34_{16});

- per il Giappone: Kawai Musical Instruments MFG. Co. Ltd (40_{16}), Roland Corporation (41_{16}), Korg Inc. (42_{16}), Yamaha Corporation (43_{16}), Casio Computer Co. Ltd (44_{16}), Sony Corporation ($4E_{16}$);

- infine, per l'Europa e il resto del mondo: Bontempi SpA (00 20 $0B_{16}$), Marshall Products Limited (00 20 16_{16}), LG Electronics (00 20 23_{16}), Behringer (00 20 32_{16}), IRCAM (00 20 39_{16}), Nokia (00 20 $4E_{16}$), Dolby Australia (00 20 $5A_{16}$), Native Instruments (00 2H 09_{16}), Philips Electronics HK Ltd (00 21 $0F_{16}$), ROLI Ltd (00 21 10_{16}), Ableton (00 21 $1D_{16}$).

Dopo il byte per il *manufacturer ID*, due byte di dati vengono riservati all'identificazione del dispositivo (*device ID*) e del modello (*model ID*). Solo se il dispositivo in ricezione riconosce il messaggio SysEx come rivolto a se stesso lo prende in considerazione, in altro caso lo ignora. La sintassi su due byte permetterebbe di distinguere solo tra 128 tipologie di dispositivo e 128 modelli di dispositivo per ciascun produttore, ma alcuni costruttori adottano dei valori standard: ad esempio, Roland assegna a tutti i propri dispositivi nel *setup* il valore *device ID* = 10_{16}, e a tutti i propri *synth* dotati di standard GS il valore *model ID* = 42_{16}.

Va detto che, trattandosi di messaggi sotto il pieno controllo del costruttore, non è particolarmente sensato cercare uno standard dalla valenza generale, quanto meno a valle del byte – o dei byte – che identificano il produttore. A puro titolo d'esempio si riportano alcuni SysEx tratti dal manuale utente del modello *Yamaha CP33*. Si tratta di uno specifico dispositivo, per cui sarebbe lecito aspettarsi, immediatamente dopo il byte di identificazione di Yamaha (43_{16}), gli stessi valori di *device ID* e *model ID* per tutti i messaggi SysEx supportati. Invece:

- per il messaggio "Special Control", *device ID* = 73_{16} (identifica il "Product ID", ossia CP33) e *model ID* = $7F_{16}$ (identifica l'"Extended Product ID");

- per il messaggio "CP33 MIDI Format", *device ID* = 73_{16}, ma *model ID* = 01_{16} (valore comune a tutta la serie CLP);

- per il messaggio "Master Setting Bulk", all'identificativo del modello sono dedicati due byte dal valore $7F_{16}$ (ordine alto) e 05_{16} (ordine basso), preceduti da un byte che denota il messaggio come *bulk request*, presumibilmente per mantenere la compatibilità con altri dispositivi Yamaha in cui è ammessa non solo la ricezione ma anche l'invio di una *bulk request*.

Se i byte teoricamente dedicati all'identificazione del dispositivo e del modello mostrano una situazione variegata, a maggior ragione la parte restante di un messaggio SysEx non può essere trattata in modo generale. I byte che seguono variano per numero e per semantica, e solo il riferimento alla documentazione tecnica del modello può venire in aiuto.

Talvolta i messaggi codificano in un'apposita sezione la lunghezza stessa del SysEx, per quanto si tratti di un'informazione ridondante: essa sarebbe determinabile a posteriori grazie alla ricezione del messaggio EOX.

Poiché la comunicazione esclusiva può implicare lunghe sequenze di byte e il canale trasmissivo non è affidabile, alle volte i messaggi SysEx dedicano uno o più byte alla cosiddetta somma di controllo (*checksum*), come nel caso dei prodotti Roland.

Esempio

Come esempio di *SysEx* universale in tempo reale, si analizza il messaggio per selezionare il tipo di riverbero in un sintetizzatore *Roland JV-1080*.

Il messaggio, in formato esadecimale, si presenta come

```
F0 41 10 6A 12 01 00 00 28 06 51 F7
```

e la spiegazione dei singoli byte è la seguente:

- $\texttt{F0}_{16}$ rappresenta il byte di stato del messaggio SYSTEM EXCLUSIVE (SYSEX);
- $\texttt{41}_{16}$ costituisce il *manufacturer ID* di Roland;
- $\texttt{10}_{16}$ rappresenta il valore di *device ID* per qualsiasi strumento Roland;
- $\texttt{6A}_{16}$ rappresenta il valore di *model ID* per il modello *JV-1080*;
- $\texttt{11}_{16}$ o $\texttt{12}_{16}$ riguardano l'identificativo del comando *Send* o *Dump*, posto a $\texttt{11}_{16}$ per la richiesta e a $\texttt{12}_{16}$ per l'invio;
- $\texttt{01 00 00}_{16}$ forniscono l'indirizzo del parametro di riverbero all'interno della memoria del dispositivo;
- $\texttt{28 06}_{16}$ costituiscono, rispettivamente, il tipo di riverbero (28_{16} identifica il delay) e il valore da impostare per esso;
- $\texttt{51}_{16}$ rappresenta la somma di controllo Roland, calcolata secondo l'algoritmo sotto illustrato;
- $\texttt{F7}_{16}$ rappresenta il byte di stato del messaggio END OF EXCLUSIVE (EOX).

La somma di controllo Roland prende in considerazione i byte che seguono l'informazione di *Send*/*Dump* e precedono il byte corrente. L'algoritmo prevede, innanzi tutto, di effettuare la somma byte per byte, ottenendo così il valore s, poi di calcolare il resto r della divisione intera di s per 80_{16}, e infine di computare il *checksum* come $c = 80_{16} - r$. Nel caso in esame: $s = (01 + 00 + 00 + 28 + 06)_{16} = 2F_{16}$; il resto della divisione intera di $2F_{16}$ per 80_{16} è $r = 2F_{16}$; infine, risulta $c = (80 - 2F)_{16} = 51_{16}$.

2.10.2 Messaggi *SysEx* universali

Considerando il primo byte di dati di un messaggio SYSTEM EXCLUSIVE, le specifiche MIDI riservano le due combinazioni 126 ($7E_{16}$) e 127 ($7F_{16}$) rispettivamente ai messaggi *SysEx* universali non in tempo reale e in tempo reale. Tali identificativi non sono specifici per un dato produttore, bensì universali; in altri termini, tutti i dispositivi MIDI sono potenziali destinatari di questi messaggi.

Apparentemente, la definizione di messaggi esclusivi (specifici al singolo modello) ma universali (rivolti a tutti i dispositivi, addirittura in modo indipendente dal canale base) potrebbe sembrare contraddittoria. In realtà questa categoria trova giustificazione nelle esigenze emerse a valle della standardizzazione MIDI 1.0, quando i produttori hanno convenuto sull'opportunità di introdurre tipologie di scambio di dati originariamente non previste: anziché rimettere mano allo standard si è preferito appoggiarsi a un meccanismo già disponibile.

Questa categoria di messaggi prevede una strutturazione ricorrente. Dopo il byte di stato SYSEX ($F0_{16}$) e il primo byte di dati, che determina la risposta in tempo reale ($7F_{16}$) o meno ($7E_{16}$), un ulteriore byte è riservato al cosiddetto *System Exclusive channel*. Sebbene i messaggi non siano di canale, inteso come canale MIDI, è data la possibilità di indicare un valore di canale *SysEx* nell'intervallo [0, 127], ossia [00_{16}, $7F_{16}$], che generalmente coincide con l'identificativo del produttore (ad esempio, 41_{16} per Roland). Questo consente di vincolare la ricezione di un messaggio universale a un dispositivo MIDI specifico. Il valore di canale 127, o $7F_{16}$, indica che il messaggio si rivolge a tutti i dispositivi MIDI. Seguono due byte che sono detti *Sub-ID #1* e *Sub-ID #2*, che determinano la funzione svolta dal *SysEx* a due livelli di definizione, uno più generico e l'altro più specifico. Seguono, infine, i byte di dati che codificano il valore da trasmettere per quella data funzione. In alcuni casi è sufficiente il livello di dettaglio offerto da *Sub-ID #2* per

veicolare l'intero contenuto informativo, dunque non vengono inviati ulteriori byte di dati. La comunicazione viene conclusa dal messaggio END OF EXCLUSIVE.

L'elenco completo dei *SysEx* universali è disponibile all'URL https://www.midi.org/specifications-old/item/table-4-universal-system-exclusive-messages. Nel seguito verranno analizzati alcuni messaggi a scopo esemplificativo.

La struttura di un generico messaggio *SysEx* universale è mostrata nella Figura 2.45.

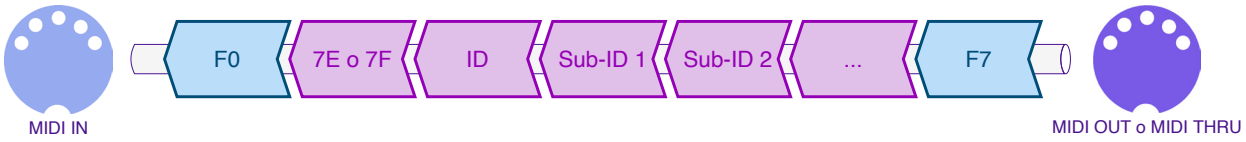


Figura 2.45. Rappresentazione grafica di un generico messaggio *SysEx* universale, non in tempo reale o in tempo reale a seconda del valore assunto dal primo byte di dati.

Esempio 1: MIDI Master Volume

Un esempio di messaggio *SysEx* universale in tempo reale è dato da *MIDI Master Volume*, il cui scopo è modificare il volume di tutti i canali. Il messaggio, in formato esadecimale, si presenta come

```
F0 7F xx 04 01    mm F7
```

e la spiegazione dei singoli byte è la seguente:

- $\texttt{F0}_{16}$ rappresenta il byte di stato del messaggio SYSTEM EXCLUSIVE (SYSEX);
- $\texttt{7F}_{16}$ identifica un messaggio universale in tempo reale;
- $\texttt{xx}_{16}$ (o $\texttt{7F}_{16}$) fornisce il *Device ID* del dispositivo destinatario. Il valore $\texttt{7F}_{16}$ richiede a tutti i dispositivi di rispondere, mentre $\texttt{xx}_{16}$ rappresenta il canale del *SysEx*, normalmente l'identificativo del costruttore (*manufacturer ID*);
- $\texttt{04}_{16}$ rappresenta il *Sub-ID #1*, il cui valore, in questo caso, indica *Device Control Message*;
- $\texttt{01}_{16}$ rappresenta il *Sub-ID #2*, il cui valore identifica *Master Volume*;
- $\texttt{ll}_{16}$ costituisce l'LSB per il nuovo valore di volume;
- $\texttt{mm}_{16}$ costituisce l'MSB per il nuovo valore di volume;
- $\texttt{F7}_{16}$ rappresenta il byte di stato del messaggio END OF EXCLUSIVE (EOX).

I byte $\texttt{ll}_{16}$ e $\texttt{mm}_{16}$ formano insieme un valore espresso su 14 bit; dunque il volume può variare tra 0000_{16} (normalmente, volume nullo) e $3FFF_{16}$ (volume massimo), con $1FFF_{16}$ come valore medio. Se, ad esempio, $\texttt{ll}_{16} = 7F_{16} = 01111111_2$ e $\texttt{mm}_{16} = 3F_{16} = 00111111_2$, il valore binario ricostruito è $11111111111111_2 = FFF_{16}$.

Esempio 2: Single-note Tuning Change

Come ulteriore esempio di *SysEx* universale in tempo reale, si affronta ora il messaggio *Single-note Tuning Change*, che permette di impostare con grande precisione l'intonazione delle note prodotte dai moduli sonori in ricezione: si tratta della cosiddetta micro-intonazione (*microtuning*). La famiglia di messaggi dedicati al *MIDI Tuning* include, oltre a *Single-note Tuning Change*, anche *Bulk Tuning Dump Request* e *Bulk Tuning Dump*, anch'essi messaggi *SysEx* ma non in tempo reale.

Il protocollo MIDI associa ai pitch i valori di frequenza stabiliti dallo standard ISO 16, che fissa la frequenza del *la* centrale al valore di 440 Hz e calcola la frequenza delle restanti note a partire da tale riferimento. In MIDI il *la* centrale è associato al pitch 69.

Il protocollo permette di modificare l'intonazione delle note attraverso un apposito messaggio, PITCH BEND CHANGE (Paragrafo 2.4.9), che però agisce sull'intero canale e di solito implica

una modifica temporanea dell'intonazione, essendo normalmente associato a un controllo con posizione di richiamo centrale. La modifica non temporanea della microintonazione può essere attuata attraverso il *fine* o il *coarse tuning* menzionati nel Paragrafo 2.5.4, ma l'effetto si riverbera su tutte le note del canale.

Il messaggio *Single-note Tuning Change*, invece, assegna in modo definitivo [18] una nuova frequenza a un dato pitch all'interno di uno specifico *program number*. La raccomandazione da parte delle specifiche è di rispondere in modo immediato al messaggio, che difatti è in tempo reale, ma senza introdurre artefatti per le note già in esecuzione.

Il messaggio, in formato esadecimale, si presenta come

```
F0 7F xx 08 02 tt ll [kk xx yy zz] F7
```

e la spiegazione dei singoli byte è la seguente:

- $\texttt{F0}_{16}$ rappresenta il byte di stato del messaggio SYSTEM EXCLUSIVE (SYSEX);
- $\texttt{7F}_{16}$ identifica un messaggio universale in tempo reale;
- $\texttt{xx}_{16}$ (o $\texttt{7F}_{16}$) fornisce il *Device ID* del dispositivo destinatario. Il valore $\texttt{7F}_{16}$ richiede a tutti i dispositivi di rispondere, mentre $\texttt{xx}_{16}$ rappresenta il canale del *SysEx*, normalmente l'identificativo del costruttore (*manufacturer ID*);
- $\texttt{08}_{16}$ costituisce il valore di *Sub-ID #1* che identifica la funzione *MIDI Tuning Standard*;
- $\texttt{02}_{16}$ costituisce il valore di *Sub-ID #2* che indica *Note change*;
- $\texttt{tt}_{16}$ fornisce il *program number* nell'intervallo [0, 127] coinvolto dall'operazione di microintonazione;
- $\texttt{ll}_{16}$ rappresenta il numero di cambiamenti di intonazione veicolati dal messaggio. Per ridurre il numero di byte trasmessi è possibile variare l'intonazione di più note all'interno di un solo SYSEX. Nel caso di $\texttt{ll} = \texttt{01}_{16}$, il messaggio descriverà il cambio di intonazione per una singola nota, e dunque conterrà, a seguire, un unico quartetto di byte `[kk xx yy zz]`;
- $\texttt{kk}_{16}$ indica il pitch MIDI di cui ridefinire la frequenza;
- `xx yy zz` complessivamente rappresentano il nuovo valore di frequenza. Il primo byte, `xx`, determina il semitono immediatamente inferiore alla frequenza desiderata nel sistema temperato, ossia secondo lo standard ISO 16; i successivi due byte, `yy zz`, specificano su 14 bit la frazione di 100 cents al di sopra del semitono precedentemente identificato;
- $\texttt{F7}_{16}$ è il byte di stato del messaggio END OF EXCLUSIVE (EOX).

Il blocco `[kk xx yy zz]` viene ripetuto per il numero di volte specificato da `ll`, e dunque la lunghezza effettiva del messaggio può essere calcolata come $8 + (\texttt{ll}_{10} \times 4)$ byte.

Tra i valori notevoli per `xx yy zz`, si segnalano `3C 00 00`, pari alla frequenza standard del *do* centrale (261,6256 Hz), e `45 00 00`, che rappresenta la frequenza del *la* dell'ottava centrale (440 Hz). Ad esempio, il valore `45 00 01` denota la frequenza 440,0016 Hz, il che permette di apprezzare la risoluzione frequenziale offerta dal meccanismo.

Si osservi che questo valore non costituisce necessariamente la micro-intonazione del *la* centrale, il che dipende dal precedente valore di `kk`: solo se posto a $69 = 45_{16}$, esso identifica effettivamente il *la* centrale. Il valore massimo utilizzabile per esprimere una frequenza è `7F 7F 7E`, che corrisponde a 13289,73 Hz, mentre `7F 7F 7F` è un valore riservato all'informazione "nessun cambiamento". Questo valore assume significato in particolare quando uno strumento che non supporta l'intervallo completo dei 128 pitch comunica dati di intonazione a un altro che, potenzialmente, li gestisce; l'invio di `7F 7F 7F` permette di non modificare quanto già presente nel ricevitore, prevenendo l'inserimento di valori errati.

[18] Con il termine "definitivo" si intende fino all'eventuale ricezione di un altro messaggio che sovrascriva l'effetto del precedente o fino al ripristino dello stato di accensione del dispositivo.

Esempio 3: General MIDI Mode On/Off

Come esempio di *SysEx* universale non in tempo reale, si analizza il messaggio *General MIDI Mode On*, il cui scopo è attivare la modalità General MIDI (Paragrafo 4.1) nei dispositivi compatibili. In formato esadecimale, esso si presenta come

```
F0 7E xx 09 01 F7
```

e la spiegazione dei singoli è la seguente:

- $\texttt{F0}_{16}$ rappresenta il byte di stato del messaggio SYSTEM EXCLUSIVE (SYSEX);
- $\texttt{7E}_{16}$ identifica un messaggio universale non in tempo reale. Si osservi l'uso di $\texttt{7E}_{16}$ in luogo di $\texttt{7F}_{16}$;
- $\texttt{xx}_{16}$ (o $\texttt{7F}_{16}$) fornisce il *Device ID* del dispositivo destinatario. Il valore $\texttt{7F}_{16}$ richiede a tutti i dispositivi di rispondere, mentre $\texttt{xx}_{16}$ rappresenta il canale del *SysEx*, normalmente l'identificativo del costruttore (*manufacturer ID*);
- $\texttt{09}_{16}$ costituisce il valore di *Sub-ID #1* che identifica la funzione *General MIDI Message*;
- $\texttt{01}_{16}$ costituisce il valore di *Sub-ID #2* che corrisponde a *General MIDI On*;
- $\texttt{F7}_{16}$ rappresenta il byte di stato del messaggio END OF EXCLUSIVE (EOX).

Allo standard General MIDI rilasciato nel 1991 si è affiancato nel 1999 il General MIDI Level 2; per questo motivo, rispetto a quanto riportato in molti manuali dell'epoca, ora *Sub-ID #2* se posto a $\texttt{01}_{16}$ identifica la funzione "General MIDI 1 System On" anziché genericamente "General MIDI On", mentre, se fissato a $\texttt{03}_{16}$, indica "General MIDI 2 System On". Per lasciare la modalità General MIDI, il valore di *Sub-ID #2* va impostato a $\texttt{02}_{16}$.

2.11 Running status

Il ***running status*** (letteralmente: stato corrente) è un'alternativa all'usuale formato di trasmissione dei dati MIDI che si applica solo a messaggi consecutivi dello stesso tipo e sullo stesso canale. La finalità è minimizzare la quantità di dati inviati eliminando informazione ridondante, il che permette di risparmiare tempo nell'invio e nella ricezione dei messaggi e, contemporaneamente, di ridurre l'occupazione del canale trasmissivo, diminuendo i fenomeni di ritardo dovuto al traffico.

L'idea che sta alla base del *running status* è la possibilità di non reinviare un byte di stato perfettamente identico a quello del messaggio precedentemente inviato. Il meccanismo si applica solo a messaggi di canale (famiglie *Channel Voice* e *Channel Mode*), il cui byte di stato rientra dunque nell'intervallo $[\texttt{80}_{16}, \texttt{EF}_{16}]$. Si osservi che la regola del byte di stato identico, oltre a implicare il richiamo della stessa funzione MIDI (dal secondo al quarto bit), richiede anche l'uso dello stesso canale (dal quinto all'ottavo bit).

I messaggi delle famiglie *System Common* e *System Exclusive* interrompono un eventuale *running status* e, a loro volta, non possono essere soggetti ad esso. Al contrario, i messaggi System Real Time, potendosi interfogliare a qualsiasi sequenza di byte, non comportano un'interruzione del *running status*: quando il ricevitore li rileva, di fatto li estrae dalla restante sequenza di byte. Questo comportamento, già descritto in un contesto senza *running status*, si applica anche al caso in esame. Pertanto l'invio e la ricezione di messaggi in tempo reale non va ad interrompere lo stato corrente. Va osservato che non si può applicare il *running status* ai messaggi in tempo reale: essi sono costituiti dal solo byte di stato, per cui presentano già la forma più compatta possibile. Ricevere due messaggi in tempo reale consecutivi e identici ed eliminare il secondo non comporterebbe una riduzione della ridondanza, bensì una perdita di informazione. Si sottolinea come la trasmissione *running status* venga interrotta anche da messaggi relativi alle voci che abbiano lo stesso identificativo (ad esempio, due NOTE-ON consecutivi) ma si trovino su canali differenti: infatti, in un caso, simile il byte di stato sarebbe differente.

Con il *running status* attivo, dopo un primo messaggio completo, i successivi messaggi sono costituiti dai soli byte di dati. Il ricevitore, quando rileva per un nuovo messaggio un byte di dati

in luogo di un byte di stato, assume che questo si riferisca allo stato di un precedente messaggio. Il processo richiede che il ricevitore conservi in una memoria tampone, detta *running status buffer*, l'ultimo byte di stato riconosciuto, che conosca il numero di byte di cui si compone normalmente quel dato messaggio, e che – conseguentemente – sappia riconoscere il primo byte del messaggio successivo. Se si tratta di un byte di dati, il dispositivo assumerà che il byte di stato mancante sia quello conservato nel buffer, e procederà di conseguenza (ad esempio, attendendo l'arrivo di tutti i suoi byte di dati, per poi rivalutare l'ulteriore prosieguo del *running status*); in caso contrario riterrà la trasmissione *running status* (temporaneamente) conclusa, salvando nel buffer il nuovo byte di stato se il messaggio è di canale.

Per comprendere i potenziali vantaggi del *running status*, si rammenti che una nota viene normalmente accesa inviando un messaggio di NOTE-ON composto da 3 byte: $\texttt{1001nnnn}_2$ seguito da $\texttt{0kkkkkkk}_2$ per il pitch e $\texttt{0vvvvvvv}_2$ per la velocity. Grazie al *running status*, per tutte le successive note accese sullo stesso canale è possibile inviare i soli byte di dati relativi a pitch e velocity. Si osservi che questa condizione viene rispettata anche nella trasmissione di un accordo, in cui i NOTE-ON (teoricamente) simultanei vengono comunque immessi serialmente sul canale trasmissivo.

Ad esempio, la sequenza di messaggi relativa all'accensione dell'accordo *do-mi-sol* nell'ottava centrale (pitch 60, 64, 67) sul primo canale implicherebbe l'invio dei seguenti byte:

```
90 3C 7F
90 40 7F
90 43 7F
```

Con il *running status* la sequenza si riduce a:

```
90 3C 7F
   40 7F
   43 7F
```

Inoltre la tecnica *running status* rende evidente il vantaggio di codificare NOTE-OFF (8n_{16}) come NOTE-ON (9n_{16}) con velocity pari a 0. Infatti, purché il *running status* non venga interrotto da messaggi di altre famiglie o da messaggi su altri canali, lunghe sequenze di accensioni e spegnimenti di note sullo stesso canale comportano l'invio di un unico byte di stato.

Ad esempio, limitandosi all'accensione e allo spegnimento di due accordi triadici, dunque composti da 3 note ciascuno, la trasmissione di messaggi senza *running status* implicherebbe l'invio di 3 NOTE-ON e 3 NOTE-OFF per ciascun accordo, e dunque un traffico dati di 36 byte. Con il *running status* e l'uso di soli NOTE-ON, la trasmissione di un messaggio completo viene seguita dai soli byte di dati dei messaggi seguenti, e quindi il traffico si riduce a $3 + 11 \times 2 = 25$ byte.

Vantaggi ancora più rilevanti si hanno per quei messaggi che devono inviare con frequenza l'aggiornamento di un valore: CHANNEL PRESSURE, POLYPHONIC KEY PRESSURE, PITCH BEND CHANGE e alcuni CONTROL CHANGE.

Un trasmettitore può essere progettato per trarre vantaggio o meno dal *running status*. Al contrario, un ricevitore dovrebbe essere in grado di gestire sia la comunicazione normale di messaggi, sia quella con *running status*. La tecnica assume particolare importanza su canali trasmissivi lenti, di cui lo standard MIDI a 31,25 Kbit/s è certamente un esempio; infatti, essa non viene adottata in protocolli di trasporto più veloci, quali MIDI-over-OSC e MIDI-over-USB.

Capitolo 3

Altri protocolli MIDI

In questo capitolo verranno trattati alcuni protocolli secondari introdotti dalle specifiche MIDI 1.0 e relativi al controllo via MIDI di apparecchiature non direttamente collegate alla produzione di suono.

Si tratta di estensioni apportate al protocollo originario grazie a messaggi SYSTEM EXCLUSIVE in tempo reale, in modo da mantenere piena compatibilità con i dispositivi MIDI che non sono in grado di supportarle.

3.1 MIDI Machine Control

MIDI Machine Control (MMC) è un protocollo concepito per consentire ai dispositivi MIDI di comunicare con sistemi di registrazione e riproduzione tradizionali e di controllarli via MIDI. Questo significa che tali dispositivi devono essere in grado di ricevere messaggi MIDI, e dunque presentare quanto meno un connettore MIDI IN.

Per rendere l'implementazione del protocollo più diretta e agevole, le specifiche MMC si rifanno all'insieme di comandi (*commands*) e risposte (*responses*) elencati nella sezione "Audio Tape Recorder" dello standard ESbus. A questo riguardo vengono ripresi i soli principi di funzionamento dei dispositivi di registrazione e riproduzione, mentre si riconosce l'esigenza di apportare modifiche sostanziali sulle modalità di comunicazione e sulla codifica dei dati, viste le differenze tra il dominio applicativo MMC e l'universo MIDI.

Riassumendo, il protocollo MMC è pensato per comunicare agevolmente con dispositivi progettati per eseguire lo stesso insieme di operazioni definite nello standard ESbus.

3.1.1 Formato dei messaggi MMC

Il protocollo MMC richiede che i messaggi vengano inviati su cavo MIDI, anche se implementazioni più recenti possono utilizzare standard quali USB grazie ad opportune interfacce di conversione da MIDI a USB. Le finalità di tali messaggi sono il controllo da remoto e la sincronizzazione di una performance MIDI tramite funzioni quali *Play*, *Fast Forward*, *Rewind*, *Stop*, *Pause* e *Record*. A tale scopo vengono utilizzati messaggi della famiglia *System Exclusive*, e in particolare messaggi universali in tempo reale (*Real Time Universal SysEx*).

Esistono due tipologie di messaggi in uso nel protocollo:

- *mcc* (Machine Control Command), per la trasmissione dal dispositivo controllore a quello controllato;
- *mcr* (Machine Control Response), per la trasmissione dal dispositivo controllato a quello controllore.

byte	comando
01_{16}	Stop
02_{16}	Play
03_{16}	Deferred Play
04_{16}	Fast Forward
05_{16}	Rewind
06_{16}	Record Strobe (Punch In)
07_{16}	Record Exit (Punch out)
08_{16}	Record Ready
09_{16}	Pause
$0A_{16}$	Eject
$0B_{16}$	Chase
$0F_{16}$	MMC Reset

Tabella 3.1. Alcuni comandi MMC.

Queste due categorie vengono caratterizzate all'interno del messaggio SysEx attraverso il campo *Sub-ID #1*, posto a 06_{16} per *mcc* e a 07_{16} per *mcr*. Il formato dei messaggi SysEx in esadecimale risulta pertanto:

```
F0 7F <device_ID> 06 <commands...> F7
F0 7F <device_ID> 07 <responses...> F7
```

Riguardo ai comandi, le specifiche elencano 37 valori ammissibili; i primi 12 di questi vengono riportati a scopo esemplificativo nella Tabella 3.1. Tra i messaggi non menzionati nell'elenco assume particolare importanza il comando Goto, che permette di avviare la registrazione o la riproduzione a un determinato istante temporale specificato tramite SMPTE (ore, minuti, secondi, frame e sub-frame). Si tratta di un comando (quarto byte posto a 06_{16}) identificato dal quinto byte posto a 44_{16}. La sua sintassi in forma esadecimale è la seguente:

```
F0 7F <device_ID> 06 44 06 01 <hr> <mn> <sc> <fr> <ff> F7
```

Alcuni messaggi previsti da MMC sembrerebbero sovrapporsi con altri presenti nelle specifiche MIDI originarie, e in particolare Start, Stop, Continue (Paragrafo 2.9.2) e Song Position Pointer (Paragrafo 2.8.3). Si osservi, però, che tali messaggi sono destinati a controllare un *sequencer*, ossia un dispositivo in cui i dati di temporizzazione sono espressi attraverso la logica delle pulsazioni e dei *MIDI clocks*. Basti pensare a come avviene il riposizionamento del puntatore alla sequenza attraverso Song Position Pointer: l'informazione veicolata è il numero di *MIDI clocks* trascorsi tra l'inizio del brano e il punto desiderato, e non un istante temporale in formato SMPTE.

Per quanto riguarda le risposte, le specifiche MMC contengono 70 codici numerici. Una trattazione dettagliata dei singoli messaggi esula dagli scopi del presente manuale; si rimanda al documento tecnico, che è parte integrante delle specifiche MIDI 1.0 del 1996, per eventuali approfondimenti.

Va sottolineato che un singolo messaggio SysEx può veicolare più comandi MMC. La parte relativa ai comandi (e alle risposte) non deve però superare la dimensione di 48 byte.

3.1.2 *Open loop, closed loop* e *handshaking*

MMC è stato ideato per operare in due differenti contesti, detti *open loop* (a ciclo aperto) e *closed loop* (a ciclo chiuso). Nei sistemi *open loop*, un singolo cavo collega il connettore MIDI OUT del dispositivo controllore con il MIDI IN del dispositivo controllato. Nei sistemi *closed loop*, invece, è richiesto un cavo aggiuntivo che connette il MIDI OUT del dispositivo controllato con il MIDI IN del controllore, consentendo una comunicazione di tipo *full duplex*. All'accensione, il dispositivo controllore si aspetta una situazione di *closed loop*. Se, una volta inviato un comando, entro 2 secondi non riceve una risposta, allora tale dispositivo assume che il *setup* sia *open loop*. Eventuali

variazioni nello stato di apertura o chiusura possono essere determinate dal controllore inviando periodicamente una richiesta che preveda una qualsiasi forma di risposta.

Il flusso di dati è controllato da due messaggi, necessari a effettuare il cosiddetto *handshaking*: `WAIT`, che indica una sospensione temporanea della trasmissione, e `RESUME`, che ne richiede la ripresa. Questi due messaggi possono essere inviati sia dal dispositivo controllore (in tal caso, `WAIT` indica che il buffer si sta riempiendo e `RESUME` che l'invio è pronto), sia dai dispositivi controllati (in tal caso, `WAIT` indica che il buffer in ricezione sta venendo predisposto per leggere nuovi dati e `RESUME` che il dispositivo è pronto per riceverli). Nel primo scenario, il destinatario del messaggio è *all-call*, ossia qualsiasi dispositivo controllato, il che viene indicato dal valore $7F_{16}$ per `device_ID`; nel secondo scenario, invece, `device_ID` identifica il dispositivo controllato che chiede al controllore di mettere in pausa o riprendere la trasmissione.

3.1.3 Identificazione dei dispositivi

Un campo fondamentale nella sintassi dei messaggi MMC sopra mostrata è `<device_ID>`. Nel *setup* è possibile associare a ogni dispositivo un valore numerico compreso tra 0 e 127. All'interno dei messaggi, il campo `<device_ID>` assume il significato di indirizzo origine o destinazione a seconda del contesto: è la destinazione se il messaggio è un comando, è l'origine se si tratta di una risposta.

Spesso i comandi vengono indirizzati a un dispositivo per volta, ma è possibile definire identificativi per interi gruppi di dispositivi, così come effettuare *broadcasting* attraverso il valore riservato di *all-call*, ossia $7F_{16}$. Al contrario, le risposte giungono sempre da un singolo dispositivo.

Un *master* può interrogare uno *slave* richiedendone l'identificativo. Il messaggio da inviare a tale scopo, detto *Identity Request*, ha la seguente rappresentazione esadecimale:

```
F0 7E <device_ID> 06 01 F7
```

Si presti attenzione al fatto che il secondo byte è $7E_{16}$ anziché $7F_{16}$, come invece nei messaggi MMC sopra elencati. Alla luce di questa osservazione, i byte 4 e 5 non vanno interpretati come comando di Stop, bensì secondo la sintassi generale illustrata nel Paragrafo 2.10.2 riguardo ai SysEx universali non in tempo reale, ossia come *Sub-ID #1* posto a 06_{16} – General Information e *Sub-ID #2* a 01_{16} – Identity Request.

La risposta inviata dal dispositivo controllato contiene informazioni su costruttore e modello. Ad esempio, un registratore digitale Fostex D-160 risponderà con il seguente SysEx:

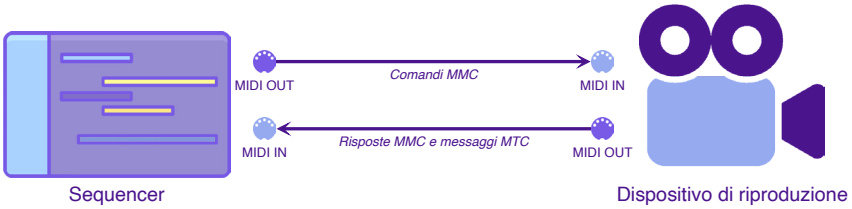
```
F0 7E <device_ID> 06 02 mm ff ff dd dd ss ss ss ss F7
```

Si nota innanzitutto la presenza di *Sub-ID #1* posto a 06_{16} – General Information e *Sub-ID #2* a 02_{16} – Identity Reply. A seguire, mm_{16} è il codice identificativo del produttore per i messaggi SysEx, $ff\ ff_{16}$ è il codice della famiglia dei dispositivi espresso su 14 bit (LSB seguito da MSB), $dd\ dd_{16}$ è il codice del dispositivo all'intero della famiglia espresso su 14 bit (LSB seguito da MSB), e, infine, $ss\ ss\ ss\ ss_{16}$ rappresenta il livello di revisione del software.

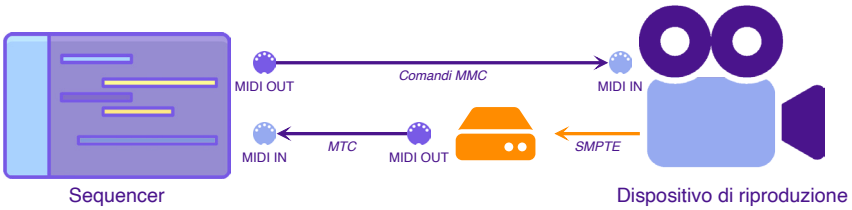
3.1.4 Esempio

Il protocollo MMC consente di controllare dispositivi di registrazione e di riproduzione a partire da una sorgente di informazioni MIDI, spesso identificabile con un *sequencer*. Si osservi che MMC non include informazioni di sincronizzazione, anche se tali informazioni possono essere inviate al dispositivo controllato, così come quest'ultimo può inviarle al controllore attraverso un'interfaccia che converte il segnale SMPTE in MTC (si veda il Paragrafo 2.8.1).

Uno scenario comune è quello illustrato nella Figura 3.1, che vede coinvolto un *sequencer* nel ruolo di *master* e un registratore in quello di *slave*. Si osservi la configurazione *closed loop*, in cui il flusso da *master* a *slave* riguarda solo comandi MMC, mentre il flusso opposto include sia risposte MMC sia messaggi MTC (Figura 3.1a). Un esempio di *slave* con unità di conversione SMPTE/MTC interna è il registratore digitale a 8 tracce *Tascam DA-78 HR*, dotato di una porta completa con connessioni MIDI IN, MIDI OUT e MIDI THRU.



(a) *Closed loop.*



(b) *Open loop* con retroazione attraverso la conversione da SMPTE a MTC.

Figura 3.1. Collegamento di un *sequencer* a un dispositivo di riproduzione via MMC.

Se il dispositivo controllato fosse in grado di generare output SMPTE ma non messaggi MTC, la loro produzione potrebbe essere demandata a un'unità esterna. Un esempio di dispositivo in grado di svolgere tale operazione è l'interfaccia MIDI *MOTU micro express*, capace di leggere timecode SMPTE e di convertirlo in MTC. In questo scenario, però, lo *slave* non potrebbe inviare risposte MMC, quindi – nonostante la retroazione – non si potrebbe parlare di *closed loop* (Figura 3.1b).

Premendo "Play" sul *sequencer* MIDI si invia un opportuno messaggio MMC al dispositivo collegato, che dunque inizia la riproduzione. Durante questa fase il registratore genera e restituisce il MIDI Time Code (MTC) con cui il *sequencer* si sincronizza. L'effetto complessivo è l'allineamento temporale tra la riproduzione audio/video da parte del registratore e quella delle sequenze MIDI all'interno del *sequencer*, detto in gergo *chasing* (letteralmente: inseguimento). Quando si preme "Stop" sul *sequencer*, il *deck* sospende la riproduzione e contestualmente smette di inviare messaggi MTC. Non ricevendo ulteriori dati MTC, il *sequencer* esce dalla modalità di *chasing*. È anche possibile, con opportuni comandi, far avanzare o riavvolgere rapidamente il nastro in riproduzione in corrispondenza di un punto all'interno della sequenza MIDI.

3.2 MIDI Show Control

MIDI Show Control (MSC) è un protocollo per il controllo via MIDI di dispositivi quali macchine sceniche, console luci (*lighting consoles*), macchine del fumo (*smoke machines*) e, in generale, apparecchiature per il *live entertainment*.

Come per il protocollo MMC, la sorgente dei dati MIDI corrisponde spesso con un *sequencer*. I dispositivi controllati devono poter ricevere messaggi attraverso una porta MIDI. Un'ulteriore analogia riguarda la famiglia di messaggi utilizzata per veicolare informazioni MSC: si tratta di *System Exclusive* in tempo reale.

L'insieme di comandi supportato è modellato sulla base dei dispositivi di controllo delle luci, dei suoni e della multimedialità, in modo che la traduzione tra il MIDI, le specifiche MSC e i comandi per i *controller* dedicati sia relativamente semplice, essendo basato sugli stessi principi di funzionamento. Però, analogamente a MMC, anche per MSC si prevede l'esigenza di apportare modifiche sostanziali sulle modalità di comunicazione e sulla codifica dei dati, date le differenze tra il dominio applicativo MSC e l'universo MIDI.

Un concetto fondamentale è quello di *cue*, ossia un evento collegato a un'azione da svolgere in un momento specifico: una variazione o un particolare effetto di illuminazione, un effetto sonoro, un cambio di scena, ecc. Nella documentazione MSC si fa spesso riferimento ai *cue numbers*, che rappresentano azioni da scatenare alla ricezione di un comando o risultati da raggiungere in un determinato intervallo di tempo.

Un sistema MSC può essere utilizzato, ad esempio, per controllare l'accensione e lo spegnimento di luci e fumo in sincronia con la riproduzione musicale. Un altro scenario d'uso riguarda una sequenza temporizzata di eventi MIDI il cui significato non è musicale: ad esempio, associati a segnali sonori e luminosi per riprodurre fedelmente una successione di tuoni e fulmini. È anche possibile interagire con macchinari in movimento sulla scena.

Il protocollo MIDI in generale non è sicuro, come dimostrato dalla necessità di testare lo stato del canale trasmissivo attraverso messaggi di ACTIVE SENSING (si veda il Paragrafo 2.9.4). Per questo motivo il protocollo MSC trova ampia applicazione in contesti "innocui", quali il controllo di luci e macchine per il fumo, ma è sconsigliato per il controllo di dispositivi a rischio: bracci meccanici, giochi pirotecnici o apparecchiature elettriche pericolose. In realtà, nelle implementazioni più recenti, i dati MIDI sfruttano un protocollo di trasporto sicuro, facendo uso, ad esempio, di connessioni Ethernet anziché di cavi MIDI.

Il primo spettacolo a fare uso delle specifiche MSC è stato la "Magic Kingdom Parade" tenutasi presso il *Walt Disney World's Magic Kingdom* nel settembre 1991. Tra i software che supportano il protocollo, si annoverano ambienti per il trattamento in tempo reale di informazione audio quali *PureData*[1] o *Max*[2] e applicazioni specifiche per il teatro e il *live entertainment* quali *Qlab*.

3.2.1 Formato dei messaggi MSC

Il protocollo MSC prevede che i messaggi vengano inviati attraverso un cavo MIDI standard. L'obiettivo è controllare da remoto e sincronizzare con una performance MIDI funzioni quali Go, Stop e Resume, tipiche dei dispositivi per il *live entertainment*.

A tale scopo vengono utilizzati messaggi della famiglia *System Exclusive*, e in particolare un messaggio universale in tempo reale (*Real Time Universal SysEx*) con *Sub-ID #1* posto a 02_{16} per tutti i comandi. Il formato è dunque il seguente:

```
F0 7F <device_ID> 02 <command_format> <command> <data> F7
```

Il protocollo prevede che, contrariamente a MMC, un singolo SYSEX non possa trasmettere più di un comando, e che il numero totale di byte non superi 128.

Riguardo al campo `device_ID`, esso verrà illustrato nel dettaglio nel Paragrafo 3.2.3.

I valori ammessi dal campo `<command_format>`, che determina il formato dell'istruzione poi specificata dal campo `<command>`, permettono di comprendere l'eterogeneità di dispositivi potenzialmente supportati da MSC. Tali valori sono elencati nella Tabella 3.2.

I campi `<command>` e `<data>` assumono valori dipendenti dal formato del comando, e dunque contestuali alla specifica categoria di apparecchiatura cui sono destinati. Esempi di valenza generale e applicabili a più categorie includono `GO`, `STOP`, `RESUME`, `TIMED_GO`, `LOAD`, `SET`, `FIRE`, `ALL_OFF`, `RESTORE`, `RESET`, `GO_OFF`. Per una trattazione più puntuale dell'argomento si rimanda il lettore alle specifiche MSC.

3.2.2 Trasmissione dei comandi e *open loop*

Le trasmissioni hanno luogo esclusivamente nella direzione che va dal *controller* ai dispositivi controllati; non è specificata o richiesta alcuna risposta ai comandi da parte di questi ultimi verso il *controller*. L'obiettivo è ottimizzare i requisiti di larghezza di banda, il tempo di risposta e l'affidabilità del sistema in caso di problemi di comunicazione con uno o più dispositivi controllati.

[1] https://puredata.info/

[2] https://cycling74.com/products/max

Hex	command_format
00	reserved for extensions
01	Lighting (General Category)
02	Moving Lights
03	Colour Changers
04	Strobes
05	Lasers
06	Chasers
10	Sound (General Category)
11	Music
12	CD Players
13	EPROM Playback
14	Audio Tape Machines
15	Intercoms
16	Amplifiers
17	Audio Effects Devices
18	Equalisers
20	Machinery (General Category)
21	Rigging
22	Flys
23	Lifts
24	Turntables
25	Trusses
26	Robots
27	Animation
28	Floats
29	Breakaways
2A	Barges
30	Video (General Category)
31	Video Tape Machines
32	Video Cassette Machines
33	Video Disc Players
34	Video Switchers
35	Video Effects
36	Video Character Generators
37	Video Still Stores
38	Video Monitors
40	Projection (General Category)
41	Film Projectors
42	Slide Projectors
43	Video Projectors
44	Dissolvers
45	Shutter Controls
50	Process Control (General Category)
51	Hydraulic Oil
52	H20
53	CO2
54	Compressed Air
55	Natural Gas
56	Fog
57	Smoke
58	Cracked Haze
60	Pyro (General Category)
61	Fireworks
62	Explosions
63	Flame
64	Smoke pots
7F	All-types

Tabella 3.2. Formati di comando ammessi in MSC.

La logica perseguita è che, durante una *live performance*, i malfunzionamenti dei singoli dispositivi controllati non dovrebbero compromettere la comunicazione con altri macchinari nel *setup*. Questo concetto, riscontrabile nella progettazione dell'intero protocollo MIDI, determina l'utilizzo di una configurazione *open loop*. Va però sottolineato che le specifiche originarie prevedevano una futuribile evoluzione *closed loop*.

Pur mantenendo la logica *open loop* tipica di una catena MIDI, un dispositivo controllato può a sua volta diventare dispositivo controllore per le apparecchiature a valle.

3.2.3 Identificazione dei dispositivi

Similmente a MMC, anche in MSC il campo `<device_ID>` assume fondamentale importanza. In questo caso, però, si tratta sempre dell'indirizzo del dispositivo destinatario, essendo la modalità operativa *open loop*.

A ogni dispositivo nel *setup* è possibile associare un valore numerico compreso tra 0 e 127. Se uno stesso comando deve essere inviato a più macchinari, una scelta banale è inviare più messaggi SysEx a differenti destinatari. Esiste però anche la possibilità di identificare gruppi di dispositivi o di usare l'indirizzo di *all-call*. Gli intervalli di valori numerici riservati alle tre possibilità sono:

- $[00_{16}, 6F_{16}]$, ossia [0, 111], per l'identificazione dei singoli dispositivi;
- $[70_{16}, 7E_{16}]$, ossia [112, 126], per l'identificazione di 15 possibili raggruppamenti;
- $7F_{16}$ ossia 127, per *all-call*.

Ciascun dispositivo deve essere in grado di rispondere sia a messaggi individuali sia a quelli indirizzati ad *all-call*; l'individuazione di gruppi è opzionale. Il protocollo ammette, inoltre, che

a un dispositivo possano essere associati più identificativi individuali e più indirizzi di gruppo, caso in cui esso risponderà ai comandi destinati a ciascuno degli indirizzi menzionati. Infine, più apparecchiature controllate possono presentare lo stesso device_ID, caso in cui risponderanno tutte agli stessi comandi propagati lungo la catena e destinati a tale indirizzo.

3.2.4 Esempio

In questo paragrafo si prende in considerazione il sistema *Cognito* di *Pathway Connectivity*, una console compatta per il controllo delle luci che supporta il protocollo MSC.

Il generico messaggio SysEx, già menzionato in precedenza, è il seguente:

```
F0 7F <device_ID> 02 <command_format> <command> <data> F7
```

Si assuma `<device_ID>` fissato a 01_{16}.

Per questo modello, `<command_format>` è tipicamente posto a 01_{16}, che denota la categoria generale dei sistemi di illuminazione.

In merito ai comandi, ossia ai valori ammessi per il campo `<command>`, *Cognito* supporta unicamente 01_{16} (`GO`), 02_{16} (`STOP`), 03_{16} (`RESUME`), 04_{16} (`TIMED_GO`) e $0A_{16}$ (`RESET`).

Nel formato di rappresentazione supportato da *Cognito*, i numeri di *cue* e di playlist vengono espressi attraverso rappresentazioni ASCII, in cui l'intervallo [0, 9] viene mappato sui valori [30_{16}, 39_{16}].

Il messaggio di `GO` alla *cue* 45 nella playlist 68 verrebbe rappresentato come

```
F0 7F 01 02 01 01 34 35 00 36 38 F7
```

ove:

- i primi due byte denotano il messaggio SysEx universale in tempo reale;
- il terzo byte, posto arbitrariamente a 01_{16}, identifica il dispositivo destinatario;
- il quarto byte, posto a 02_{16}, caratterizza un generico messaggio MSC;
- il quinto byte, posto a 01_{16}, indica un messaggio della categoria generale per i dispositivi di controllo delle luci;
- il sesto byte, posto a 01_{16}, identifica il comando di `GOTO`;
- i byte a seguire forniscono una rappresentazione del *cue number* (la sequenza $34_{16}\,35_{16}$ si traduce in 45) e della playlist ($36_{16}\,38_{16}$ si traduce in 68), separati da 00_{16}, come previsto dal formato del comando;
- l'ultimo byte, $F7_{16}$, è un messaggio EOX.

Capitolo 4

General MIDI e altre estensioni

Questo capitolo è dedicato alle estensioni di maggior rilievo apportate negli anni al protocollo MIDI originario al fine di standardizzarne alcuni aspetti legati alle performance musicali. Le conseguenze di tali interventi si sono riverberate sulla progettazione e sulla commercializzazione di dispositivi MIDI compatibili con le nuove specifiche, particolarmente rivolti a determinate classi di utenti (ad esempio gli appassionati di videogame, i tastieristi che lavorano con basi musicali, e via dicendo).

Nel corso del capitolo, verranno trattati il General MIDI livello 1 (Paragrafo 4.1), le iniziative indipendenti dette Roland GS e Yamaha XG (Paragrafo 4.2), il General MIDI livello 2 (Paragrafo 4.3) e la recente estensione nota come MIDI Polyphonic Expression (Paragrafo 4.4).

4.1 General MIDI

Il **General MIDI System Level 1**, chiamato per brevità **General MIDI** (GM), è una specifica che definisce le caratteristiche minime di una classe di sintetizzatori MIDI. Sviluppata da MMA e JMSC e pubblicata per la prima volta nel 1991, la sua consultazione è stata poi associata al documento di specifica MIDI 1.0 [28].

GM mira a risolvere alcuni potenziali problemi e ambiguità nel funzionamento dei sintetizzatori MIDI, ed è specificamente pensato per garantire un elevato grado di compatibilità nella riproduzione di brani MIDI da parte dei sintetizzatori che lo adottano. Va osservato che un modulo sonoro può risultare compatibile con il protocollo MIDI pur senza supportare questo set aggiuntivo di specifiche che, peraltro, è stato fissato un decennio dopo il rilascio del protocollo originario. Inoltre, nei sintetizzatori compatibili con GM, è possibile "spegnere" la modalità operativa GM, il che non inibisce la capacità di ricevere e interpretare correttamente i messaggi in arrivo dalla catena a monte.

La motivazione che ha spinto al rilascio delle specifiche GM è che, in loro assenza, la riproduzione di dati MIDI su differenti sintetizzatori, o semplicemente su dispositivi con impostazioni diverse, può portare a risultati notevolmente eterogenei e, in alcuni casi, difficilmente prevedibili. Prima dell'avvento di GM, le configurazioni e i dati MIDI dovevano essere studiati per un particolare *setup* al fine di ottenere la performance prevista.

A titolo di esempio, si pensi a un messaggio NOTE-ON associato al Canale c e immesso sulla catena. In generale, non è dato sapere quale timbro nel sintetizzatore destinatario sia correntemente associato al Canale c. Il problema potrebbe essere risolto inviando prima un PROGRAM CHANGE sul Canale c tramite cui impostare il timbro al valore p, ma – ancora una volta – le specifiche MIDI non determinano un'associazione prefissata tra p e uno specifico timbro. Questa situazione, estesa ai vari sintetizzatori in cascata, determina potenzialmente la produzione di timbri non coerenti in risposta a un singolo messaggio. Altri esempi paradigmatici includono la risposta a messaggi PITCH BEND CHANGE, che varia a seconda di come è impostato l'intervallo da suddividere, la mappatura dei timbri di batteria, e il numero di voci supportate nella polifonia.

Per utenti professionisti, capaci di mantenere sotto controllo i parametri delle performance ed esperti nell'uso dei dispositivi, questa variabilità nelle risposte dei sintetizzatori può essere fonte di interesse e oggetto di personalizzazione. Ad esempio, sarebbe molto vincolante dover assegnare in ciascun sintetizzatore un dato numero di programma al timbro di clavicembalo se la performance non ne prevede l'utilizzo. Analogamente, sarebbe impossibile predeterminare tutti i timbri di interesse per una qualsiasi performance MIDI, potendo essa spaziare dalla colonna sonora di un videogioco a un sofisticato brano di musica elettronica con suoni campionati. Al contrario, per gli utenti non esperti è un vantaggio poter fare affidamento su un set predefinito di impostazioni e associazioni che renda loro fruibile una *MIDI song* così come è stata concepita. Esistono quindi due esigenze differenti: da un lato, mantenere la flessibilità nella caratteristiche e nella configurazione dei sintetizzatori (e preservare la piena compatibilità con le specifiche MIDI originarie), e, dall'altro, supportare una "configurazione MIDI minimale", adottabile dai produttori come standard industriale, che funga da riferimento per alcune tipologie d'uso del MIDI. GM risponde a quest'ultima esigenza.

Come detto, il supporto a GM da parte dei sintetizzatori non è richiesto per poterli definire compatibili con MIDI: esistono moduli sonori MIDI che non implementano le specifiche GM, e questi possono convivere in un *setup* con sintetizzatori GM; inoltre, sui dispositivi GM in generale è possibile passare a una modalità di funzionamento non GM.

4.1.1 Requisiti per il GM

I dispositivi MIDI, per potersi definire compatibili con GM, devono soddisfare alcuni requisiti minimi. Si osservi che queste caratteristiche riguardano non solo i sintetizzatori, ma anche i *controller*.

Riguardo al numero di voci, i dispositivi devono supportare l'accensione simultanea di 24 voci ripartite tra suoni melodici e percussivi (talvolta, l'indicazione è di 16 voci melodiche e 8 percussive), con risposta al parametro di velocity.

Riguardo ai canali, i dispositivi GM devono supportare simultaneamente tutti i 16 canali, con possibilità di assegnare a ciascun canale uno strumento diverso. Il Canale 10 è riservato ai suoni percussivi, per cui i pitch vengono interpretati in modo differente rispetto a quanto avviene sugli altri canali (si veda più avanti). Ogni canale deve gestire la polifonia, ossia l'accensione di più note simultanee.

I dispositivi di sintesi supportano un minimo di 128 numeri di programma MIDI e 47 suoni di percussioni. Per i suoni melodici, le associazioni tra numeri di programma e timbri sono prefissate: si tratta di 16 famiglie di timbri, ciascuna costituita da 8 strumenti (si veda la Tabella 4.1). Accanto a un elenco necessariamente non esaustivo di strumenti tradizionali ed elettronici, si può notare la presenza di effetti sonori quali il cinguettio, l'applauso e lo sparo di pistola, che di rado trovano applicazione nelle performance musicali [1].

Per quanto concerne i suoni percussivi, i moduli sonori GM mappano i pitch (e non i numeri di programma) in arrivo sul Canale 10 su strumenti a suono non determinato, come mostrato nella Tabella 4.2.

Quindi c'è una differenza sostanziale tra la gestione dei suoni melodici (in arrivo su tutti i canali tranne che sul Canale 10, ove verrebbero interpretati come percussivi) e quella dei suoni percussivi. Nel primo scenario, è il numero di programma a determinare il timbro, dunque l'informazione viene associata al canale attraverso messaggi PROGRAM CHANGE. Nel secondo caso il timbro percussivo è legato al valore del pitch nel messaggio NOTE-ON sul Canale 10.

Si rammentano alcuni aspetti che dovrebbero risultare oramai usuali al lettore. Innanzitutto la numerazione dei timbri nella tabella parte da 1, ma all'interno del messaggio i valori ammessi ricadono nell'intervallo [0, 127]. Ad esempio, per selezionare la celesta (numero di programma 9), va inviato il seguente messaggio PROGRAM CHANGE:

[1] Peraltro, si osservi che tali effetti possono essere modulati in altezza sfruttando il parametro del pitch; è possibile, ad esempio, eseguire una scala di Sol maggiore con suoni di squillo del telefono.

Piano	**Chromatic Percussion**	**Organ**	**Guitar**
1 Acoustic Grand Piano	9 Celesta	17 Drawbar Organ	25 Acoustic Guitar (nylon)
2 Bright Acoustic Piano	10 Glockenspiel	18 Percussive Organ	26 Acoustic Guitar (steel)
3 Electric Grand Piano	11 Music Box	19 Rock Organ	27 Electric Guitar (jazz)
4 Honky-tonk Piano	12 Vibraphone	20 Church Organ	28 Electric Guitar (clean)
5 Electric Piano 1	13 Marimba	21 Reed Organ	29 Electric Guitar (muted)
6 Electric Piano 2	14 Xylophone	22 Accordion	30 Overdriven Guitar
7 Harpsichord	15 Tubular Bells	23 Harmonica	31 Distortion Guitar
8 Clavinet	16 Dulcimer	24 Tango Accordion	32 Guitar harmonics
Bass	**Strings**	**Ensemble**	**Brass**
33 Acoustic Bass	41 Violin	49 String Ensemble 1	57 Trumpet
34 Electric Bass (finger)	42 Viola	50 String Ensemble 2	58 Trombone
35 Electric Bass (pick)	43 Cello	51 Synth Strings 1	59 Tuba
36 Fretless Bass	44 Contrabass	52 Synth Strings 2	60 Muted Trumpet
37 Slap Bass 1	45 Tremolo Strings	53 Choir Aahs	61 French Horn
38 Slap Bass 2	46 Pizzicato Strings	54 Voice Oohs	62 Brass Section
39 Synth Bass 1	47 Orchestral Harp	55 Synth Voice	63 Synth Brass 1
40 Synth Bass 2	48 Timpani	56 Orchestra Hit	64 Synth Brass 2
Reed	**Pipe**	**Synth Lead**	**Synth Pad**
65 Soprano Sax	73 Piccolo	81 Lead 1 (square)	89 Pad 1 (new age)
66 Alto Sax	74 Flute	82 Lead 2 (sawtooth)	90 Pad 2 (warm)
67 Tenor Sax	75 Recorder	83 Lead 3 (calliope)	91 Pad 3 (polysynth)
68 Baritone Sax	76 Pan Flute	84 Lead 4 (chiff)	92 Pad 4 (choir)
69 Oboe	77 Blown Bottle	85 Lead 5 (charang)	93 Pad 5 (bowed)
70 English Horn	78 Shakuhachi	86 Lead 6 (voice)	94 Pad 6 (metallic)
71 Bassoon	79 Whistle	87 Lead 7 (fifths)	95 Pad 7 (halo)
72 Clarinet	80 Ocarina	88 Lead 8 (bass + lead)	96 Pad 8 (sweep)
Synth Effects	**Ethnic**	**Percussive**	**Sound effects**
97 FX 1 (rain)	105 Sitar	113 Tinkle Bell	121 Guitar Fret Noise
98 FX 2 (soundtrack)	106 Banjo	114 Agogo	122 Breath Noise
99 FX 3 (crystal)	107 Shamisen	115 Steel Drums	123 Seashore
100 FX 4 (atmosphere)	108 Koto	116 Woodblock	124 Bird Tweet
101 FX 5 (brightness)	109 Kalimba	117 Taiko Drum	125 Telephone Ring
102 FX 6 (goblins)	110 Bag pipe	118 Melodic Tom	126 Helicopter
103 FX 7 (echoes)	111 Fiddle	119 Synth Drum	127 Applause
104 FX 8 (sci-fi)	112 Shanai	120 Reverse Cymbal	128 Gunshot

Tabella 4.1. Associazione tra numeri di programma e timbri GM a suono determinato.

35 Acoustic Bass Drum	51 Ride Cymbal 1	67 High Agogô
36 Electric Bass Drum	52 Chinese Cymbal	68 Low Agogô
37 Side Stick	53 Ride Bell	69 Cabasa
38 Acoustic Snare	54 Tambourine	70 Maracas
39 Hand Clap	55 Splash Cymbal	71 Short Whistle
40 Electric Snare	56 Cowbell	72 Long Whistle
41 Low Floor Tom	57 Crash Cymbal 2	73 Short Guiro
42 Closed Hi-hat	58 Vibra Slap	74 Long Guiro
43 High Floor Tom	59 Ride Cymbal 2	75 Claves
44 Pedal Hi-hat	60 High Bongo	76 High Woodblock
45 Low Tom	61 Low Bongo	77 Low Woodblock
46 Open Hi-hat	62 Mute High Conga	78 Mute Cuica
47 Low-Mid Tom	63 Open High Conga	79 Open Cuica
48 Hi-Mid Tom	64 Low Conga	80 Mute Triangle
49 Crash Cymbal 1	65 High Timbale	81 Open Triangle
50 High Tom	66 Low Timbale	

Tabella 4.2. Associazione tra pitch MIDI e timbri percussivi GM.

```
1100nnnn 00001000
```

Inoltre, anche con l'adozione delle specifiche GM, esiste ampia discrezionalità da parte dei costruttori sulla qualità dei timbri prodotti dai sintetizzatori. In altre parole, GM garantisce che al timbro 1 venga assegnato uno strumento che macroscopicamente suona su tutti i moduli sonori compatibili come un pianoforte acustico da concerto (e non un flauto o una chitarra), ma la fedeltà del suono può essere molto eterogenea a seconda delle tecniche di sintesi, degli algoritmi, dei campioni in uso, ecc.

Riguardo ai messaggi di canale, i dispositivi compatibili con GM devono supportare i numeri di CONTROL CHANGE 1 (*Modulation wheel*), 7 (*Channel volume*), 10 (*Channel pan*), 11 (*Expression controller*), 64 (*Sustain pedal*), 100 (*Registered Parameter Number LSB*), 101 (*Registered Parameter Number MSB*), 121 (*All controllers off*) e 123 (*All notes off*). Vanno inoltre gestite le variazioni di pressione del canale (Paragrafo 2.4.6) e il pitch bend (Paragrafo 2.4.9).

Infine, essi devono rispondere all'immissione di numeri di parametro registrati e non registrati (si veda il Paragrafo 2.5.4) con granularità fine e grossolana e permettere di impostare l'intervallo di pitch bend.

La modalità operativa GM si può accendere e spegnere attraverso un messaggio SYSEX universale. Infatti alcuni sintetizzatori presentano tanto moduli e set di *patch* GM quanto non GM. Ne è un esempio l'expander multitimbrico *Yamaha MU80*, che presenta entrambe le modalità nonché il supporto all'estensione XG (si veda il Paragrafo 4.2). Quando la modalità operativa GM è abilitata, il banco di suoni corrente viene sostituito da quello GM. Spegnendo il modo GM, è invece possibile accedere ai restanti timbri con i consueti messaggi di PROGRAM CHANGE e di CONTROL CHANGE di tipo *Bank Select*.

Un altro messaggio SYSEX richiesto da GM è quello che determina il volume *master* del sintetizzatore. Nel Paragrafo 2.10.2 vengono riportati esempi di entrambi i SYSEX menzionati.

4.2 Roland GS e Yamaka XG

Seguendo la logica che ha portato alle specifiche GM, alcuni produttori hanno deciso di creare ulteriori estensioni ancora più restrittive in termini di caratteristiche dei dispositivi, di interpretazione di alcuni comandi MIDI e di definizione di specifiche sequenze di byte. In particolare, Roland ha sviluppato lo standard GS e Yamaha lo standard XG. I dispositivi GS e quelli XG sono dunque compatibili con lo standard GM.

Roland GS, indicato brevemente con GS, è l'acronimo di *General Standard* o *General Sound*. Rispetto alle specifiche GM, GS aggiunge 98 strumenti a suono determinato, 15 strumenti percussivi, 8 kit di batteria, 3 effetti (*reverb*, *chorus* e *variation*). L'estensione GS è stata introdotta per la prima volta sui moduli della serie *Sound Canvas*, a partire dal *Roland SC-55* del 1991. Tale modello supportava 317 timbri, con polifonia a 16 voci melodiche e 8 voci percussive simultanee, e ha conosciuto una grande popolarità anche grazie all'emulazione del *Roland MT-32* (pur non supportandone le caratteristiche di programmabilità). Oltre alla serie *Sound Canvas*, Roland ha introdotto la modalità GS nella propria linea professionale attraverso la tastiera *JV-30* e la scheda di espansione *VE-GS1* ideata per altri strumenti della serie *JV*. La compatibilità GS è fornita nella specifica GM2 (vedi Paragrafo 4.3) che Roland ha contribuito a creare e sta attivamente supportando.

Anche Yamaha ha lavorato su una propria estensione di GM, chiamata **Yamaha XG**, acronimo di *eXtended General MIDI*. La finalità di XG sono simili a quelle di GS: aumentare il numero di timbri predefiniti, implementare *controller* aggiuntivi e fornire accesso ad alcuni parametri standard addizionali, in modo da aumentare le possibilità di controllo e il realismo nella performance MIDI. Riguardo agli strumenti, le specifiche prevedono l'aumento dei timbri dai 128 propri di GM a 480 (o 361 in alcune implementazioni) propri di XG, con 11 kit di batteria aggiuntivi. Lo standard XG ha conosciuto principalmente tre raffinamenti delle specifiche, note come livelli: XG level 1 (1994), XG level 2 (1997) e XG level 3 (1998). XG supporta la polifonia a 32, 64 o 128 voci, a seconda del livello. Quest'ultimo aspetto influenza anche il numero di canali MIDI supportati: 16, 32 o 64, a

Figura 4.1. Chip *Yamaha YMF744B-V XG*.

seconda del livello. Per mantenere la piena compatibilità con MIDI, però, ogni blocco da 16 canali viene gestito da una differente porta MIDI; quindi, ad esempio, i dispositivi che implementano XG level 2 devono avere 2 porte indipendenti per supportare 32 canali, e ne servono 4 per i 64 canali di XG level 3. Tutti i dispositivi XG sono dotati di un sistema di elaborazione degli effetti con riverbero e *chorus* indipendenti su ciascuno dei canali supportati, danno la possibilità di fare routing di qualsiasi canale, e possono simulare effetti aggiuntivi quali amplificatori per chitarra e pedali *wah-wah*.

Yamaha ha rilasciato il primo prodotto basato su XG nel 1994: si tratta del *MU80 Tone Generator*. Nel 1995 è stata commercializzata la scheda di espansione *DB50XG*, primo dispositivo compatibile con XG per utenti PC e, ai tempi del suo rilascio, diretta concorrente di *Creative Wave Blaster*. Nel 1996 nasce il modulo esterno MU10 (in pratica, una scheda *DB50XG* all'interno di un case) e successivamente la scheda ISA *SW60XG* per PC. Oltre a un'unità di generazione del suono, entrambi i dispositivi includono un banco di suoni campionati da 4 MB ai tempi molto apprezzato dagli esperti. Tra le schede dotate di tecnologia XG vale la pena citare il modello *SW1000XG*, molto in voga nell'industria musicale professionale. Numerose tastiere con funzioni di sinesi prodotte da Yamaha e rivolte al pubblico amatoriale o professionale hanno implementato XG nella sua forma completa o in un suo sottoinsieme noto come *XGlite*, rilasciato nel 2002. Inoltre, alcuni notebook includono il chipset *Yamaha YMF7xx* (vedi Figura 4.1), dotato di un sintetizzatore MIDI compatibile con XG in scala ridotta. Va citato, infine, il *SoftSynthesizer S-YXG50*, anch'esso compatibile con XG. Si tratta di un sintetizzatore virtuale, ormai fuori produzione, pensato per applicazioni principalmente videoludiche in cui musica ed effetti sonori venivano demandati a MIDI. Essendo puramente software e facendo uso di un set di suoni salvato in una *wavetable* da 2 MB o 4 MB, il suo punto di forza era l'ottimo compromesso tra alta qualità dei campioni e prezzo economico.

A livello di compatibilità, la maggior parte dei moduli sonori XG può passare automaticamente alla modalità *TG300B*, che è un'emulazione di GS in grado di supportare adeguatamente la riproduzione delle performance pensate per lo standard concorrente. Anche Korg, azienda vicina a Yamaha, ha rilasciato strumenti con compatibilità XG, in alcuni casi attraverso l'installazione di una scheda di espansione (come nel modello *NS5R*), in altri casi tramite l'inserimento di componentistica XG direttamente all'interno degli strumenti (come nel modello *NX5R*). Korg è l'unico produttore al di fuori di Yamaha ad aver prodotto strumenti certificati XG, tra cui le tastiere

e i moduli sonori appartenenti alla serie *N*, che offrono supporto anche agli standard GS e GM2 promossi da Roland.

4.3 General MIDI Level 2

Le specifiche GM di inizio anni '90 vengono definitivamente superate nel 1999 con il rilascio di **General MIDI Level 2** (GM2). Questa nuova versione, dal punto di vista terminologico, rende più opportuno chiamare l'originale GM come *General MIDI Level 1*. Si tratta di un'estensione ulteriore del protocollo MIDI ispirata dalle precedenti specifiche GM e dallo standard GS, promulgata dalla *MIDI Manufacturers Association* (MMA).

I requisiti generali per i sintetizzatori compatibili con GM2 prevedono la polifonia a 32 note simultanee, la gestione contemporanea di 16 canali MIDI, il supporto di fino a 16 strumenti melodici (su tutti i canali) e di due kit percussivi (sui Canali 10 e 11).

GM2 raddoppia il numero di timbri standardizzati, passando dai 128 propri di GM a 256. Per mantenere la compatibilità con le specifiche MIDI, si fa uso dei banchi di suoni. In particolare, la selezione di uno specifico strumento richiede di usare un messaggio CONTROL CHANGE 0 (Bank Select MSB) impostato a 121 e un messaggio CONTROL CHANGE 32 (Bank Select LSB) prima di PROGRAM CHANGE. Il banco di suoni numerato 0 contiene i suoni GM, quindi viene mantenuta la compatibilità anche con la versione precedente di General MIDI.

Ad esempio, al primo *program number* (indice 0) viene ora fatto corrispondere l'Acoustic Grand Piano sul banco 0, il Wide Acoustic Grand sul banco 1 e il Dark Acoustic Grand sul banco 2. Si osservi che non tutti i *program number* sono presenti su tutti i banchi: il banco 0, come detto, contiene tutti e soli i timbri GM, quindi i 128 valori trovano piena corrispondenza; il numero di programma 6 (indice 5) è implementato in ben 5 banchi (Chorused Electric Piano, Detuned Electric Piano 2, Electric Piano 2 Variation, Electric Piano Legend, Electric Piano Phase), mentre il programma 9 (indice 8) corrisponde al solo timbro di Celesta nel banco 0.

I kit di suoni percussivi riprendono l'estensione GS e si selezionano attraverso il *Bank Select MSB* posto a 120 e il *Bank Select LSB* a 0. All'interno di questo banco si ritrovano i seguenti numeri di programma:

- 1 – *Standard Kit*, l'unico previsto dall'originario GM;
- 9 – *Room Kit*, registrato in un piccolo ambiente chiuso;
- 17 – *Power Kit*, con suoni kick e snare più potenti;
- 25 – *Electronic Kit*, che riprende i suoni delle batterie elettroniche;
- 26 – *TR-808 Kit*, che si rifà al modello *Roland TR-808*;
- 33 – *Jazz Kit*, con suoni di kick e snare più morbidi rispetto al kit standard;
- 41 – *Brush Kit*, ove i suoni sono ottenuti con le spazzole;
- 49 – *Orchestra Kit*, con percussioni da orchestra sinfonica e timpani;
- 57 – *Sound FX Kit*, collezione di effetti sonori.

Lo standard GM2 estende inoltre i numeri di nota dedicati, all'interno di un kit percussivo, a identificare i suoni. I pitch ammessi ora vanno da 27 a 87, includendo 14 numeri di nota addizionali e implementandoli nei kit numerati da 1 a 49 (tutti a esclusione del *Sound FX Kit*).

Infine, GM2 stabilisce un nuovo e più ricco elenco di messaggi CONTROL CHANGE, di *Registered Parameter Numbers* e di messaggi SYSEX universali che i sintetizzatori compatibili devono essere in grado di gestire.

4.4 MIDI Polyphonic Expression

MIDI Polyphonic Expression (MPE), originariamente chiamato *Multidimensional Polyphonic Expression*, non costituisce un'estensione propriamente detta delle specifiche MIDI, ma rappresenta piuttosto un modo di impiego dei messaggi già previsti in MIDI 1.0 al fine di

consentire a *controller* "multidimensionali" una maggiore espressività nell'esecuzione. Grazie a un'interazione più sofisticata su ciascuna nota suonata, un *controller* MPE può simulare il comportamento di strumenti acustici reali, in particolare quelli cordofoni, e supportare timbriche appositamente progettate.

MPE deve la sua nascita, il suo sviluppo e la sua affermazione principalmente all'opera di ROLI, azienda produttrice di *controller* evoluti con funzioni di sintesi che traggono beneficio dalla sua applicazione. Presentate per la prima volta alla fiera NAMM del 2015, le specifiche MPE vedono successivamente la collaborazione di Apple, Moog Music e altri produttori. Nel gennaio 2016 viene istituito un gruppo di lavoro dedicato presso MMA. Trattandosi di una modalità operativa compatibile con MIDI 1.0 anziché di un nuovo standard o di un'estensione, non si può parlare di una vera e propria standardizzazione in ambito MIDI, bensì di un'aggiunta (*enhancement*) adottata ufficialmente dalla comunità nel gennaio 2018.

L'idea alla base di MPE è riservare un singolo canale MIDI a ciascuna nota suonata. L'associazione è temporanea, ossia ha luogo dal messaggio di NOTE-ON al corrispettivo NOTE-OFF. In pratica, il concetto di accensione della nota si sovrappone a quello di occupazione di un dato canale, ma con un'allocazione dinamica delle risorse. Questo approccio permette di controllare individualmente il parametro di velocity all'attacco e al rilascio della nota, come pure di disporre di CONTROL CHANGE, CHANNEL PRESSURE (Aftertouch) e PITCH BEND CHANGE contestuali alla singola nota.

Negli strumenti MIDI tradizionali, i messaggi NOTE-ON, NOTE-OFF, CONTROL CHANGE, CHANNEL PRESSURE (Aftertouch) e PITCH BEND CHANGE fanno parte della famiglia *Channel Voice* e vengono trasmessi e ricevuti su un singolo canale. Ad esempio, agendo sulla *wheel* per il pitch bend di un *controller* tradizionale, i messaggi generati influenzano tutte le note suonate dalla tastiera, essendo i suoi tasti associati a uno stesso canale MIDI [2]. In un *controller* hardware che implementa MPE, invece, ogni nota trasmette i propri messaggi di controllo su un canale a essa temporaneamente dedicato. Il meccanismo potrebbe ricordare quello di un *controller* con tante aree di *split* quanti sono i tasti, ma, per mantenere la compatibilità con i 16 canali MIDI, le associazioni sono limitate in numero e variabili nel tempo.

Un tipico *controller* MPE, al fine di sfruttare le caratteristiche dell'estensione, deve essere in grado di leggere molti più parametri rispetto agli strumenti elettronici tradizionali. Ad esempio, i dispositivi della serie *ROLI Seabord*, pur offrendo un'interfaccia simile a quella delle tastiere (Figura 4.2), presentano una superficie continua, in grado di leggere movimenti orizzontali (asse x) e verticali (asse y) sul singolo tasto. I primi generano messaggi PITCH BEND CHANGE, simulando il comportamento di una corda soggetta al bending; i secondi si traducono in CONTROL CHANGE 74, che in MPE corrisponde a un parametro detto *timbre*. Per quanto riguarda l'*aftertouch*, ossia la sensibilità alla variazione di pressione sul tasto, MPE, per ovvi motivi, impiega il messaggio di CHANNEL PRESSURE anziché quello di POLYPHONIC KEY PRESSURE.

Esistono però altri messaggi MIDI di canale che riguardano il funzionamento del *controller* MPE nel suo complesso (ad esempio, PROGRAM CHANGE, altri CONTROL CHANGE quali 7 – Volume, 64 – Sustain, ecc.). A questo scopo, uno dei canali disponibili viene identificato come Master o Main Channel e destinato a gestire i messaggi di valenza generale. Quindi, secondo le specifiche MPE, sono al massimo 15 le note simultanee, dato che i canali disponibili nello standard MIDI sono 16 e, in MPE, uno di questi viene designato come Master Channel. La configurazione MPE di default, ad esempio, prevede per le *ROLI Seabord* che il Canale 1 funga da Master e i Canali dal 2 al 16 siano associabili dinamicamente alle note. Nel caso, piuttosto raro, di avere già 15 note simultaneamente accese, quelle ulteriori saranno distribuite sui canali in uso; di conseguenza alcuni messaggi di controllo influenzeranno più note associate al medesimo canale.

Se l'intero *setup* MIDI è in grado di supportare la modalità MPE, il risultato della performance è pienamente predicibile. Qualche problema potrebbe sorgere, invece, se un *controller* MPE venisse collegato a un sintetizzatore "tradizionale". Innanzitutto, quest'ultimo dovrebbe essere

[2] Fanno eccezione le tastiere che permettono di definire aree di *split* e associarle a canali differenti, ma all'interno di ciascuna area si verifica la stessa situazione.

Figura 4.2. Interfaccia del *controller ROLI Seabord Rise 25*.

in ascolto su tutti i canali, in modo da non smarrire messaggi di performance in arrivo su canali diversi dal canale base. In un setup tradizionale, se il *controller* invia messaggi sul Canale N, è sufficiente che il modulo sonoro sia in ascolto sul solo Canale N; ma questo non è sufficiente in un contesto MPE, che sfrutta potenzialmente tutti i canali disponibili per veicolare informazioni di performance. Inoltre un sintetizzatore multi-timbrico destinatario dei messaggi di performance deve essere opportunamente configurato per associare a tutti i canali lo stesso timbro, altrimenti ogni nota generata (anche da uno stesso *controller*) è potenzialmente riprodotta da uno strumento diverso, peraltro in modo difficilmente prevedibile per via dell'allocazione dinamica delle risorse.

MPE è anche predisposto per operare in modalità multi-timbrica, grazie all'identificazione delle cosiddette zone. Teoricamente è possibile creare fino a 8 zone, ripartendo i 16 canali MIDI in 8 canali *master* e 8 canali membri, ma è raro che un dispositivo ne gestisca più di 2. La configurazione di tali zone è demandata al messaggio CONTROL CHANGE RPN 6 sul canale dal quale la zona ha inizio, seguito dal CONTROL CHANGE 6 – Data Entry MSB, che contiene un valore pari al numero di canali per nota da destinare alla zona. Ad esempio, trasmettendo sul Canale 2 l'RPN 6 seguito dal CC 6 con valore 7, si genera una zona dal Canale MIDI 2 all'8. In questo caso, il Master Channel della zona è il Canale 1, perché il ruolo di Master Channel viene assegnato automaticamente al canale che precede quello sul quale viene trasmesso l'RPN 6. Volendo creare una seconda zona, compresa tra il Canale MIDI 10 e il 16, con il 9 come Master Channel, si deve trasmettere l'RPN 6 seguito dal CC 6 con valore 7 sul Canale MIDI 10.

Per ulteriori dettagli si rimanda alla documentazione ufficiale disponibile sul sito midi.org.[3]

[3] https://www.midi.org/midi-articles/midi-polyphonic-expression-mpe

Capitolo 5

Standard MIDI Files

Fino a questo punto della trattazione sono state prese in considerazione principalmente performance musicali estemporanee, in cui le azioni dei musicisti si traducono in messaggi MIDI attraverso i sensori di cui sono dotati i *controller*, e tali messaggi, una volta ricevuti dai sintetizzatori, generano (quasi) istantaneamente i corrispettivi suoni. Non si è affrontato, invece, il problema di salvare in memoria e caricare da memoria le sequenze di messaggi MIDI, il che richiede un opportuno formato di file.

Dal punto di vista storico, una risposta standard a questa esigenza viene trovata a distanza di tempo dal rilascio delle specifiche 1.0. Il formato *Standard MIDI Files* (SMF) nasce infatti nel 1986 a opera di Dave Oppenheimer di *OPCODE System*, e solo dal 1988 viene ufficialmente accolto da MMA. Grazie al supporto della polifonia e della multi-timbricità, e alla modesta occupazione di spazio, i file MIDI diventano presto popolari sul Web, e sono tuttora ampiamente utilizzati per la realizzazione e la condivisione di basi musicali.

Al giorno d'oggi i principali sistemi operativi e gran parte dei media player offrono supporto nativo alla riproduzione di file MIDI. Inoltre, molti *editor* digitali di partitura permettono di esportare la notazione musicale in formato MIDI, e, di converso, hanno una funzione di importazione dei file MIDI che cerca di ricostruirne il contenuto in forma simbolica.

Lo scopo del formato è l'interscambio di dati MIDI temporalmente marcati (ossia dotati di *timestamp*), tanto tra programmi installati sullo stesso computer, quanto tra sistemi diversi. I file in questo formato contengono metadati e informazioni di performance che vengono interpretate da un apposito lettore software o hardware, delegando la produzione finale del suono a un modulo sonoro collegato. In fase di progettazione è stata prestata particolare cura alla compattezza nella rappresentazione, risultato ottenibile proprio in virtù del fatto che i file MIDI non contengono dati audio, bensì comandi per pilotare opportunamente i sintetizzatori.

In questo capitolo il formato verrà descritto in dettaglio, discusso nelle sue peculiarità ed esemplificato. Innanzitutto, nel Paragrafo 5.1 si opererà un confronto con altri approcci, evidenziando come SMF non possa essere considerato a pieno titolo né un formato di partitura né un formato audio. Nel Paragrafo 5.2 si analizzeranno le caratteristiche tecniche dello standard e nel Paragrafo 5.3 si introdurrà il concetto di *chunk*, che rappresenta la struttura dati su cui il formato si basa. All'interno di un file MIDI è possibile codificare messaggi MIDI, eventi *SysEx* e meta-eventi; poiché i primi due argomenti sono già stati trattati nel Capitolo 2, il Paragrafo 5.4 si concentrerà sui soli meta-eventi. Infine, il Paragrafo 5.5 presenterà alcuni esempi, concisi ma completi, opportunamente commentati.

5.1 Confronto con i formati di partitura e audio

Il set di messaggi MIDI descrive un linguaggio progettato specificamente per trasmettere informazioni sulle performance musicali. Non si tratta propriamente né di musica, in quanto MIDI non codifica i simboli di partitura, né di suono, in quanto MIDI non veicola campioni audio (se non in casi eccezionali, facendo uso di messaggi SysEx). I messaggi MIDI sono concepiti per descrivere

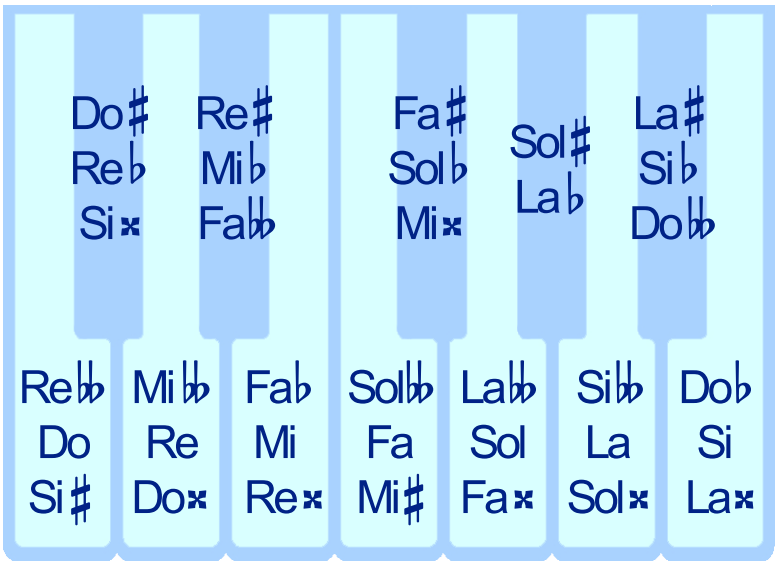


Figura 5.1. Omofonia tra note con diversi nomi e stati di alterazione.

una performance musicale tralasciando gli aspetti notazionali non significativi per la generazione del suono, ma consentendo comunque a un dispositivo che li riceve di ricostruire l'esecuzione musicale nel modo più fedele possibile.

Di conseguenza, il formato SMF non può dirsi né un formato di partitura né un formato audio. Si parla più propriamente di formato di *rappresentazione sub-simbolica*, o di *computer-driven performance*. Scopo del presente paragrafo è chiarire la differenza tra l'informazione MIDI da un lato, e l'informazione di partitura o l'informazione audio dall'altro.

5.1.1 MIDI e informazione di partitura

Apparentemente, il formato SMF è in grado di codificare informazione simbolica, tanto che gli *editor* di partitura normalmente leggono dati MIDI da file e ne forniscono una rappresentazione grafica su pentagramma. Le capacità in tal senso sono però limitate: la ricchezza simbolica di una partitura viene fortemente compromessa, e sovente vengono introdotti "errori" che riguardano perfino aspetti fondamentali della notazione.

A titolo di esempio, si riportano due casi emblematici:

- altezza della note. All'interno di una catena MIDI, un dato *key number* (ad esempio 60) è legato alla pressione di uno specifico tasto del *controller* (nell'esempio, il tasto del *do* centrale), e all'emissione della frequenza corrispondente (nell'esempio, 261,63 Hz) da parte del sintetizzatore. Quanto detto è coerente con il suo trasposto nel mondo analogico, dove la pressione del tasto determina la sollecitazione di una corda (o di un insieme di corde) finalizzata all'emissione di una specifica frequenza: si pensi alla meccanica di un pianoforte o di un clavicembalo. Tanto nel caso analogico quanto in quello digitale, però, solo a monte dell'azione del musicista interviene l'informazione di partitura che ne determina il gesto (nell'esempio, la lettura di *do* naturale o di *si*♯ o di *re*𝄫); dalla pressione del tasto in avanti, tale informazione non è più rilevante. Come mostrato nella Figura 5.1, limitandosi alle alterazioni doppie comunemente ammesse dal sistema musicale occidentale, ogni tasto può corrispondere a 3 notazioni distinte in partitura, a eccezione di *sol*♯/*la*♭. Il legame non biunivoco tra l'altezza notata e la corrispondente frequenza, una volta consumato, non è ricostruibile a posteriori; sono possibili solo delle inferenze rispetto a una scrittura ritenuta più plausibile, sulla base delle regole dell'armonia tonale. Ad esempio, se il contesto lascia presupporre la tonalità di Sol maggiore, sarà assai più probabile che il pitch 66 corrisponda a *fa*♯ anziché a *sol*♭;

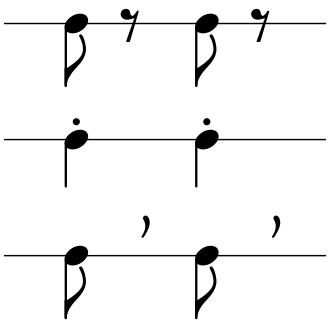


Figura 5.2. Tre differenti notazioni in partitura che potrebbero comportare, a livello di esecuzione, la stessa temporizzazione dei messaggi di NOTE-ON e NOTE-OFF.

- durata delle note. Mentre nella notazione musicale a un singolo glifo si fa corrispondere l'informazione sulla durata della nota, in formato MIDI questa deve essere ricostruita come distanza – espressa in *ticks* – tra il messaggio NOTE-OFF e il corrispettivo NOTE-ON. Anche nel caso – piuttosto improbabile – di esecuzione perfettamente a metronomo, una sequenza di distanze in *ticks* non è univocamente legata a una data scrittura in partitura: come mostrato nella Figura 5.2, potrebbe essere dovuta a un dato pattern ritmico (ad esempio, note da un ottavo seguite da pause di un ottavo), o all'intervento di segni di articolazione (ad esempio, note da un quarto con staccato), o alla presenza di simboli che alterano l'agogica (ad esempio, ottavi con segni di respiro).

A maggior ragione, il formato MIDI non consente di codificare i segni di agogica (indicazioni metronomiche, *rallentando*, *accelerando*, ecc.), di dinamica (***p***, ***mf***, ***f***, ecc.), di articolazione (legato, portato, staccato, ecc.) e via dicendo, tenendo traccia, al più, del loro effetto sulla performance.

Si osservi che il formato non nasce per supportare l'informazione simbolica di partitura. L'obiettivo del protocollo MIDI è la comunicazione numerica atta a pilotare, in ultima analisi, macchine per la produzione di suono. Nel momento in cui un sintetizzatore viene correttamente istruito sulla frequenza da emettere, sull'intensità da applicare e sull'istante esatto di accensione e spegnimento dell'evento sonoro, non si ha necessità di trasmettere informazione di partitura, che è non rilevante e genererebbe traffico indesiderato. Il formato SMF trae spunto dalle caratteristiche del protocollo e non presenta, tra le sue finalità, l'obiettivo di codificare informazione simbolica.

5.1.2 MIDI e informazione audio

MIDI viene spesso erroneamente citato tra i formati audio in ambito digitale. A differenza dei file audio digitali propriamente detti (WAV, AIFF, ecc.) e dei supporti digitali quali i compact disc audio (CD-DA), un file MIDI non necessita l'acquisizione e l'archiviazione di campioni di suono; esso contiene, piuttosto, un elenco di eventi. In altre parole, il tipo di informazione codificata in SMF non è un insieme di dati audio, ma una serie di comandi che istruiscono un modulo sonoro nel produrre audio. Questo conferisce ai file MIDI un'occupazione di spazio assai minore rispetto ai file audio digitali. Per quantificare il vantaggio, è stato considerato un brano di musica sinfonica della durata approssimativa di 10 minuti; in particolare, si è scelta una performance del primo movimento della *Sinfonia n°4* di Ludwig Van Beethoven. La codifica audio occupa circa 100 MB in formato WAV, circa 50 MB in un formato *lossless* (compresso senza perdita) quale FLAC, circa 10 MB in formato MP3 a 128 kbps, e meno di 100 KB in formato MIDI.

Questo aspetto ha decretato l'iniziale successo di MIDI come formato per l'audio su dispositivi dotati di poca memoria – ad esempio, erano MIDI le prime suonerie polifoniche dei telefoni cellulari – e per la distribuzione di contenuti musicali in un'epoca in cui le reti erano molto meno veloci e il formato MP3 non era ancora diffuso.

Un altro vantaggio degli SMF rispetto ai formati audio sta nella possibilità di riorganizzare e modificare gli eventi in modo agile. Ad esempio, è molto semplice in un *sequencer* stabilire che le note di un dato canale debbano essere eseguite da un timbro di flauto anziché da un sassofono, o che una certa parte debba essere trasposta all'ottava superiore, mentre tali operazioni all'interno di un audio non multitraccia risulterebbero molto complesse o addirittura impossibili con le tecniche attuali.

Il rovescio della medaglia sta in una certa imprevedibilità del risultato sonoro finale. La gradevolezza della performance ottenuta dall'esecuzione di un file MIDI dipende sia dalla cura prestata alla codifica dei messaggi, sia dalla qualità del sintetizzatore, e in particolare del banco di suoni impiegato. Lo stesso file MIDI, riprodotto da sistemi differenti, può portare a risultati qualitativi assai eterogenei. Un discorso analogo vale, ad esempio, per la colonna sonora MIDI di un videogame riprodotta da schede audio differenti [1]. Chi distribuisce un file MIDI può porre grande cura nella temporizzazione degli eventi, nell'uso delle dinamiche, nell'applicazione di effetti, e via dicendo, ma non ha controllo sul modulo sonoro utilizzato per la sua riproduzione, e questo limita notevolmente l'impiego dei file MIDI in contesti professionali.

Come effetto collaterale difficilmente prevedibile, proprio la variabilità nella qualità di riproduzione ha decretato il successo dei file MIDI come basi musicali. Trattandosi di suoni diversi da quelli delle registrazioni audio commerciali, i file MIDI di composizioni, prodotti o rielaborati dall'utente finale, non comportano corresponsione di diritti di riproduzione, ma solo di eventuali diritti d'autore. Va detto che, sotto il profilo giuridico, i file MIDI si possono considerare a tutti gli effetti opere dotate di originalità e valore artistico, quindi qualsiasi forma di registrazione, riproduzione, comunicazione e diffusione da parte di soggetti terzi è considerata violazione dei diritti d'autore, se non si possiede una regolare licenza. Viste le peculiarità dei file MIDI, può però risultare molto difficile dimostrarne la paternità.

5.1.3 Black MIDI

A conclusione del paragrafo, si ritiene utile menzionare uno scenario che mette in evidenza le peculiarità di MIDI come formato di *computer-driven performance*. Si tratta di un genere di composizione musicale chiamato *Black MIDI*, caratterizzato dall'uso di un enorme numero di note (anche) simultanee. Si spazia dalle migliaia alle migliaia di miliardi di eventi sonori. Sebbene non esista una definizione formale del genere, è proprio dall'estrema densità notazionale che deriva la definizione "black": si tratta di brani la cui partitura, se notata secondo le modalità tradizionali, sarebbe quasi completamente annerita dalla rappresentazione fitta delle note.

Composizioni impossibili o estremamente difficili da eseguire esistevano ben prima della nascita del protocollo MIDI. Tra i precursori del genere, è possibile annoverare Conlon Nancarrow, che creava composizioni musicali molto complesse codificandole su schede perforate per pianola (strumento conosciuto anche come autopiano o pianoforte meccanico); sono noti, ad esempio, i suoi 49 "Studies for Player Piano" composti tra il 1948 e il 1992. Un'altra composizione di rilievo in questo ambito è "Circus Galop", scritta da Marc-Andre Hamelin tra il 1991 e il 1994 e contenente arrangiamenti impossibili per un pianista, con accordi che arrivano ad abbracciare anche 21 note simultanee. Infine, dal punto di vista storico è opportuno citare "The Black Page" di Frank Zappa, composizione che risale alla metà degli anni '70: già nel nome sembra essere un precursore del genere *Black MIDI*.

Sebbene non ne sia nota l'esatta origine, la popolarità globale di questa tecnica compositiva sembra potersi ricondurre a un brano caricato su YouTube dall'utente Kakakakaito1998 nel febbraio 2011.

Il *Black MIDI* sfrutta tanto le caratteristiche notazionali quanto le funzioni di *computer-driven performance* legate al MIDI. Infatti, da un lato tale tecnica si basa sulla possibilità di generare

[1] In rete esistono numerosi test sulla qualità audio dei MIDI riprodotti da schede audio diverse. Un confronto approfondito si trova all'URL https://youtu.be/YXFYWJ7dbz0, in cui la colonna sonora MIDI del noto videogioco *Doom* viene eseguita da più di 30 *sound card* differenti.

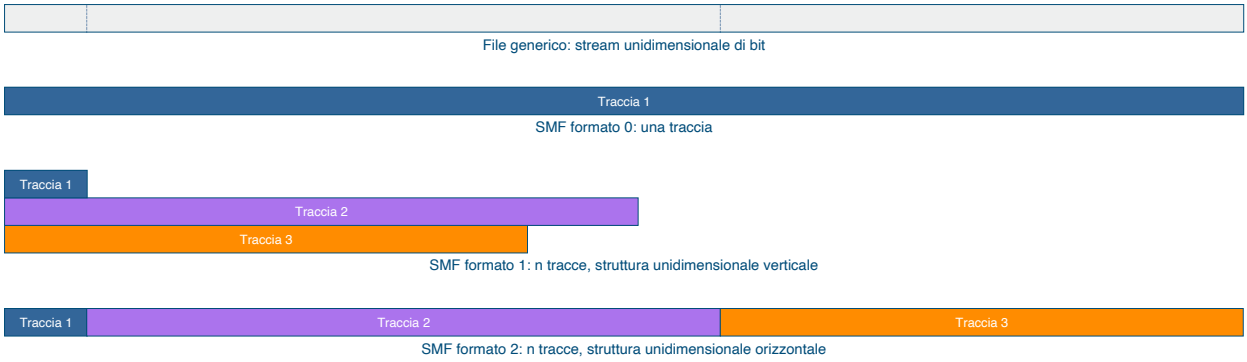


Figura 5.3. Confronto grafico tra SMF in formato 0, 1 e 2.

partiture assai dense sull'asse orizzontale (metrico e ritmico) e verticale (armonico), al punto da trarne il nome; ma, d'altro canto, solo l'esecuzione da parte di un sistema di elaborazione, composto dal *sequencer* e da uno o più sintetizzatori, può portare a un risultato audio fruibile da parte degli ascoltatori.

5.2 Il formato SMF

Rispetto al protocollo MIDI, in cui gli eventi sono rappresentati attraverso messaggi inviati in modo seriale e – per quanto possibile – istantaneo, il formato dei file MIDI prevede una marcatura temporale. Questa differenza deriva dal fatto che il protocollo MIDI sia stato concepito per performance estemporanee, in cui il gesto deve essere tradotto in suono in tempo reale, mentre il formato SMF deve "storicizzare" tali performance, quindi tenere traccia delle distanze temporali tra l'occorrenza dei diversi eventi.

Le specifiche elencano 3 tipi di file MIDI:

- **formato 0**. Tutti i messaggi e i metadati vengono salvati all'interno di un'unica traccia, indipendentemente dal canale di afferenza. L'informazione di canale è comunque preservata all'interno dei messaggi, quindi un *synth* sarà in grado di gestire correttamente i messaggi di canale della sequenza;
- **formato 1**. Sono presenti più tracce la cui riproduzione deve essere simultanea. Una scelta tipica è utilizzare la prima traccia per i metadati e le successive tracce per i singoli canali;
- **formato 2**. Sono supportate tracce multiple indipendenti, ciascuna con indicazioni metriche e valori metronomici propri che possono variare durante l'esecuzione del brano. Le tracce contenute nel file si dispongono una dopo l'altra. Questo formato può contenere più *song* MIDI all'interno di un unico file.

Un file MIDI contiene uno stream di dati binari, necessariamente unidimensionale, con i byte numerati da 0 in senso crescente. Nell'ottica dei formati SMF, però, questa sequenza afferisce a un'unica traccia o si ripartisce su più tracce, generando dal punto di vista logico tre differenti strutturazioni dell'informazione. In riferimento alla Figura 5.3, il formato 1 può essere visto come una forma unidimensionale in senso verticale, mentre il formato 2 come una forma unidimensionale in senso orizzontale.

Solo i primi due formati hanno conosciuto una significativa diffusione. I file in formato 0 sono in generale più piccoli, in quanto comportano un minore *overhead* rispetto alla gestione di tracce multiple. Per questo motivo erano considerati preferibili ai file in formato 1 quando l'occupazione di memoria e la larghezza di banda delle reti costituivano criticità. Il formato 1, invece, può essere visualizzato e modificato in modo più semplice e diretto. Il formato 2 non ha mai trovato particolare

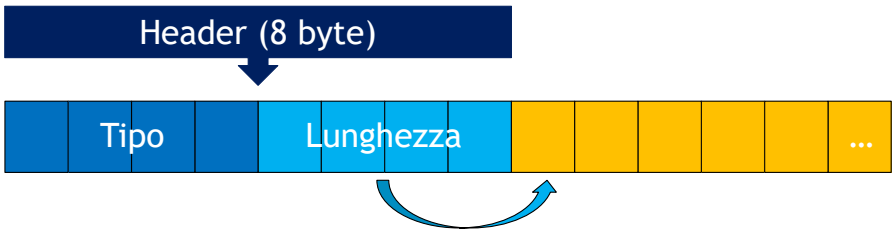


Figura 5.4. Struttura di un generico *chunk* in cui la parte blu (lunga 8 byte) identifica l'intestazione, e la parte arancione (di lunghezza variabile ed espressa negli ultimi 4 byte dell'intestazione) rappresenta il blocco di dati associati.

supporto all'interno dei *sequencer*, ed è stato pensato originariamente per non meglio precisate applicazioni extra-musicali.

L'estensione standard per i file MIDI è .mid. Alcuni software, tra cui *sequencer* quali Digital Performer, Cubase e Sonar, utilizzano in alternativa l'estensione .smf.

Riguardo al *media type*, identificatore binomio per i formati dei file e dei contenuti trasmessi via Internet, SMF è associato a `audio/x-midi` o `audio/mid`. I *media type* sono assegnati e standardizzati dall'*Internet Assigned Numbers Authority* (IANA), e prima del 2018 erano noti con l'acronimo MIME (*Multipurpose Internet Mail Extensions*).

Un'osservazione riguardo al rapporto tra Standard MIDI Files e General MIDI: si tratta di due specifiche indipendenti, che hanno seguito percorsi di progettazione e sviluppo diversi. Nel documento di specifica del 1996, che include entrambe le estensioni, la parte sugli Standard MIDI files precede quella sul General MIDI e non fa mai riferimento ad essa. Ricordando però che l'obiettivo di un file MIDI è "storicizzare" una data performance, distribuirla e renderla fruibile in modo simile per tutti gli utenti, l'adozione del General MIDI assume grande rilievo grazie alla standardizzazione di una serie di aspetti legati a timbri, polifonia, e via dicendo. In altre parole, un file MIDI può essere eseguito anche da un *setup* di dispositivi che non supportano General MIDI, ma il risultato della performance sarà assai meno prevedibile.

5.3 Organizzazione in *chunk*

Un file MIDI si presenta come una sequenza di byte organizzati in unità informative dette *chunk*. Si tratta di blocchi di memoria con una struttura prefissata. Come mostrato nella Figura 5.4, un *chunk* presenta:

- un'**intestazione** (*header*) di dimensione fissa pari a 8 byte, di cui
  - i primi 4 byte identificano il tipo di *chunk*;
  - i restanti 4 byte codificano la dimensione in byte del blocco di dati che segue l'intestazione;
- un **blocco di dati** (*data*) di dimensione variabile, che ospita il vero e proprio contenuto informativo del *chunk*.

Essendo disponibili al più 4 byte per esprimere la dimensione del blocco dei dati, ed essendo il byte l'unità di misura, un *chunk* avrà dimensione complessiva massima di 8 byte (intestazione) + $2^{4\times8}$ byte (dati) = 4.294.967.304 byte.

In ambito informatico esistono numerosi formati basati su *chunk*. Un esempio è il formato WAV per l'audio digitale, che è un sottoinsieme delle specifiche RIFF di Microsoft per la memorizzazione di file multimediali. In un file .wav, il primo *chunk* ha i primi 4 byte posti a $52\,49\,46\,46_{16}$ (ossia la codifica ASCII della sigla `RIFF` in notazione *big-endian*), seguito da 4 byte che denotano la lunghezza in byte della parte restante del file in notazione *little-endian*. Altri casi notevoli di formati basati su una strutturazione in *chunk* sono AIFF, AVI, PNG e RealMedia.

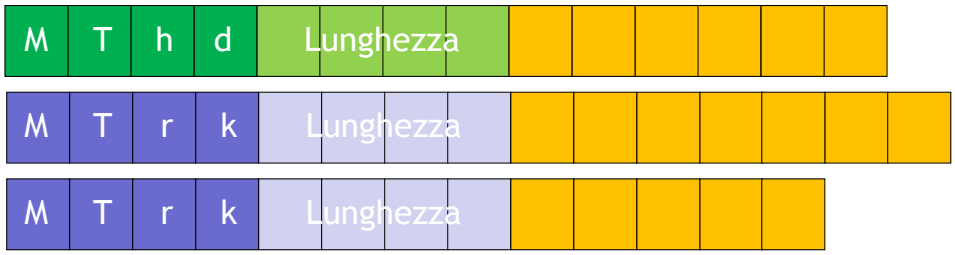


Figura 5.5. Struttura di un file MIDI, in cui un *chunk* di intestazione (in verde) è seguito da uno o più *chunk* di traccia (in lilla).

Il formato dei file MIDI presenta un'architettura che richiama l'*Interchange File Format* (IFF) creato da Electronic Arts nel 1985, da cui deriveranno come diretta emanazione i formati AIFF, GIF, TIFF, PNG e RIFF, e, indirettamente a partire da RIFF, anche WAV, AVI e altri formati per l'interscambio di risorse multimediali. Un file MIDI, però, si differenzia rispetto a un generico file IFF in quanto non consente l'annidamento di *chunk* e non richiede che la dimensione del *chunk* in byte sia dispari.

Un file MIDI è costituito da un *chunk* di intestazione (*MIDI Track header chunk*) seguito da uno o più *chunk* di traccia (*MIDI Track chunk*). Questa strutturazione è mostrata graficamente nella Figura 5.5, ove il colore verde identifica il *chunk* di intestazione e il colore lilla i successivi *chunk* di traccia. Il numero di riquadri di colore arancio nella figura è puramente indicativo.

Nel caso di file in formato 0, si ha un'unica traccia, e dunque un solo *chunk* di traccia; nel caso di file in formato 1, la situazione più comune è avere n *chunk* di traccia con $n > 1$, ma il caso $n = 1$ è comunque ammesso.

5.3.1 *Chunk* di intestazione

Il *chunk* di intestazione, che nelle specifiche MIDI prende il nome *MIDI Track header (MThd) chunk*, codifica le impostazioni generali del file MIDI che si applicano all'intera *song*. Nel formato MIDI questa struttura dati ha una lunghezza fissa pari a 14 byte. Il *chunk* si compone, in particolare, di:

- 4 byte per l'identificazione del tipo di *chunk* (parte di intestazione). Vengono posti a $4D\ 54\ 68\ 64_{16}$, ossia `MThd` in codifica ASCII ;
- 4 byte per la dimensione dei dati associati al *chunk* (parte di intestazione). Trattandosi di un *chunk* di lunghezza fissa che nel complesso occupa 14 byte, il blocco di dati sarà lungo $14 - 8 = 6$ byte; per tale motivo questi 4 byte vengono invariabilmente posti a $00\ 00\ 00\ 06_{16}$;
- 2 byte per l'identificazione del formato di file MIDI (parte dei dati). Sebbene esistano solo 3 formati, per cui sarebbero sufficienti 2 bit, le specifiche prevedono di esprimere il valore su 2 byte, posti a $00\ 0x_{16}$, in cui x è una delle cifre esadecimali 0, 1 o 2;
- 2 byte per il numero di tracce (parte dei dati). Si tratta del numero $nn\ nn_{16}$ di *chunk* di traccia presenti nel seguito del file. Avendo riservato 2 byte a questo valore, il numero massimo di tracce in un file MIDI è pari a $2^{2\times8} = 65.536$. Nel caso di file MIDI di formato 0, per mantenere la coerenza i due byte dovranno valere $00\ 01_{16}$;
- 2 byte per la risoluzione temporale (parte dei dati). Questo valore, espresso su 2 byte, permette di determinare la granularità per la temporizzazione degli eventi all'interno del file MIDI, in particolare nei successivi *chunk* di traccia. L'argomento verrà approfondito ed esemplificato nel seguito.

I valori numerici sono rappresentati con l'ordine di byte detto *big endian*, in cui la memorizzazione/trasmissione inizia dal byte più significativo (estremità alta), per finire col meno

significativo. Nel caso di una parola a 16 bit, il numero 123_{16} viene codificato in *big endian* come 01 23_{16}, mentre in modalità *little endian* risulta 23 01_{16}.

Risoluzione temporale

Vista la sua fondamentale importanza per la temporizzazione degli eventi, la risoluzione temporale tt tt_{16}(detta *division*) merita un apposito approfondimento.

Lo standard supporta due tipi di risoluzione: *metrical time* e *timecode-based time*. A livello di codifica i due approcci si caratterizzano per il valore del bit più significativo di tt tt_{16}, posto a 0 nel primo caso e a 1 nel secondo.

Nel caso di risoluzione *metrical time*, i restanti 15 bit rappresentano il numero di *ticks* in cui viene suddivisa la pulsazione MIDI, che convenzionalmente viene fatta coincidere con la figura ritmica del quarto. Per tale motivo il valore è detto *Pulse per Quarter Note* (*PPQN*), ossia numero di impulsi per quarto. Si tratta dell'approccio più utilizzato da parte dei *sequencer* e degli *editor* digitali di partitura nella generazione di file MIDI.

Se invece il bit più significativo è posto a 1, la risoluzione temporale è detta *timecode-based* e fa riferimento a come si suddivide l'unità di tempo del secondo, in modo consistente con SMPTE e MIDI Time Code. I bit di indice da 14 a 8 contengono i valori -24, -25, -29, -30, espressi in complemento a due, che corrispondono agli standard SMPTE e MTC; il valore -29 identifica la frequenza 30 drop frame. Il byte meno significativo, i cui bit hanno indice da 7 a 0, rappresenta la risoluzione all'interno di un frame.

I due metodi si rifanno dunque a due tecniche differenti per rappresentare la sincronizzazione. Nel caso di risoluzione *metrical time*, il valore è relativo alla pulsazione, la cui durata in termini temporali assoluti può essere calcolata solo disponendo dell'informazione di bpm; al variare del bpm, la risoluzione in termini assoluti può diventare più fine o più grezza. Nel caso *timecode-based*, il riferimento è invece il tempo misurato in unità assolute.

Esempi di risoluzione *metrical time*

Come detto, questo approccio permette di esprimere intervalli temporali in riferimento alla suddivisione della pulsazione da un quarto. Sebbene non si sia ancora fatto cenno alla necessità di temporizzazione in un file MIDI, questione demandata ai *chunk* di traccia, è lecito attendersi che la dimensione temporale entri in gioco per esprimere la distanza tra eventi, ad esempio tra il messaggio di accensione (NOTE-ON) e di spegnimento (NOTE-OFF) di una nota.

Il meccanismo del *PPQN* consente all'autore del file MIDI di scegliere il livello di granularità temporale più idoneo alle proprie esigenze, codificando tale valore nell'intestazione del file MIDI stesso. Se, ad esempio, tt tt = 00 60_{16} = 96, per rappresentare un intervallo temporale di un quarto tra due eventi successivi il numero di *ticks* dovrà essere $n = 96$, e $n = 48$ per eventi che distano un ottavo. Il modo in cui sono rappresentati tali valori numerici all'interno dei *chunk* di traccia verrà illustrato nel prossimo paragrafo.

Per calcolare la durata di un *tick* in unità di tempo assolute, è necessario conoscere tanto il valore b del bpm (numero di pulsazioni da un quarto al minuto) quanto il valore p di *PPQN* (risoluzione in *ticks* per pulsazione da un quarto). Sia d_{ms} la durata in millisecondi del *tick* che si vuole calcolare, e sia d_s la sua durata in secondi. Per definizione, b rappresenta quante pulsazioni hanno luogo in 60 s = 60.000 ms, e $b \times p$ rappresenta il numero di *ticks* in 60.000 ms. Pertanto:

$$d_{ms} = (60.000/(b \times p)) \text{ ms},$$

$$d_s = (60/(b \times p)) \text{ s}.$$

Ad esempio, per $b = 60$ e $p = 20$ si ottiene $d = 0,05$ s, quindi un quarto dura $0,05 \times 20 = 1$ s. Invece, per $b = 120$ e $p = 25$ si ottiene $d = 0,02$ s, e – coerentemente con il raddoppio del bpm – un quarto dura $0,02 \times 25 = 0,5$ s.

5.3.2 *Chunk* di traccia

Il *chunk* di traccia, che prende il nome *MIDI Track (MTrk) chunk*, contiene una sequenza di dati MIDI che si riferiscono principalmente alla performance. Questa struttura dati, per ovvi motivi, non può avere una lunghezza predeterminata, dipendendo la dimensione del blocco di dati dalla quantità e dal tipo di informazioni che reca. Si può riconoscere comunque la strutturazione tipica del *chunk*, che contempla:

- 4 byte per l'identificazione del tipo di *chunk* (parte di intestazione), posti a $4D\,54\,72\,6B_{16}$, ossia `MTrk` in codifica ASCII;
- 4 byte per la dimensione dei dati associati al *chunk* (parte di intestazione). Trattandosi di un *chunk* di lunghezza variabile, non è possibile conoscerne a priori il valore;
- n byte per la codifica di eventi relativi alla traccia (parte di dati).

All'interno del blocco di dati, gli eventi sono sempre rappresentati come coppie costituite dal *delta time* (da qui in avanti abbreviato come Δt), distanza temporale relativa all'evento precedente, e dalla codifica del nuovo evento (*event*). L'intera parte di dati per un *chunk* di traccia è costituita da una sequenza di coppie $\{\Delta t, \textit{event}\}$.

Il Δt si esprime in funzione del valore di *PPQN* specificato nel *chunk* di intestazione. All'interno della coppia $\{\Delta t, \textit{event}\}$ questa informazione non può essere omessa, ma il suo valore può essere posto a 0: un Δt nullo implica distanza 0 rispetto all'evento precedente (simultaneità) o, in mancanza di esso, rispetto al riferimento temporale di inizio traccia. Quindi, al contrario di quanto avviene in una performance MIDI, in cui due eventi – pur generati in modo perfettamente sincrono – vengono inviati in serie sul canale trasmissivo, all'interno di un file MIDI è possibile, dal punto di vista logico, rappresentare eventi simultanei. Va però sottolineato che tali eventi, una volta inviati come messaggi sul canale trasmissivo MIDI, torneranno a essere gestiti in modo seriale.

Si osservi che sarebbe limitativo prefissare un numero di byte per la rappresentazione del Δt. Due eventi possono risultare anche molto distanziati tra loro in termini di *PPQN*: si pensi a una nota di pedale o di bordone tenuta per n battute, il cui messaggio di NOTE-OFF segue il rispettivo NOTE-ON a distanza di $p \times q \times n$ *ticks*, ove p è il numero di *ticks* in un quarto e q è il numero di quarti all'interno della battuta. Un esempio ancora più lampante è la distanza temporale tra eventi per uno strumento che, in una parte orchestrale, presenta molte battute d'aspetto. Per questo motivo il valore di Δt presenta una codifica a lunghezza variabile, descritta nel seguito del paragrafo.

Riguardo agli **eventi** nella coppia $\{\Delta t, \textit{event}\}$, esistono 3 possibilità:

1. *MIDI event*. Si tratta di uno dei messaggi di canale elencati nel Capitolo 2, utilizzati per la descrizione della performance vera e propria. Essendo identica la sintassi, questa tipologia non verrà ulteriormente approfondita. In un file MIDI, si può attivare il *running status* (vedi Paragrafo 2.11), che permette di tralasciare l'invio del byte di stato, ma non è possibile omettere la componente Δt della coppia, nemmeno quando questa fosse nulla;
2. *SysEx event*. Rappresenta una sequenza di byte all'interno di una comunicazione esclusiva, aperta dal valore $F0_{16}$. Anche in questo caso si tratta di un argomento già discusso. Per qualsiasi chiarimento si rimanda al Paragrafo 2.8.5;
3. *Meta-event*. Si tratta di un argomento nuovo, in quanto una performance MIDI non prevede l'invio di meta-informazioni sotto forma di messaggi (se non eventualmente sfruttando il meccanismo dei *SysEx*). Ai meta-eventi sarà dedicato il Paragrafo 5.4.

Valori numerici a lunghezza variabile

Nei file MIDI si pone il problema di rappresentare Δt dalla lunghezza variabile, in quanto il valore massimo ammissibile non è predicibile. I numeri vengono dunque rappresentati come sequenze di

Dec	Hex	Bin	Bin var.	Hex var.
0	0	0	(0)0000000	00
64	40	1000000	(0)1000000	40
127	7F	1111111	(0)1111111	7F
128	80	10000000	(1)0000001 (0)0000000	81 00
8.192	20 00	100000 00000000	(1)1000000 (0)0000000	C0 00
2.097.327	20 00 AF	100000 00000000 10101111	(1)0000001 (1)0000000 (1)0000001 (0)0101111	81 80 81 2F

Tabella 5.1. Esempi di rappresentazione di valori in lunghezza variabile.

byte *big endian*, in cui il primo bit funge da identificatore e i restanti 7 bit contengono il *payload*. Il valore numerico, se necessario, viene ripartito su più byte; per i valori compresi tra 0 e 127, essendo questi rappresentabili su 7 bit, è sufficiente un unico byte, e non si coglie differenza tra la consueta codifica a lunghezza fissa e quella variabile.

Nel caso più generale, la rappresentazione in base 2 del valore viene ripartita in aggregazioni di 7 bit, aggiungendo zeri in posizione più significativa in modo da avere un numero di cifre multiplo di 7. Passando agli ottetti che costituiscono i byte, per ciascuno di essi il ruolo del bit più significativo b è quello di distinguere tutti i byte che trasportano una parte del valore numerico ($b = 1$) da quello meno significativo ($b = 0$).

In pratica, quando un sistema di elaborazione valuta un *chunk* di traccia MIDI e si aspetta il Δt della coppia $\{\Delta t, event\}$, al fine di ricostruirne il valore esso prende in considerazione tutti i byte in sequenza che incontra fino al primo byte che ha $b = 0$ come bit più significativo. Questa condizione potrebbe essere soddisfatta anche dal primo byte, caso tutt'altro che raro: si pensi agli eventi simultanei, il cui Δt è 0, o agli eventi che distano tra loro di un numero di *PPQN* inferiore a 128. Una volta disponibili tutti i byte della codifica a lunghezza variabile, il valore può essere ricostruito prendendo in considerazione per ciascuno di essi i 7 byte meno significativi.

Esaurita la valutazione della componente Δt, il sistema si aspetta un evento appartenente a uno dei tre tipi sopra menzionati, la cui dimensione può essere facilmente ricavata dalla sintassi dell'evento stesso. A seguire, se la traccia non è conclusa, il sistema si aspetterà un nuovo valore di Δt, e così via.

A titolo di esempio, nella Tabella 5.1 sono riportati alcuni valori in base 10, la loro rappresentazione in base 16 e in base 2 (con il minimo numero di cifre richieste), la loro codifica in base 2 a lunghezza variabile (con il minimo numero di byte richiesti), e, infine, la rappresentazione in base 16 a lunghezza variabile, ossia il valore di Δt nella codifica esadecimale propria dei *chunk*.

Si può notare che, per i valori minori di 128, la rappresentazione esadecimale standard e quella a lunghezza variabile coincidono; infatti tutti i bit mancanti per completare l'ottetto, compreso quello più significativo, vengono sostituiti da zeri.

Volendo limitare il numero di byte impiegabili a 4, ossia a 32 bit, il massimo valore rappresentabile è una sequenza di $7 \times 4 = 28$ bit posti a 1, che equivale a 268.435.455 in base 10, e a FFFFFFF in base 16; preponendo opportunamente gli 1 e gli 0, questo valore ha una rappresentazione esadecimale a lunghezza variabile pari a FF FF FF 7F. Sebbene sia chiaramente possibile usare un numero qualsiasi di byte per una codifica a lunghezza variabile, i software devono opportunamente dimensionare le parole di memoria per contenerne la rappresentazione binaria, e 32 bit sono normalmente considerati sufficienti: infatti il valore di 2×10^8 *ticks* con *PPQN* = 96 *ticks* per pulsazione e al tempo metronomico (estremamente sostenuto) di 500 bpm equivarrebbe al tempo di circa 3 giorni, decisamente sovradimensionato come Δt all'interno di una performance musicale.

Informazioni di inizio traccia

Un file MIDI, all'interno dei *chunk* di traccia, non contiene solo dati di performance opportunamente temporizzati (NOTE-ON, NOTE-OFF, PITCH BEND CHANGE, ecc.), ma anche informazioni aggiuntive delegate ai meta-eventi (note sul copyright, compositore, ecc., descritte nel Paragrafo 5.4) e dati di *setup* aggiuntivi che permettono di avere un maggior controllo sull'esecuzione da parte di un sintetizzatore.

Ad esempio, in mancanza di indicazione esplicita, l'associazione tra canale e timbro potrebbe risultare arbitraria. A questo riguardo, le specifiche suggeriscono per ciascun canale MIDI di prendere in considerazione alcuni messaggi la cui collocazione temporale dovrebbe precedere l'invio di dati di performance:

- Bank Select (valore di default = 0) e Program Change;
- Reset All Controllers;
- Initial Volume (CC7) (livello di default = 100);
- Expression (CC11) (livello iniziale = 127);
- Hold pedal (valore 0 = off);
- Pan (valore centrale = 64);
- Modulation (valore iniziale = 0);
- Pitch bend range;
- Reverb (valore 0 = off);
- Chorus level (valore 0 = off).

Sebbene la collocazione temporale più logica per i messaggi menzionati sia all'inizio della traccia, quindi ciascuno di essi debba prevedere $\Delta t = 0$, va evidenziato che la gestione simultanea di un numero considerevole di eventi (considerando anche la necessità di replicarli sui vari canali) potrebbe causare il problema del *MIDI choking*. Per questo motivo le specifiche suggeriscono di prendere in considerazione una possibile separazione degli eventi di 5–10 *ticks*.

5.4 Meta-eventi

Le specifiche SMF contemplano una serie di cosiddetti *meta-eventi*, il cui scopo è definire alcune proprietà aggiuntive rispetto alla performance MIDI. Non è necessario che i software supportino – totalmente o parzialmente – la gestione dei meta-eventi, in quanto i messaggi MIDI sono sufficienti a ricostruire la performance in modo completo.

Esattamente come i messaggi MIDI, all'interno dei *chunk* di traccia anche i meta-eventi vengono rappresentati e temporizzati attraverso coppie $\{\Delta t, \textit{event}\}$.

La sintassi di un generico meta-evento, espressa in esadecimale, prevede:

- `FF` come byte di apertura di un generico meta-evento. Si osservi che nel protocollo MIDI questo stesso valore è associato al messaggio SYSEM RESET (Paragrafo 2.9.4);
- `xx` come byte di identificazione dello specifico meta-evento;
- `len` per esprimere la lunghezza in byte della parte restante del meta-evento. Si tratta di un'informazione particolarmente significativa nei casi in cui la lunghezza non sia predicibile. Tale valore non si esprime necessariamente su un singolo byte, ma potrebbe richiedere la codifica a lunghezza variabile già descritta nel Paragrafo 5.3.2.

I meta-eventi che all'interno di una traccia risultano simultanei (ossia separati tra loro da Δt nullo) possono essere gestiti in qualsiasi ordine. Se è presente informazione di copyright, le specifiche raccomandano di posizionarla quanto prima nel file, in modo da renderla immediatamente evidente. Come ulteriore suggerimento, il numero di sequenza e il nome della sequenza e della traccia, se specificati, dovrebbero essere posizionati all'istante 0. Ogni traccia deve necessariamente concludersi con un meta-evento di fine traccia.

5.4.1 Numero di sequenza

Il meta-evento Numero di sequenza (*Sequence Number*), opzionale e identificato dal codice `00`, si presenta nella forma:

```
FF 00 02 ss ss
```

e serve a specificare il numero di sequenza. Se presente, esso deve posizionarsi all'inizio della corrispettiva traccia, ossia deve precedere qualsiasi evento a Δt non nullo e qualsiasi evento MIDI, anche se posto a $\Delta t = 0$.

In un file di tipo 0 o 1, questo meta-evento dovrebbe trovarsi nella prima traccia (che per il tipo 0 è anche l'unica). In un file di tipo 2, esso viene usato per identificare ciascun pattern. Se l'identificativo viene omesso, ne fa le veci la posizione della sequenza nell'ordine in cui si trova nel file MIDI.

5.4.2 Testo generico

Il meta-evento Testo (*Text Event*), identificato dal codice `01`, si presenta nella forma

```
FF 01 len text
```

La parte `len` quantifica, attraverso un valore a lunghezza variabile, il numero di byte richiesto dal testo che segue. Per agevolare l'interscambio informativo, per i caratteri del testo viene utilizzata la codifica ASCII, per cui ogni carattere occupa un byte.

Le specifiche indicano come il meta-evento possa genericamente essere usato per descrivere informazione testuale, senza indicarne in modo più specifico la funzione (ad esempio, liriche, definizione di punti di ancoraggio o altro). Viene però proposto di posizionare il meta-evento all'inizio di una traccia per definirne il nome, e in particolare l'orchestrazione.

I meta-eventi il cui codice va da `01` a `0F`, descritti nel seguito, sono anch'essi riservati a informazioni testuali, ma con obiettivi ben precisi [2].

5.4.3 Avviso di copyright

Il meta-evento Avviso di copyright (*Copyright Notice*), identificato dal codice `02`, si presenta nella forma

```
FF 02 len text
```

e serve a codificare informazioni relative ai diritti. Il testo, espresso in codifica ASCII, dovrebbe contenere i caratteri `(C)`, l'anno e il titolare dei diritti. Nel caso il file MIDI contenga più brani, tutte le informazioni di copyright andrebbero inserite in un unico meta-evento posto all'inizio del file, ossia nel primo *chunk* di traccia, all'istante 0 e con precedenza su qualsiasi evento MIDI.

5.4.4 Nome di sequenza/traccia

Il meta-evento Nome di sequenza/traccia (*Sequence/Track Name*), identificato dal codice `03`, si presenta nella forma

```
FF 03 len text
```

e, se posto in un file di tipo 0 o nella prima traccia di un file di tipo 1, è pensato per contenere il nome della sequenza; negli altri casi, tale meta-evento descrive il nome della traccia.

5.4.5 Nome dello strumento

Il meta-evento Nome dello strumento (*Instrument Name*), identificato dal codice `04`, si presenta nella forma

```
FF 04 len text
```

ed è pensato per fornire una descrizione testuale del tipo di strumentazione. Considerando che una traccia può contenere messaggi di performance afferenti a più canali, le specifiche suggeriscono

[2] In realtà, nelle specifiche l'ultimo meta-evento di testo a essere descritto ha il codice identificativo `07`.

di sfruttare il meta-evento Prefisso di canale MIDI (Paragrafo 5.4.9) per specificare il canale di riferimento, oppure per inserire l'informazione di canale direttamente nel testo del meta-evento.

5.4.6 Testo cantato

Il meta-evento Testo cantato (*Lyric*), identificato dal codice `05`, si presenta nella forma

```
FF 05 len text
```

e contiene il testo da cantare a un dato istante di tempo. Tipicamente le liriche vengono suddivise in sillabe, ciascuna affidata a un meta-evento diverso. Non è strettamente richiesta, dal punto di vista sintattico, corrispondenza tra un meta-evento e un evento MIDI di performance quale NOTE-ON. Normalmente, però, il testo cantato viene sincronizzato con la musica corrispondente; questo risultato si ottiene attraverso l'uso di Δt nulli rispetto all'accensione delle note.

5.4.7 Marcatore

Il meta-evento Marcatore (*Marker*), identificato dal codice `06`, si presenta nella forma

```
FF 06 len text
```

e normalmente si posiziona nella prima traccia di un file in formato 1 o nell'unica traccia disponibile in un file di tipo 0. Il meta-evento serve ad attribuire un'etichetta a un determinato punto della performance, come nel caso delle lettere di chiamata (A, B, ecc., dette in inglese *rehearsal marks*) o dei nomi di sezione ("prima strofa", "ritornello", ecc.).

5.4.8 Punto di sincronizzazione

Il meta-evento Punto di sincronizzazione (*Cue Point*), identificato dal codice `07`, si presenta nella forma

```
FF 07 len text
```

e fornisce la descrizione di un evento temporizzato che deve aver luogo in un video, sul palcoscenico o in situazioni paragonabili (ad esempio, "l'oggetto cade di mano", "viene aperta la tenda", ecc.).

5.4.9 Prefisso di canale MIDI

Il meta-evento Prefisso di canale MIDI (*MIDI Channel Prefix*), identificato dal codice `20`, si presenta nella forma

```
FF 20 01 cc
```

La lunghezza complessiva del meta-evento, in questo caso, è prefissata e vale 4 byte. Parimenti si può constatare che il suo *payload* è di 1 byte, come specificato dal terzo byte.

L'obiettivo del meta-evento è specificare nel byte `cc` il canale MIDI cui si riferiscono tutti gli eventi che seguono, compresi i *SysEx* e i meta-eventi. Questa informazione rimane in vigore fino a un successivo meta-evento dello stesso tipo o a un messaggio MIDI con informazione di canale.

L'utilità del meta-evento è evidente quando messaggi di tracce differenti vengono convogliati all'interno di un'unica traccia, come nel caso di file MIDI di tipo 0.

5.4.10 Porta MIDI

Il meta-evento Porta MIDI (*MIDI Port*), identificato dal codice `21`, si presenta nella forma

```
FF 21 01 pp
```

ed è un evento opzionale e deprecato (ma tuttora ampiamente in uso) che specifica il numero di porta MIDI da cui emettere i messaggi MIDI. Ricordando la limitazione di 16 canali per porta, questa possibilità estende il numero di canali disponibili supportando dispositivi multi-porta. Il byte `pp` è il numero di porta, mentre il valore 0 rappresenta convenzionalmente la prima porta disponibile nel sistema.

Si tratta di un meta-evento che si colloca a inizio traccia, prima della lettura di qualsiasi messaggio MIDI di performance.

5.4.11 Fine della traccia

Il meta-evento Fine della traccia (*End of Track*), identificato dal codice `2F`, si presenta nella forma

```
FF 2F 00
```

e la sua presenza è strettamente richiesta dalle specifiche SMF per definire il punto di terminazione della traccia. Questa informazione è particolarmente utile nel caso di concatenazione di più tracce o di *looping*.

Si osservi che il terzo byte, normalmente dedicato a specificare la lunghezza del contenuto informativo del meta-evento, ha valore 0. Infatti l'intera informazione è veicolata dalla semplice presenza del meta-evento, senza necessità di specificare ulteriori valori.

5.4.12 Tempo

Il meta-evento Tempo (*Set Tempo*), identificato dal codice `51`, si presenta nella forma

```
FF 51 03 tt tt tt
```

e ha l'obiettivo di determinare la durata in microsecondi della pulsazione di un quarto (semiminima). Un altro modo di esprimere lo stesso concetto è dire che 1/24 di tale valore rappresenta la durata di un *MIDI clock* in microsecondi, visto che in una pulsazione da un quarto si trovano 24 *MIDI clocks*.

Questo meta-evento, che consente di specificare il metronomo iniziale e di determinarne eventuali cambi durante la performance, permette la sincronizzazione con protocolli di temporizzazione quali SMPTE e MIDI Time Code (Paragrafo 2.8.1). Teoricamente il meta-evento dovrebbe essere specificato in corrispondenza dei *MIDI clocks*, per dare maggiori garanzie di compatibilità con i dispositivi sincronizzati.

In mancanza di un'esplicita codifica, il tempo di un file MIDI è 120 bpm. Nei file in formato 0, questo meta-evento dovrebbe trovarsi all'inizio dell'unica traccia, in formato 1 nella prima traccia e in formato 2 all'inizio di ciascuna sezione indipendente.

5.4.13 Scostamento SMPTE

Il meta-evento Scostamento SMPTE (*SMPTE Offset*), identificato dal codice `54`, si presenta nella forma

```
FF 54 05 hh mm ss ff xx
```

Questo evento designa l'istante temporale SMPTE a cui si vuole far iniziare il *chunk* di traccia. Se presente, esso dovrebbe collocarsi a inizio traccia, prima di qualsiasi evento a
dt non nullo e prima di qualsiasi evento MIDI, anche a Δt nullo.

I byte `hh`, `mm`, `ss` e `ff` si riferiscono a ore, minuti, secondi e frame SMPTE, e la codifica è la stessa prevista da MIDI Time Code, incluse le convenzioni sulla rappresentazione delle ore. Il parametro `xx` specifica una parte frazionaria di deviazione rispetto ai fotogrammi, in termini di centesimi di frame.

Se il file MIDI è di tipo 1, l'informazione di *offset* dovrebbe trovarsi nella sola traccia dedicata alla mappatura del tempo.

5.4.14 Indicazione di tempo

Il meta-evento Indicazione di tempo (*Time Signature*), identificato dal codice `58`, si presenta nella forma

```
FF 58 04 nn dd cc bb
```

e permette di specificare l'indicazione metrica attraverso 4 valori numerici, come precisato dal terzo byte. In particolare, `nn` e `dd` sono, rispettivamente, il numeratore e il denominatore della frazione. Quest'ultimo, però, viene annotato come esponente della potenza negativa di 2: quindi un denominatore 0 rappresenta l'intero ($2^0 = 1/1$), 1 la metà ($2^{-1} = 1/2$), 2 il quarto ($2^{-2} = 1/4$), 3 l'ottavo ($2^{-3} = 1/8$), e così via.

Il parametro `cc` restituisce il numero di *MIDI clocks* per ogni impulso di metronomo, il che permette di specificare che il metronomo scandisce una figura ritmica differente rispetto a quella della frazione. Ad esempio, nel tempo di 4/4, il metronomo potrebbe battere le metà.

Il parametro `bb` esprime il numero di trentaduesimi che trovano posto nella pulsazione che ha luogo ogni 24 *MIDI clocks*; si tratta dell'intervallo temporale che viene comunemente associato alla figura da un quarto. La presenza di `bb` potrebbe sorprendere, in quanto dovrebbero essere sempre 8 le figure ritmiche da un trentaduesimo all'interno di un quarto. Però l'associazione della figura ritmica da un quarto alla pulsazione che ha luogo ogni 24 *MIDI clocks* è arbitraria, quindi `bb` consente di mantenere la compatibilità con altri sistemi – presumibilmente precedenti al rilascio delle specifiche – in cui la pulsazione da 24 MIDI *clocks* veniva associata a valori ritmici differenti.

Nei file in formato 0, questo meta-evento dovrebbe trovarsi all'inizio dell'unica traccia, in formato 1 nella prima traccia e in formato 2 all'inizio di ciascuna sezione indipendente.

In assenza di indicazione diversa, il metro di un file MIDI è 4/4.

Esempio

Si voglia esprimere attraverso un meta-evento il tempo di 6/8, in cui il metronomo scandisce i quarti col punto (una pulsazione ogni 3 ottavi) e non viene modificata l'associazione di 24 *MIDI clocks* per quarto. Il meta-evento risulterebbe:

```
FF 58 04 06 03 24 08
```

il che, nel dettaglio, implica:

- `FF` – generico meta-evento;
- `58` – identificativo del meta-evento per l'indicazione di tempo;
- `04` – lunghezza in byte del *payload*;
- `06` – numeratore della frazione;
- `03` – esponente cui elevare a potenza negativa la base 2 per ottenere il denominatore della frazione;
- `24` – numero di *MIDI clocks* per pulsazione scandita dal metronomo. Si presti attenzione al fatto che 24_{16} corrisponde a 36_{10}. Per comprenderne la semantica, questo numero va rapportato al valore 24_{10} che corrisponde alla pulsazione principale;
- 08 – numero di trentaduesimi all'interno della pulsazione principale da 24 *MIDI clocks*, che dunque viene associata al quarto.

5.4.15 Armatura di chiave

Il meta-evento Armatura di chiave (*Key Signature*), identificato dal codice `59`, si presenta nella forma

```
FF 59 02 aa mm
```

ed esprime la tonalità corrente del brano in termini di numero di alterazioni `aa`. La corrispondenza tra valori interi con segno in base 10 e numero di alterazioni in chiave è: 0 = nessuna alterazione, 1 = 1 diesis, 2 = 2 diesis, ..., 7 = 7 diesis, -1 = 1 bemolle, -2 = 2 bemolli, ..., -7 = 7 bemolli. Nel passaggio a esadecimali, i valori negativi sono rappresentati in complemento a 2, per cui -1 = FF_{16}, -2 = FE_{16}, ..., -7 = $F9_{16}$.

Il byte `mm`, infine, è un flag che specifica se la tonalità sia maggiore (00_{16}) o minore (01_{16}).

Si osservi che questa informazione, salvata all'interno dei file MIDI, può aiutare un software di notazione musicale nel ricostruire correttamente l'informazione di altezza e di stato di alterazione partendo dal pitch: nella tonalità di Re maggiore, il pitch 66 verosimilmente corrisponderà a Fa# anziché a Sol♭.

A scopo di esempio, si voglia decodificare il meta-evento `FF 59 02 FF 00`: esso indica la tonalità di Fa maggiore, in quanto `FF` rappresenta 1 bemolle in chiave e `00` seleziona, tra le due scale candidate, la relativa maggiore.

5.4.16 Meta-evento specifico al *sequencer*

Il Meta-evento specifico al *sequencer* (*Sequencer-Specific Meta-Event*), identificato dal codice `7F`, si presenta nella forma

```
FF 7F len data
```

e consente di rappresentare meta-eventi specifici per determinati strumenti software.

Il primo byte di dati, o i primi 3 byte di dati, rappresentano il *manufacturer ID*, in accordo con quanto visto per i messaggi SysEx (Paragrafo 2.10.1). La parte rimanente del *payload* dipende dalle scelte progettuali e implementative del singolo *sequencer*.

5.5 Esempi

Verranno ora presentati alcuni esempi, sintetici ma completi, generati a partire da notazione musicale trascritta con diversi software ed esportata in formato SMF. Per ciascun esempio verrà mostrata la partitura originale e la visualizzazione esadecimale del file binario prodotto, con indici di riga in base 16 e trascrizione ASCII sul lato destro.

5.5.1 Come segmentare un file MIDI

In questa sezione si richiamano alcune regole pratiche per l'analisi di un generico file MIDI, partendo dalla visualizzazione del suo contenuto binario offerta da un viewer esadecimale.

Il primo passaggio consiste nel riconoscimento dei *chunk* di cui il file si compone. Come detto, il *chunk* è una generica struttura dati che trova applicazione in molteplici formati. Nel caso del formato SMF, un file valido si compone di una sequenza di *chunk*. In particolare, il primo di questi è sempre un *chunk* di intestazione di traccia, seguito da un numero strettamente positivo di *chunk* di traccia. L'inizio dei *chunk* può essere facilmente riconosciuto cercando all'interno del file MIDI le sequenze esadecimali che fungono da identificativi dei due tipi ammessi:

- $4D\,54\,68\,64_{16}$. Questa sequenza identifica l'inizio del primo *chunk*, che è necessariamente di intestazione, pertanto corrisponde alla stringa "MThd" ed è presente sempre e solo nei byte 0-3 di un generico file MIDI;

- 4D 54 72 $6B_{16}$. Questa sequenza corrisponde alla stringa “MTrk” e compare almeno una volta all’interno del file MIDI.

Inoltre, ogni *chunk* di traccia viene concluso dal meta-evento di fine della traccia FF 2F 00_{16}. Questa considerazione offre un metodo alternativo per identificare i *chunk* di traccia: a parte il primo di essi, che inizia invariabilmente al byte 14 (avendo numerato 0 il primo byte), tutti quelli successivi iniziano immediatamente dopo la sequenza di byte FF 2F 00_{16}.

Una volta riconosciuti i *chunk* di cui si compone il file, si può riconoscere la loro parte di intestazione (byte 0-7 di ciascuna struttura dati), suddividendola ulteriormente nella sottoparte di identificazione del tipo di *chunk* (byte 0-3) e in quella di dimensione del blocco di dati (byte 4-7). Un terzo metodo – meno immediato dei precedenti – per riconoscere l’occorrenza dei vari *chunk* si basa proprio su quest’ultima informazione, ossia la dimensione della parte di *payload*: è necessario ricordare che il valore nei byte 4-7 rappresenta la lunghezza in byte della sola parte di dati del *chunk*, ed è espresso in base 16.

Infine, si possono valutare i contenuti dei *chunk*, ricordando che nel *chunk* di intestazione di traccia i byte vanno letti a coppie e assumono un significato prefissato, mentre nei *chunk* di traccia si devono enucleare le coppie $\{\Delta t$, *event*$\}$ che, nel loro insieme, coprono completamente il blocco dei dati.

I Δt sono espressi in codifica a lunghezza variabile, quindi costituiti da un numero variabile di byte, il cui ultimo ottetto ha bit più significativo posto a 0. Riguardo agli eventi, questi possono essere di soli 3 tipi: eventi MIDI, eventi *SysEx* e meta-eventi. Gli eventi *SysEx* e i meta-eventi iniziano per FF_{16}. Normalmente, gli eventi MIDI iniziano con il *nibble* che li caratterizza nella tabella dei messaggi riassunta nel Paragrafo 2.6, ma bisogna ricordare il potenziale intervento del *running status* (Paragrafo 2.11). Quindi, se dopo l’enucleazione del Δt non ci si ritrova in una delle situazioni elencate, i byte dell’evento vanno interpretati secondo la logica del *running status*, e quindi riferiti all’ultimo byte di stato ricevuto in ordine di tempo.

5.5.2 Esempio 1

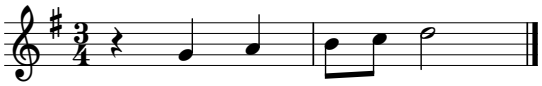


Figura 5.6. Spartito dell’Esempio 1.

```
 0: 4D 54 68 64 00 00 00 06  00 01 00 01 01 E0 4D 54  MThd.........àMT
10: 72 6B 00 00 00 57 00 FF  58 04 03 02 18 08 00 FF  rk...W.ÿX......ÿ
20: 59 02 01 00 00 FF 51 03  07 A1 20 00 B1 79 00 00  Y....ÿQ..¡ .±y..
30: C1 49 00 B1 07 64 00 0A  40 00 5B 00 00 5D 00 00  ÁI.±.d..@.[..]..
40: FF 21 01 00 83 60 91 43  50 83 47 43 00 19 45 50  ÿ!..`CPGC..EP
50: 83 47 45 00 19 47 50 81  63 47 00 0D 48 50 81 63  GE..GPcG..HPc
60: 48 00 0D 4A 50 87 0F 4A  00 01 FF 2F 00           H..JP.J..ÿ/.
```

Figura 5.7. Rappresentazione esadecimale e ASCII del file .mid dell’Esempio 1, generato da *MuseScore*.

Il primo esempio si basa su un semplice tema melodico per flauto trascritto con *MuseScore* 3.4.2 [3] ed esportato in formato MIDI.

A livello di macro-segmentazione del file risultante, in aggiunta al necessario *chunk* di intestazione (byte 0–13), si nota la presenza di un unico *chunk* di traccia (byte 14–108).

[3] *MuseScore* è un programma di notazione musicale WYSIWYG multipiattaforma per Linux, macOS e Microsoft Windows. Si tratta di un applicativo gratuito e distribuito come software open source pubblicato sotto licenza *GNU General Public License*. Il sito web di *MuseScore* è https://musescore.org/it

Chunk di intestazione

Per quanto riguarda il primo *chunk*, si ritrova la strutturazione standard che prevede 4 byte di valore $4D\ 54\ 68\ 64_{16}$, la cui trascrizione ASCII mostrata nella parte destra della Figura 5.7 corrisponde a "MThd"; a seguire, nei byte 4–7, è presente il valore $00\ 00\ 00\ 06_{16} = 6$, che è prefissato per il *chunk* di intestazione MIDI; i restanti 6 byte (byte 8–13), considerati a coppie, identificano il formato di file ($00\ 01_{16}$), il numero di tracce ($00\ 01_{16}$) e il valore di *PPQN* ($01\ E0_{16}$).

Una prima osservazione riguarda il fatto che, pur contenendo il file MIDI una sola traccia (il che è coerente con lo spartito creato), esso sia comunque di tipo 1. All'atto dell'esportazione, *MuseScore* non chiede all'utente che formato di file MIDI utilizzare. Gestendo partiture composte da n strumenti e dunque n tracce, la scelta generale più naturale è un file in formato 1. Uno spartito per strumento solo viene dunque gestito come un caso particolare dello scenario più generico. Sarebbe sufficiente sostituire il valore del byte 9 con 00_{16} per ottenere un file MIDI in formato 0 sintatticamente e logicamente valido.

Il *PPQN* è posto a $01\ E0_{16} = 480$ *ticks* per pulsazione (da un quarto). Si tratta del valore di default assunto dalle impostazioni di *MuseScore*. Il *PPQN* sarà fondamentale per valutare le distanze temporali tra eventi nel *chunk* di traccia.

Chunk di traccia

Per ovvi motivi, anche il *chunk* di traccia presenta un'intestazione da 8 byte. I byte 14–17 contengono il valore $4D\ 54\ 72\ 6B_{16}$ ("MTrk"), mentre i byte 18–21 servono a dimensionare la parte restante del *chunk*: il valore $00\ 00\ 00\ 57_{16}$ in base 10 equivale a 87 byte. Ragionando a ritroso, Figura 5.7 mostra che il file occupa in tutto 109 byte, numerati da 0 a 108; per quanto visto finora, il primo *chunk* richiede 8 + 6 byte, che peraltro è la dimensione prefissata per il *chunk* di intestazione di qualsiasi file MIDI. Prendendo in considerazione gli 8 byte di intestazione del secondo e ultimo *chunk*, la parte rimanente (ossia il blocco di dati) deve occupare 109 - 14 - 8 = 87 byte, il che conferma la coerenza del valore contenuto nei byte 18–21.

Analizzando il blocco dati del *chunk* di traccia, si tratta ora di determinare le coppie $\{\Delta t, event\}$ che descrivono la performance.

Per enucleare il primo Δt, l'algoritmo prevede di analizzare in sequenza i byte fino ad arrivare al primo di essi che abbia il bit più significativo posto a 0. In questo caso,si tratta del primo ottetto binario incontrato (00_{16}), quindi il primo evento avrà luogo a inizio traccia. Si tratta, nella fattispecie, di un meta-evento (FF_{16}) di tipo Indicazione di tempo (58_{16}) che occupa 4 ulteriori byte (04_{16}) e il cui valore si ricostruisce nel seguente modo: numeratore della frazione = 3 (03_{16}), denominatore = 4 (02_{16}, che è l'esponente cui elevare a potenza negativa la base 2), numero di *MIDI clocks* per quarto = 24 (18_{16}, valore standard) e numero di trentaduesimi nella pulsazione = 8 (08_{16}, valore standard).

A seguire si trova un'altra coppia $\{\Delta t, event\}$ posta a inizio traccia (00_{16}) [4] e contenente un meta-evento (FF_{16}). In questo caso,si tratta dell'Armatura di chiave (59_{16}), che codifica l'informazione su 2 byte (02_{16}) dedicati, rispettivamente, al numero di alterazioni in chiave (01_{16} = 1 diesis) e alla caratterizzazione del modo maggiore o minore (00_{16} = scala maggiore). In effetti la tonalità scelta all'atto di impostare la partitura in *MuseScore* era Sol maggiore.

Nel seguito si trova un'ulteriore coppia $\{\Delta t, event\}$, posta a inizio traccia (00_{16}) e contenente un meta-evento (FF_{16}). Si tratta del Tempo (51_{16}), espresso su 3 byte (03_{16}) in termini di numero di microsecondi per pulsazione. Il valore esadecimale $07\ A1\ 20_{16}$ è pari a 500.000 µs = 0.5 s. Poiché la pulsazione da un quarto occupa mezzo secondo, in un minuto si verificheranno 120 pulsazioni; il tempo metronomico, quindi, è 120 bpm.

La successiva coppia $\{\Delta t, event\}$, ancora una volta simultanea al meta-evento precedente (00_{16}) e dunque posta a inizio traccia, riguarda, infine, un evento MIDI. $B1_{16}$ è un CONTROL CHANGE

[4] In realtà *delta-time* = 0 esprime simultaneità rispetto all'evento precedente, che incidentalmente si trova a inizio traccia.

(*nibble* B_{16}) sul Canale 2 (*nibble* 1_{16}), per cui è lecito attendersi due byte di dati: il primo vale 121 (79_{16}) e implica Reset All Controllers, mentre il secondo ha valore 0 (00_{16}) in quanto lo specifico numero di control change non richiede ulteriori dati.

A seguire si verifica un altro evento simultaneo (00_{16}) che riguarda un PROGRAM CHANGE (*nibble* C_{16}) sul Canale 2 (*nibble* 1_{16}). Questo messaggio richiede un unico byte di dati, il cui valore è quello della *patch* da associare al canale. In termini matematici, 49_{16} = 73, ma in General MIDI il primo elemento della tabella è numerato 1 anziché 0. In definitiva, si tratta del *program number* 74, cui General MIDI associa il timbro del flauto.

L'evento successivo, anch'esso posizionato a inizio traccia (00_{16}), è un ulteriore CONTROL CHANGE sul Canale 2 ($B1_{16}$), che stavolta riguarda il volume del Canale (07_{16}) e lo imposta al valore 100 (64_{16}).

A questo punto si verifica una situazione che richiede attenzione. Il primo ottetto binario ha valore 00_{16}, quindi si tratta di un Δt che occupa un solo byte. A seguire, ci si aspetta il primo byte di un evento; il valore $0A_{16}$, però, non rientra nella casistica attesa: non può essere il byte di stato di un messaggio MIDI di canale, né il byte di stato di un SYSEX ($F0_{16}$), né l'avvio di un meta-evento (FF_{16}), tutti casi in cui il bit più significativo sarebbe posto a 1; neppure può trattarsi di un nuovo Δt, in quanto la coppia precedente risulterebbe incompleta della componente dell'evento. Per comprendere la situazione ci si deve rifare alla tecnica *running status*, che prevede che l'ultimo byte di stato ricevuto rimanga in vigore fino alla sua ridefinizione. Alla luce di questa considerazione, i byte successivi vengono interpretati come byte di dati relativi allo stato corrente $B1_{16}$, ossia CONTROL CHANGE sul canale 2. Ricordando la sintassi di questo messaggio, sono 2 i byte di dati attesi: $0A_{16}$ = 10 determina un CC di Pan, cui viene assegnato il valore 40_{16} = 64 (posizione centrale).

Lo stesso scenario si presenta per i byte successivi, sempre caratterizzati da Δt pari a 00_{16}. In particolare, 5B 00_{16} e 5D 00_{16} impostano a 0 i CC 91 e 93, legati, rispettivamente, a effetti di riverbero e di chorus.

L'evento successivo, sempre simultaneo a quelli precedenti e dunque posto a inizio traccia, è caratterizzato dalla sequenza di byte FF 21 01 00_{16}, quindi è un meta-evento di specifica della porta MIDI di uscita, fissata alla prima porta disponibile nel sistema attraverso l'identificativo 00_{16}.

Quanto descritto finora esaurisce il blocco di dati riguardanti le informazioni di inizio traccia. La coppia successiva, infatti, ha un Δt non nullo, pari a $83\,60_{16}$. La lettura del Δt si estende valutando i bit più significativi negli ottetti via via considerati: $83_{16} = 10000011_2$, $60_{16} = 01100000_2$, e qui ci si ferma, essendo nullo il bit più significativo. Omettendo i bit più significativi, si può ricostruire il valore binario del Δt, pari a $0000011\,1100000_2$ = 480. Richiamando il valore di *PPQN* esplicitato nel *chunk* di intestazione, si evince come il primo evento di performance venga posizionato in corrispondenza della seconda pulsazione. Quello che segue è l'identificativo dell'evento: 91_{16} significa NOTE-ON sul Canale 2, per cui l'evento è lecito attendersi altri due byte relativi all'evento. I byte di dati di NOTE-ON risultano 43_{16} = 67 per il pitch, ossia il *sol* naturale dell'ottava centrale, e 50_{16} = 80 per la velocity.

L'uso di un Δt non nullo per distanziare il primo messaggio di NOTE-ON rispetto all'inizio della traccia permette di richiamare un'osservazione valida in generale per i formati di performance: le pause non vengono esplicitamente codificate, ma emergono naturalmente dall'assenza di suono. Se, nel seguito del tema melodico, fosse comparsa un'ulteriore pausa, il NOTE-ON del nuovo simbolo sarebbe risultato opportunamente distanziato dal NOTE-OFF della nota precedente la pausa. Questo processo è estremamente lineare nel caso di eventi riferibili a un'unica voce; se invece comparissero più voci all'interno della stessa traccia, appartenenti allo stesso canale o a canali diversi, è necessario rammentare che il Δt si riferisce sempre e comunque all'evento precedente. A questo proposito, si consideri lo spartito nella Figura 5.8 e si ipotizzi che i messaggi NOTE-ON e NOTE-OFF di entrambe le voci vengano codificati nella stessa traccia. Considerando la voce superiore, la durata della pausa non può essere semplicemente calcolata dal Δt del NOTE-ON del Fa, in quanto l'evento precedente non è il NOTE-OFF del Do naturale, bensì (presumibilmente) il NOTE-OFF del Mi alla voce inferiore, rispetto al quale il NOTE-ON del Fa potrebbe essere

simultaneo [5].

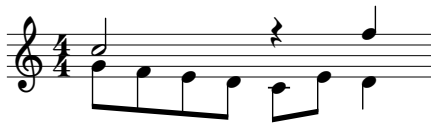


Figura 5.8. Calcolo della durata delle pause in uno spartito con più voci.

Tornando al caso in esame, la nuova coppia ha $\Delta t = 83\,47_{16} = 10000011\,01000111_2$, che si riconduce al valore $11100011_2 = 455$. Ancora una volta, il *running status* è l'unico modo per giustificare i byte successivi, posti a $43\,00_{16}$: poiché lo stato corrente è NOTE-ON sul Canale 2, si tratta dell'evento di spegnimento della nota precedentemente accesa attuato attraverso la sintassi alternativa di NOTE-OFF (Paragrafo 2.4.3).

Il successivo Δt è $19_{16} = 25$, e i byte $45\,50_{16}$ implicano l'accensione di una nuova nota con pitch $45_{16} = 69$ (La naturale dell'ottava centrale) e velocity $50_{16} = 80$. Quindi, nell'esportazione effettuata da *MuseScore*, l'accensione della seconda nota di battuta 1 non è perfettamente simultanea allo spegnimento della prima. La distanza tra due NOTE-ON consecutivi risulta comunque 455 + 25 = 480, che è il valore di MIDI *ticks* per la pulsazione di un quarto.

I quattro byte successivi (byte 80–83) richiamano quanto già visto per lo spegnimento della prima nota (byte 73–77). Anche in questo caso,il Δt è $83\,47_{16} = 455$ e lo stato corrente comporta lo spegnimento della nota accesa, il cui pitch è $45_{16} = 69$.

I byte 84–86 hanno valore $19\,47\,50_{16}$: dopo $19_{16} = 25$ *ticks* ha luogo l'accensione del pitch $47_{16} = 71$ (Si naturale dell'ottava centrale) con velocity $50_{16} = 80$. Lo spegnimento della nota ha luogo con Δt $81\,63_{16} = 10000001\,01100011_2$ che corrisponde a $11100011_2 = 227$, valore pari alla metà di 455. Se ne evince una regola empirica: *MuseScore*, nel distanziare temporalmente i NOTE-OFF rispetto ai NOTE-ON corrispondenti, erode una piccola parte del valore ritmico, pari al 5% circa di quella nominale. I *ticks* mancanti al completamento della figura ritmica vengono reintrodotti come brevi silenzi tra una nota e quella successiva, ossia tra NOTE-OFF e NOTE-ON, attraverso un *delta-time* non nullo associato a quest'ultimo evento. Si osservi che tale aspetto non è parte integrante delle specifiche SMF, ma un effetto introdotto dal software in uso.

A $\Delta t = 0D_{16} = 13$ si ha l'accensione del pitch $48_{16} = 72$ con velocity $50_{16} = 80$, spento con $\Delta t = 81\,63_{16}$ cui corrisponde sempre il valore 227. Si noti che 227 + 13 = 240 = 480 / 2, il che è coerente con il rapporto di durata tra ottavo e quarto.

Lo schema si ripete nuovamente per i byte 98–104, posti a 0D 4A 50 87 0F 4A 00: $\Delta t = 0D_{16}$ = 13 per l'accensione dell'ultima nota, con pitch $4A_{16} = 74$ e velocity $50_{16} = 80$, e spegnimento dopo $87\,0F_{16} = 10000111\,00001111$ che corrisponde a 1110001111_2= 911 *ticks*. Anche la durata in *ticks* della figura ritmica corrispondente a una metà equivale a circa il 95% del valore teorico di $480 \times 2 = 960$ *ticks*.

A una distanza minima di 01_{16}, la traccia viene terminata dal meta-evento di fine traccia: $FF\,2F\,00_{16}$.

Si osservi come, già in questo semplice esempio, gran parte dei messaggi di canale relativi alle voci abbiano beneficiato della tecnica *running status*, con un considerevole risparmio di spazio. Questo risultato è più facile da conseguire nei file MIDI in formato 1, in cui eventi su canali distinti vengono salvati in tracce distinte. Se più strumenti, associati a canali diversi, avessero condiviso la stessa traccia, come richiesto dal formato 0, lo stato corrente – che include l'informazione di canale – sarebbe cambiato di frequente, vanificando in larga misura i vantaggi della tecnica *running status*.

[5] Gli aspetti di incertezza derivano dall'algoritmo di esportazione in MIDI. L'ordinamento di eventi simultanei, per quanto improbabile, potrebbe essere invertito, con uno scambio di posizione tra il NOTE-OFF del Mi e il NOTE-ON del Fa. Inoltre, non tutti i software pongono il NOTE-OFF delle note nell'esatto istante in cui il loro valore ritmico si esaurisce. Come si vedrà nel seguito, *MuseScore*, ad esempio, "accorcia" leggermente la durata delle note.

5.5.3 Esempio 2

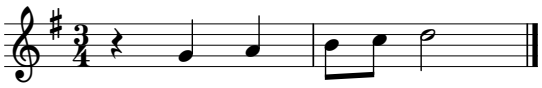


Figura 5.9. Spartito dell'Esempio 2.

```
 0: 4D 54 68 64 00 00 00 06  00 00 00 01 03 C0 4D 54  MThd.........ÀMT
10: 72 6B 00 00 00 44 00 C1  49 00 FF 58 04 03 02 18  rk...D.ÁI.ÿX....
20: 08 00 FF 51 03 07 A1 20  87 40 91 43 60 87 40 81  ..ÿQ..¡ @C`@
30: 43 00 00 91 45 60 87 40  81 45 00 00 91 47 60 83  C..E`@E..G`
40: 60 81 47 00 00 91 48 60  83 60 81 48 00 00 91 4A  `G..H``H..J
50: 60 8F 00 81 4A 00 00 FF  2F 00                    `.J..ÿ/.
```

Figura 5.10. Rappresentazione esadecimale e ASCII del file .mid dell'Esempio 2, esportato da *Cockos REAPER*.

Il secondo esempio è la trascrizione dello stesso tema melodico dell'Esempio 1, effettuata però attraverso *Cockos REAPER 6.11*. [6] Sebbene i due file MIDI derivino da un unico spartito e, quando eseguiti da uno stesso sintetizzatore, suonino in modo pressoché indistinguibile, si notano alcune differenze tra i documenti riportati nella Figura 5.7 e 5.10. La più evidente a una prima analisi sta nella dimensione del file: 109 byte del file MIDI prodotto da *MuseScore*, contro 90 byte di quello esportato da *REAPER*. È lecito attendersi che le differenze si concentrino nel blocco di dati del *chunk* di traccia, in quanto il *chunk* di intestazione di traccia occupa sempre 14 byte e l'*header* del *chunk* di traccia ne richiede 8.

Nonostante gli eventi di performance nei due esempi siano gli stessi, le differenze sono dovute a vari aspetti della codifica. Innanzitutto, a valori più piccoli di *PPQN* (quindi a una risoluzione temporale meno raffinata) corrisponde in generale una minore occupazione di spazio nella codifica dei Δt a lunghezza variabile. In questo specifico esempio si sarebbe potuta adottare una quantizzazione estremamente grezza, arrivando ad assegnare solo 2 *ticks* per pulsazione da un quarto [7]; nell'ambito di questa ipotesi, nessun Δt avrebbe richiesto più di un byte. Rispetto a *MuseScore*, *REAPER* ha invece raddoppiato la risoluzione temporale.

Altri aspetti che incidono sulla dimensione complessiva del file sono il numero e il tipo di meta-eventi ed eventi di inizio traccia inseriti dal software nella codifica. Nel caso in esame, sarà questa la differenza determinante.

Si segnala, infine, l'adozione o meno del *running status* come ulteriore elemento di differenziazione. Con lo stato corrente abilitato, la scelta di posizionare gli eventi di performance in un'unica traccia o su più tracce, la scelta di usare la sintassi alternativa per NOTE-OFF, e addirittura l'ordine in cui si dispongono gli eventi simultanei, sono aspetti che possono incidere considerevolmente sulla dimensione finale del file.

Chunk di intestazione

Per quanto riguarda il primo *chunk*, la cui lunghezza complessiva è sempre di 14 byte e i primi 8 byte sono da considerarsi prefissati, si notano solo due differenze rispetto all'Esempio 1.

La prima di queste sta nel formato del file MIDI, impostato a 0. Si tratta di un'opzione che l'utente può scegliere nell'interfaccia di *REAPER* all'atto dell'esportazione del file. Il diverso

[6] *REAPER* è una *digital audio workstation* con funzioni di *sequencing* prodotta da *Cockos*, disponibile per macOS, per Windows e, in versione beta, per Linux. Il sito web di *REAPER* è https://www.reaper.fm/

[7] Questa scelta avrebbe consentito di posizionare correttamente i messaggi di NOTE-ON e NOTE-OFF per tutti i valori ritmici presenti fino al più piccolo, ossia l'ottavo; non avrebbe permesso, però, di ridurre la durata effettiva della nota rispetto a quella nominale, come nell'esempio precedente.

formato si riflette sul valore dei byte 8-9, che passa da 00 01 dell'Esempio 1 a $00\,00_{16}$, ma non sul valore dei byte 10-11, in quanto – a maggior ragione in un file in formato 0 – sarà sempre uno il *chunk* di traccia.

La seconda differenza riguarda il *PPQN*, posto, in questo caso, a $03\,C0_{16}$ = 960 *ticks* per pulsazione da un quarto, il doppio di quanto riportato nell'Esempio 1.

Chunk di traccia

All'interno dell'intestazione del *chunk* di traccia, l'unica novità rispetto all'esempio precedente sta nella rappresentazione della dimensione in byte del blocco di dati, pari a $00\,00\,00\,44_{16}$ = 68 byte. Anche in questo caso,il valore è coerente con la dimensione totale del file: 90 - 14 - 8 = 68.

Le coppie {Δt, *event*} di inizio traccia esportate da *REAPER* sono:

- $(00, C1\,49)_{16}$ – PROGRAM CHANGE analogo a quello dell'esempio precedente;
- $(00, FF\,58\,04\,03\,02\,18\,08)_{16}$ – Indicazione metrica analoga a quella dell'esempio precedente;
- $(00, FF\,51\,03\,07\,A1\,20)_{16}$ – Tempo metronomico analogo a quello dell'esempio precedente.

Rispetto all'esempio precedente, si nota l'assenza di informazioni sulla tonalità del brano [8] e di eventi di CONTROL CHANGE.

La successiva coppia {Δt, *event*} è $(87\,40,\ 91\,43\,60)_{16}$. I primi due byte rappresentano il Δt: passando dalla codifica a lunghezza variabile al valore ricostruito, si ha $10000111\,01000000_2$= 1111000000_2= 960 *ticks*, che, rapportato al *PPQN*, è esattamente la durata di una pulsazione. Nei byte successivi si ha la codifica dell'evento. Si tratta di midimsgNote-On sul Canale 2 per il pitch 43_{16} = 67 (*sol* naturale dell'ottava centrale) con velocity 60_{16} = 96. Si evince che *REAPER*, di default, esporta le note a una velocity superiore rispetto a *MuseScore*.

La coppia {Δt, *event*} che segue è $(87\,40,\ 81\,43\,00)_{16}$. Rispetto alla strategia adottata da *MuseScore*, che introduceva aspetti di espressività nell'esecuzione della melodia, *REAPER* agisce da *sequencer*, per cui l'evento NOTE-OFF segue esattamente di una pulsazione il corrispettivo NOTE-ON. Riguardo alla codifica dell'evento, si nota l'utilizzo di un vero e proprio messaggio di NOTE-OFF (*nibble* 8_{16}), che vanifica l'effetto dell'eventuale *running status*; l'evento è regolarmente rappresentato su 3 byte.

Il successivo evento di NOTE-ON risulta perfettamente simultaneo al precedente NOTE-OFF, quindi temporizzato con $\Delta t\ 00_{16}$.

Si lascia al lettore l'interpretazione delle coppie successive, che non presentano elementi di novità rispetto a quanto già discusso.

5.5.4 Esempio 3

Il terzo esempio, il cui spartito è mostrato nella Figura 5.11, è la trascrizione di due battute della canzone "Nel blu dipinto di blu", effettuata con *MuseScore 3* e integrata per quanto riguarda i metadati con *Anvil Studio* 2020 [9].

Nonostante si tratti di un piccolo estratto, il file risultante occupa ben 514 byte. Verranno, pertanto, analizzati solo gli aspetti di particolare interesse del file MIDI.

---

[8] Il software usato per produrre il file MIDI, in questo caso, è un *sequencer*, non un *editor*di partitura in cui è prassi specificare la tonalità di impianto.

[9] *Anvil Studio* è un'applicazione gratuita per la composizione musicale rivolta agli utenti che desiderano comporre, registrare e ascoltare file MIDI. Sviluppata da *Willow Software*, è disponibile solo per *Microsoft Windows*. Il sito web è https://www.anvilstudio.com/

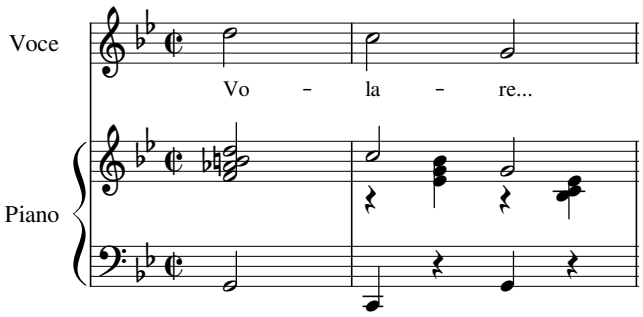


Figura 5.11. Spartito dell'Esempio 3.

```
  0: 4D 54 68 64 00 00 00 06  00 01 00 05 03 C0 4D 54  MThd.........ÀMT
 10: 72 6B 00 00 00 22 00 FF  03 0B 6D 69 64 69 5F 65  rk..."..ÿ..midi_e
 20: 78 70 6F 72 74 00 FF 58  04 01 01 18 08 00 FF 51  xport.ÿX......ÿQ
 30: 03 07 A1 20 00 FF 2F 00  4D 54 72 6B 00 00 00 50  ..¡ .ÿ/.MTrk...P
 40: 00 FF 03 08 4D 65 74 61  64 61 74 61 00 FF 01 16  .ÿ..Metadata.ÿ..
 50: 4E 65 6C 20 62 6C 75 20  64 69 70 69 6E 74 6F 20  Nel blu dipinto
 60: 64 69 20 62 6C 75 00 FF  02 22 46 72 61 6E 63 6F  di blu.ÿ."Franco
 70: 20 4D 69 67 6C 69 61 63  63 69 2C 20 44 6F 6D 65   Migliacci, Dome
 80: 6E 69 63 6F 20 4D 6F 64  75 67 6E 6F 00 FF 2F 00  nico Modugno.ÿ/.
 90: 4D 54 72 6B 00 00 00 58  00 FF 03 04 56 6F 63 65  MTrk...X.ÿ..Voce
 A0: 00 FF 04 04 56 6F 63 65  00 FF 05 02 56 6F 00 C0  .ÿ..Voce.ÿ..Vo.À
 B0: 34 00 B0 07 64 00 B0 0A  40 00 B0 5B 00 00 B0 5D  4.°.d.°.@.°[..°]
 C0: 00 00 90 4A 50 00 B0 79  00 8E 7E 90 4A 00 02 FF  ..JP.°y.~J..ÿ
 D0: 05 02 6C 61 00 90 48 50  8E 7E 90 48 00 02 FF 05  ..la.HP~H..ÿ.
 E0: 02 72 65 00 90 43 50 8E  7E 90 43 00 00 FF 2F 00  .re.CP~C..ÿ/.
 F0: 4D 54 72 6B 00 00 00 B0  00 FF 03 18 50 69 61 6E  MTrk...°.ÿ..Pian
100: 6F 66 6F 72 74 65 20 28  6D 61 6E 6F 20 64 65 73  oforte (mano des
110: 74 72 61 29 00 FF 04 0A  50 69 61 6E 6F 66 6F 72  tra).ÿ..Pianofor
120: 74 65 00 FF 59 02 FE 00  00 C1 00 00 B1 07 64 00  te.ÿY.þ..Á..±.d.
130: B1 0A 40 00 B1 5B 00 00  B1 5D 00 00 91 41 50 00  ±.@.±[..±]..AP.
140: 91 44 50 00 91 47 50 00  91 4A 50 00 B1 79 00 8E  DP.GP.JP.±y.
150: 1E 91 41 00 00 91 44 00  00 91 47 00 00 91 4A 00  .A..D..G..J.
160: 62 91 48 50 87 40 91 3F  50 00 91 43 50 00 91 46  bHP@?P.CP.F
170: 50 86 5E 91 48 00 30 91  3F 00 00 91 43 00 00 91  P^H.0?..C..
180: 46 00 32 91 43 50 87 40  91 3A 50 00 91 3C 50 00  F.2CP@:P.<P.
190: 91 3F 50 86 5E 91 43 00  30 91 3A 00 00 91 3C 00  ?P^C.0:..<.
1A0: 00 91 3F 00 00 FF 2F 00  4D 54 72 6B 00 00 00 52  .?..ÿ/.MTrk...R
1B0: 00 FF 03 1A 50 69 61 6E  6F 66 6F 72 74 65 20 28  .ÿ..Pianoforte (
1C0: 6D 61 6E 6F 20 73 69 6E  69 73 74 72 61 29 00 FF  mano sinistra).ÿ
1D0: 04 0A 50 69 61 6E 6F 66  6F 72 74 65 00 FF 59 02  ..Pianoforte.ÿY.
1E0: FE 00 00 91 2B 50 8E 1E  91 2B 00 62 91 24 50 87  þ..+P.+.b$P
1F0: 0E 91 24 00 87 72 91 2B  50 87 0E 91 2B 00 00 FF  .$.r+P.+..ÿ
200: 2F 00                                             /.
```

Figura 5.12. Rappresentazione esadecimale e ASCII del file .mid dell'Esempio 3, esportato da *Anvil Studio*.

Chunk di intestazione

Le informazioni ivi contenute mostrano che il file MIDI è di tipo 1 (byte 8-9 = $00\,01_{16}$), contiene 5 *chunk* di traccia (byte 10-11 = $00\,05_{16}$) e il *PPQN* corrisponde a 960 *ticks* per pulsazione da un quarto (byte 12-13 = $03\,C0_{16}$).

Primo *chunk* di traccia

Il primo *chunk* di traccia occupa i byte 14-55, come facilmente riscontrabile dal fatto che i byte 14-17 corrispondono a 4D 54 72 $6B_{16}$ ("MTrk") e i byte 53-55 a FF 2F 00_{16} (meta-evento di fine traccia). A riprova di questo, i byte 18-21 contengono il valore 00 00 00 22_{16} = 34, ossia il numero di byte restanti di cui il *chunk* si compone.

Riguardo al contenuto della parte riservata ai dati, in questo primo *chunk* si trovano informazioni sulla *song* non relative alla performance.

Il primo meta-evento, collocato a inizio traccia, è FF 03_{16}, ossia il nome della traccia che viene espresso negli $0B_{16}$ = 11 byte seguenti. Nel caso di meta-eventi testuali, si tratta di convertire i codici ASCII nei corrispettivi caratteri, come mostrato nella parte destra di Figura 5.12. Dunque il nome della prima traccia è "midi_export".

Segue un meta-evento, anch'esso collocato a Δt = 0, che specifica l'indicazione di tempo: FF 58 04 01 01 18 08_{16}. Si tratta di 1/2, contrariamente a quanto mostrato in partitura, ma si deve tener conto che la prima battuta è incompleta: anche in *MuseScore* la prima battuta ha un tempo effettivo di 1/2, sebbene risulti notato come 2/2. Procedendo, 18_{16} = 24 è il numero di *MIDI clocks* per pulsazione, pertanto la pulsazione corrisponde alla figura ritmica del quarto; infine, il numero di trentaduesimi nella pulsazione principale è 8, quindi questa corrisponde effettivamente al valore standard del quarto.

Segue un ulteriore meta-evento collocato a Δt = 0 e relativo al tempo metronomico. FF 51 03 07 A1 20_{16} significa che ogni pulsazione impiega 07 A1 20_{16} = 500.000 µs = 0.5 s. Il metronomo è ancora una volta 120 bpm.

Il meta-evento di fine traccia viene codificato come contemporaneo all'evento precedente.

Secondo *chunk* di traccia

Con uno dei molteplici metodi sopra esposti, si evince che il secondo *chunk* di traccia occupa i byte 56-144.

Si tratta di una sequenza di meta-eventi, tutti caratterizzati da Δt pari a 0. Il primo di questi è il Nome della traccia (FF 03_{16}), che occupa 08_{16} = 8 byte e la cui stringa è "Metadata". Segue un meta-evento di Testo generico (FF 01_{16}), che richiede 16_{16} = 22 byte contenenti la stringa "Nel blu dipinto di blu". Il successivo meta-evento è un Avviso di copyright (FF 02_{16}), espresso su 22_{16} = 34 byte: si tratta della stringa "Franco Migliacci, Domenico Modugno".

Terzo *chunk* di traccia

I restanti *chunk* di traccia contengono informazione di performance. L'analisi prenderà in considerazione solo il primo di essi, in quanto le strutture dati successive non presentano aspetti di rilievo.

A partire dal byte 152, viene specificata la coppia {Δt, *event*} relativa al meta-evento Nome di sequenza/traccia (FF 03_{16}), di lunghezza 04_{16} = 4 byte e di valore "Voce".

Al byte 160 si trova la coppia {Δt, *event*} del meta-evento Nome di strumento (FF 04_{16}), anch'esso di lunghezza 04_{16} = 4 byte e di valore "Voce". Il nome della traccia e dello strumento, invece, risultano differenti tra loro nei successivi *chunk* di traccia.

Sempre a Δt = 0 si incontra un meta-evento Testo cantato (FF 05) di lunghezza 02_{16} = 2 byte e impostato alla prima sillaba del testo cantato: "Vo".

Quello che segue, tralasciando i Δt sempre posti a 0, è un PROGRAM CHANGE (C0 34_{16}, ai byte 175-176), seguito da 4 CONTROL CHANGE (byte 177-192), da NOTE-ON (90 4A 50, byte 194-196), e da un ulteriore CONTROL CHANGE (byte 198-200).

La successiva coppia presenta un Δt pari a 8E $7E_{16}$, ossia $10001110\,01111110_2$, che si trasforma in 11101111110_2 = 1918 MIDI *ticks*, molto prossimo al valore di 1920 *ticks* che corrisponde a $PPQN \times 2$. Questo implica lo spegnimento della nota precedentemente accesa attraverso un messaggio NOTE-ON con velocity 0.

Si prende ora in esame un'ultima coppia $\{\Delta t$, *event*$\}$. Trascorso il tempo di $02_{16} = 2$ *ticks*, che permette di allineare il nuovo evento alla griglia delle pulsazioni da un quarto, si presenta la sillaba successiva (FF 02_{16}) di lunghezza $02_{16} = 2$ byte e di valore "la".

Capitolo 6

MIDI 2.0

La progettazione del protocollo MIDI ha avuto inizio nei primi anni '80, quando le esigenze dei produttori, le caratteristiche dei dispositivi per la musica e il suono e la tipologia stessa di tali apparecchiature erano profondamente diverse da quelle attuali.

L'idea di superare le specifiche MIDI 1.0 nasce nel 2005 e si concretizza al termine di una lunga gestazione, con la standardizzazione di MIDI 2.0 avvenuta in febbraio 2020. È in occasione del NAMM 2020, fiera mondiale dei prodotti musicali, che MMA e AMEI annunciano congiuntamente la nascita di MIDI 2.0 [1]. Il risultato è stato raggiunto grazie agli sforzi di grandi produttori di strumenti e apparecchiature musicali, tra cui Roland, Yamaha, Korg e Roli, ma anche con il coinvolgimento diretto di aziende quali Apple, Microsoft, Google e di altri nomi legati al mondo del software come Ableton e Native Instruments.

MIDI 2.0 rappresenta un'estensione di MIDI 1.0: non intende sostituirlo, ma – al contrario – si basa sugli stessi principi cardine, aspetti architetturali e caratteristiche semantiche. Per tali motivi MIDI 2.0 mantiene piena retrocompatibilità con le specifiche originarie.

Ciononostante, molti aspetti di MIDI 2.0 sono davvero rivoluzionari. Il nuovo protocollo introduce, ad esempio, l'auto-configurazione e lo scambio di informazioni tra dispositivi sia hardware sia software, l'integrazione nativa con DAW e applicazioni web, la risoluzione estesa dei comandi di controllo, una maggiore espressività grazie ai controlli nota per nota, 256 canali per connessione anziché 16 e un timing più accurato. Altro aspetto di grande rilievo è la gestione della comunicazione bidirezionale tra dispositivi [23].

L'ecosistema MIDI 2.0 è complesso e articolato, come evidenzia la Figura 6.1 tratta dal sito midi.org [2].

6.1 Principali componenti di MIDI 2.0

Le specifiche MIDI 2.0 estendono il MIDI 1.0 attraverso cinque documenti, uno di panoramica, che ha lo scopo di introdurre la nuova versione, e quattro di specifiche vere e proprie. Nel dettaglio, si tratta di:

1. MIDI Capability Inquiry (MIDI-CI);
2. Common Rules for MIDI-CI Profiles;
3. Common Rules for MIDI-CI Property Exchange;
4. Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol.

[1] AMEI (*Association for Musical Electronics Industry*) e MMA (*MIDI Manufacturers Association*) sono, rispettivamente, le autorità per il MIDI per il Giappone e in ambito internazionale.

[2] https://www.midi.org/midi-articles/details-about-midi-2-0-midi-ci-profiles-and-property-exchange

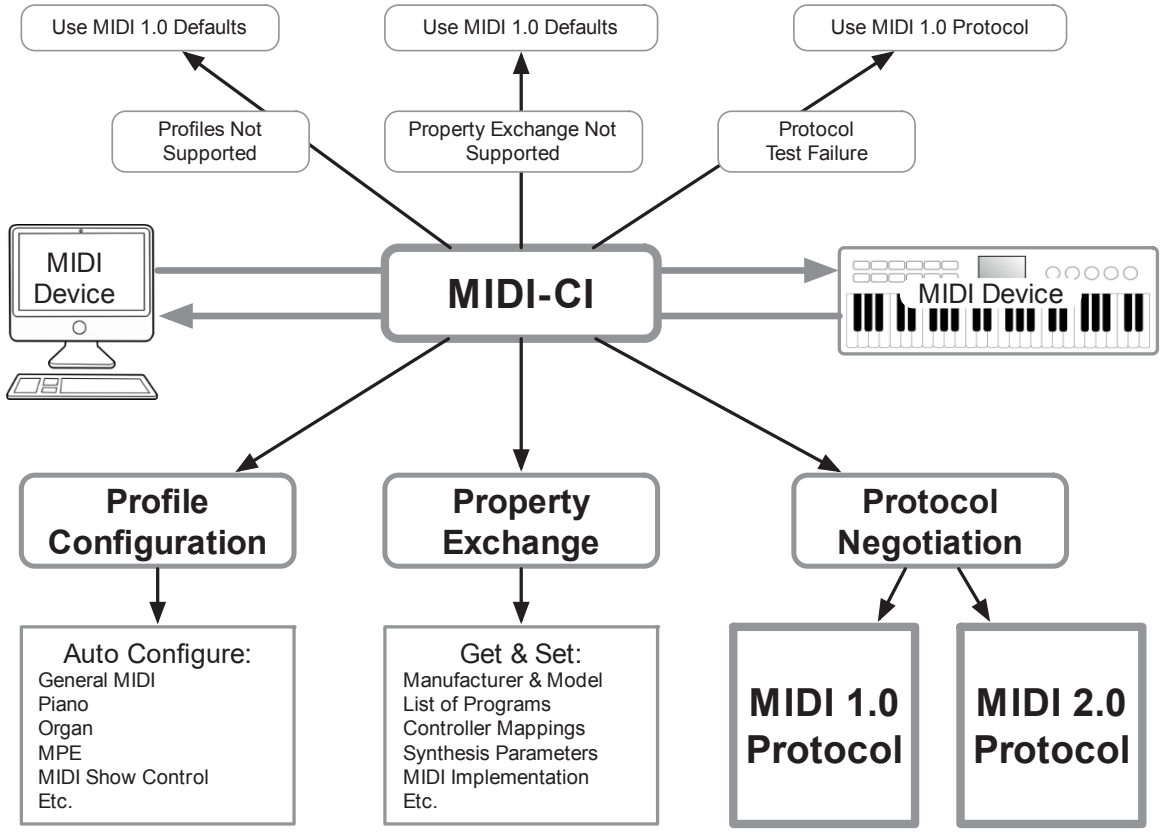


Figura 6.1. L'ambiente MIDI 2.0 (figura tratta dal sito midi.org).

Insieme, questo gruppo di documenti definisce l'architettura di MIDI 2.0 e le sue connessioni con MIDI 1.0. Si sottolinea che nessuno dei documenti citati è autocontenuto, in quanto si tratta di estensioni rispetto alle specifiche originarie. Da qui l'importanza, anche per un manuale scritto dopo l'avvento di MIDI 2.0, di condurre un'analisi dettagliata delle specifiche MIDI 1.0.

A ciascuno degli argomenti citati viene dedicato uno dei prossimi paragrafi nel seguito del capitolo.

6.2 MIDI-CI

MIDI Capability Inquiry (MIDI-CI) costituisce la parte di specifiche MIDI 2.0 volta ad abilitare l'estensione del protocollo MIDI originario. MIDI-CI definisce un'architettura che consente ai dispositivi in grado di comunicare in modo bidirezionale di accordarsi nell'uso delle funzionalità estese di MIDI 2.0, pur preservando la compatibilità con i restanti dispositivi nel *setup*. A tale scopo, MIDI-CI prevede un meccanismo di *fallback*: se un dispositivo non supporta le nuove specifiche, il protocollo MIDI opera secondo le specifiche 1.0.

Gli obiettivi perseguiti nella progettazione di MIDI-CI sono:

- estendere le potenzialità di MIDI preservando la retrocompatibilità;
- consentire a un mittente di conoscere le capacità di un destinatario;
- permettere una configurazione più agevole di un gruppo di dispositivi attraverso i profili di auto-configurazione;
- definire metodi per recuperare o impostare un ampio spettro di proprietà dei dispositivi;
- definire metodi per negoziare la scelta del protocollo da usare nella comunicazione tra dispositivi.

Tali obiettivi vengono perseguiti attraverso 3 aree di funzionalità, chiaramente mostrate nella Figura 6.1:

1. negoziazione del protocollo (*Protocol Negotiation*). Queste funzionalità riguardano la scelta del protocollo tra dispositivi;

2. configurazione dei profili (*Profile Configuration*). Queste funzionalità si rivolgono all'utilizzo dei profili di auto-configurazione;

3. scambio di proprietà (*Property Exchange*). Queste funzionalità si focalizzano sul recupero e sull'impostazione dei valori delle proprietà dei dispositivi attraverso il formato *JavaScript Object Notation* (JSON).

Come si avrà modo di approfondire nel prossimo paragrafo, i dispositivi MIDI, anche quelli di ultima generazione, operano inizialmente seguendo il protocollo MIDI 1.0 e, solo dopo aver preso reciproci accordi attraverso MIDI-CI, utilizzano le funzionalità estese proprie di MIDI 2.0.

6.2.1 Retrocompatibilità

La compatibilità con dispositivi antecedenti al rilascio delle specifiche 2.0, che costituiscono la stragrande maggioranza dell'hardware e del software attualmente in uso, è ovviamente un aspetto di prioritaria importanza. Gli utenti si aspettano che i nuovi dispositivi MIDI funzionino perfettamente con quelli commercializzati negli ultimi 33 anni.

Considerando la struttura dei messaggi MIDI 1.0 e la loro semantica, aspetti ampiamente descritti nel Capitolo 2, si evince che tutti i byte di stato MIDI risultino definiti, come pure i loro codici operativi e i *payload*, ossia i dati ad essi associati. In questo contesto sarebbe difficile definire nuovi messaggi o modificare il formato dei messaggi MIDI esistenti, se non utilizzando messaggi SysEx. D'altro canto, l'intento di estendere le potenzialità del MIDI attraverso funzioni aggiuntive richiede un nuovo protocollo basato su messaggi MIDI estesi.

Per salvaguardare la retrocompatibilità, i dispositivi devono essere in grado di condividere – attraverso lo scambio di informazioni – le proprie potenzialità con quelle delle apparecchiature cui sono collegati, e conoscere a loro volta le potenzialità di queste ultime. A tale scopo, quando due dispositivi vengono collegati tra loro, essi usano MIDI 1.0 e, solo dopo aver verificato la capacità da parte di entrambi di supportare MIDI 2.0, entrano in tale modalità operativa. Nello scenario in cui i dispositivi condividano il supporto alle stesse funzionalità MIDI estese, essi possono decidere di utilizzarle. Scopo fondamentale di MIDI-CI è mettere a disposizione un meccanismo di scambio informativo, abilitando eventualmente l'uso delle funzionalità estese.

La soluzione implementata da MIDI-CI per garantire retrocompatibilità richiede una comunicazione bidirezionale. Una volta stabilita una connessione MIDI-CI tra i dispositivi, i messaggi di interrogazione e risposta delineano le possibilità di ciascun dispositivo. A questo punto MIDI-CI negozia o si configura automaticamente per utilizzare le funzionalità comuni tra i dispositivi. MIDI-CI fornisce inoltre meccanismi di test da effettuare quando si abilitano nuove funzionalità: se un test fallisce, i dispositivi tornano a utilizzare MIDI 1.0 per quella specifica funzione. Quindi l'adozione delle nuove funzionalità può essere vista come un processo incrementale: non è necessario che due dispositivi collegati tra loro implementino entrambi tutte le estensioni previste da MIDI 2.0. Per ogni evenienza, MIDI-CI fornisce un meccanismo di *fallback* a MIDI 1.0, il che consente alle macchine di funzionare correttamente, sebbene, in tal caso, non vadano a sfruttare le proprie caratteristiche più avanzate.

6.2.2 Livelli di MIDI-CI

MIDI-CI presenta una strutturazione a livelli che richiama il modello *Open Systems Interconnection* (OSI), conosciuto anche come modello ISO/OSI, in quanto standard promulgato nel 1984

dall'International Organization for Standardization [3]. Adottato principalmente nel campo delle telecomunicazioni e dell'informatica, il modello OSI definisce una struttura a livelli o strati (*layer*) composta da una pila di protocolli di comunicazione suddivisa in 7 livelli, i quali, nel complesso, coprono tutte le funzionalità della rete.

Il modello proposto è logico-gerarchico. Il grado di astrazione più basso è costituito dallo strato fisico, che riguarda il mezzo trasmissivo e la propagazione del segnale, seguito dal livello di collegamento dati e da quello di rete; si tratta dei cosiddetti *livelli dei mezzi*, perché trattano la propagazione del segnale sul mezzo trasmissivo. I quattro livelli superiori, detti *livelli degli host*, sono, rispettivamente, trasporto, sessione, presentazione e applicazione.

MIDI-CI si focalizza sugli *host*, e in analogia con il modello OSI presenta i seguenti livelli, dal più basso (trasporto) al più alto (applicazione):

- trasporto (*transport*). Riguarda il mezzo per la connessione hardware o software. Esempi sono costituiti dai cavi DIN a 5 pin, USB, Ethernet, ecc. La funzione MIDI delegata a questo livello è detta discovery, e stabilisce l'accoppiamento di un input e un output per la comunicazione bidirezionale tra dispositivi, indipendentemente dal mezzo in uso;

- larghezza di banda (*bandwidth*). Considera aspetti relativi alla velocità di produzione e trasferimento dei dati. Esempi di questo sono il valore 31250 bps per i cavi DIN a 5 pin e la velocità di 1 Mbps su connessioni USB. Non esiste una specifica funzione MIDI-CI legata a questo livello, in quanto è lo strato di trasporto a occuparsene;

- formato dei pacchetti (*packet format*). Si tratta del contenitore per il *payload* del protocollo, descritto al livello superiore. Esempi in tal senso sono forniti dal flusso di dati MIDI 1.0, dai messaggi a 32 bit per USB-MIDI, o dal nuovo Universal MIDI Packet introdotto in MIDI 2.0 e descritto nel seguito. Anche in questo caso non esiste una funzione MIDI-CI specifica, in quanto il formato dei pacchetti viene delegato allo strato di trasporto;

- protocollo (*protocol*). Riguarda il linguaggio per la trasmissione dei dati, ad esempio come esprimere NOTE-ON, CONTROL CHANGE o PROGRAM CHANGE nel tradizionale formato MIDI 1.0, che contempla 1 bit di flag per lo stato e 7 bit per i dati, o nel nuovo formato MIDI 2.0. La funzione MIDI-CI riferibile a questo livello riguarda la cosiddetta negoziazione del protocollo (*protocol negotiation*), ossia la selezione del protocollo MIDI 1.0, del protocollo MIDI 2.0 con o senza marcatura temporale e l'abilitazione delle funzionalità estese;

- profili (*Profiles*). Si focalizza sui profili MIDI-CI, intesi come collezioni di parametri del dispositivo e messaggi MIDI comuni ai vari produttori. Esempi tratti dal passato includono tecnologie citate nel Capitolo 4, quali General MIDI e MIDI Polyphonic Expression; altre possibilità, quali l'*Hi-res Piano Profile*, sono invece contemplate dalle specifiche MIDI 2.0. La funzione MIDI-CI, in questo caso, è la configurazione dei profili (*Profile Configuration*), che può avvenire in modo automatico tra dispositivi che condividono profili comuni;

- implementazione MIDI (*MIDI Implementation*). Concerne i canali e i messaggi supportati, di cui un esempio è la mappatura dei *controller*. MIDI-CI consente di scoprire i dettagli implementativi di un dispositivo MIDI grazie alla funzione di scambio delle proprietà (*Property Exchange*), comune anche allo strato successivo;

- dispositivo (*Device*). Riguarda i dettagli e le specifiche dei modelli di dispositivo. Qualsiasi prodotto MIDI in commercio è un valido esempio per questo livello. Ancora una volta viene chiamata in causa la funzione MIDI-CI di scambio delle proprietà (*Property Exchange*), che permette di leggere e impostare un ampio spettro di proprietà o di stati nei dispositivi.

[3] https://www.iso.org/standard/20269.html

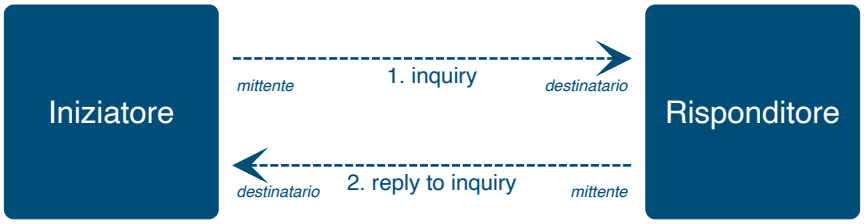


Figura 6.2. Iniziatore e risponditore per una transazione MIDI-CI.

6.2.3 Aspetti topologici

MIDI-CI richiede comunicazioni bidirezionali tra dispositivi. A tale scopo ogni dispositivo deve presentare una porta di input accoppiata con la porta di output di un altro dispositivo, entrambi compatibili con MIDI-CI. Come detto, se solo uno dei dispositivi presenta tali caratteristiche, la comunicazione è unidirezionale e avviene secondo la logica MIDI 1.0.

Ogni coppia di porte (una porta di input e una porta di output) viene utilizzata per le cosiddette transazioni MIDI-CI (*MIDI-CI Transactions*), ossia un insieme di messaggi MIDI-CI che include una richiesta (*inquiry*) inviata da un dispositivo iniziatore (*Initiator device*) e una risposta alla richiesta inviata da un dispositivo risponditore (*Responder device*). Il concetto di *inquiry*, qui tradotto per semplicità come "richiesta", in lingua inglese è più forte di *request*: si tratta, per meglio dire, di un'indagine, una ricerca, un'investigazione. Il riscontro da parte del risponditore potrebbe essere un singolo messaggio che soddisfa la richiesta, un insieme di messaggi che soddisfano la richiesta o un messaggio di errore.

Va sottolineato che, in una connessione bidirezionale MIDI-CI, entrambi i dispositivi coinvolti sono tanto mittenti quanto destinatari dei messaggi che costituiscono una transazione. In altri termini, i ruoli di iniziatore e risponditore riguardano la comunicazione bidirezionale di una transazione, al cui interno entrambi i dispositivi inviano e ricevono messaggi, e ciascuno dei due può assumere il ruolo di iniziatore. La Figura 6.2 serve a fissare graficamente questi concetti.

Nell'ottica di MIDI-CI, le *inquiry* sono finalizzate ad apprendere qualcosa sul destinatario. Il protocollo, infatti, assume che nel sistema sia l'iniziatore a voler apprendere qualcosa sul risponditore, adattando il proprio output alle funzionalità supportare da quest'ultimo, e non viceversa.

Compito dell'iniziatore è impostare e negoziare i parametri della transazione per garantire l'interoperabilità tra i dispositivi. L'obiettivo normalmente è configurare entrambi i dispositivi per una successiva comunicazione, che vede l'iniziatore come mittente e il risponditore come destinatario dei messaggi MIDI. Le impostazioni possono propagarsi in entrambe le direzioni, ossia riguardare allo stesso modo iniziatore e risponditore, o in un'unica direzione, configurando l'iniziatore sulla base delle caratteristiche del risponditore. A questo riguardo esistono due differenti procedure di negoziazione bidirezionale. Alcune transazioni MIDI-CI creano impostazioni che sono comuni in entrambe le direzioni. In questi casi, quando l'iniziatore determina e controlla la negoziazione di determinati parametri, lo fa in modo identico nelle due direzioni, provocando un cambiamento simultaneo nel protocollo in uso in entrambe le direzioni (negoziazione del protocollo). In virtù di tale simmetria, non è rilevante quale dei due dispositivi funga da iniziatore. Altre transazioni MIDI-CI, invece, creano impostazioni per l'interoperabilità in un'unica direzione. In questo scenario, quando l'iniziatore determina e controlla la negoziazione di determinati parametri, lo fa nell'ottica dell'interoperabilità nei messaggi inviati dall'iniziatore stesso al risponditore. Si tratta della situazione standard per le transazioni volte alla configurazione dei profili e allo scambio di proprietà, di cui si parlerà, rispettivamente, nei Paragrafi 6.3 e 6.4.

Il caso in cui ciascuno dei dispositivi voglia impostare alcuni parametri dell'altro, ma non gli stessi parametri e/o valori, necessariamente non rientra nello scenario della negoziazione (bidirezionale) di impostazioni bidirezionali sopra menzionato. Si tratta piuttosto di due

negoziazioni (bidirezionali) di impostazioni in una sola direzione, in cui i ruoli di iniziatore e di risponditore, a un certo punto, si scambiano per consentire al secondo dispositivo di impostare il primo.

Va ricordato che l'adozione di MIDI-CI pone alcune limitazioni sulla topologia del *setup* MIDI rispetto a quanto supportato da MIDI 1.0. In particolare, è caldamente sconsigliato l'uso dei connettori MIDI THRU e di dispositivi quali i *MIDI merger*, il cui comportamento non è definito nelle specifiche e potrebbe portare a situazioni di errore. Stanno comunque nascendo implementazioni "intelligenti" sia per le porte MIDI sia per i dispositivi di *merging*, che trattano correttamente la comunicazione bidirezionale.

6.2.4 Indirizzamento dei messaggi e MUID

I messaggi MIDI-CI vengono scambiati tra dispositivi utilizzando indirizzi o instradamenti determinati da una combinazione di:

1. una connessione MIDI bidirezionale;
2. campi di identificazione dei dispositivi all'interno di messaggi SysEx;
3. MUID dei dispositivi.

Un aspetto innovativo della versione 2 del protocollo è dato proprio dal *MIDI Universal ID* (MUID), un numero pseudocasuale a 28 bit generato da un dispositivo MIDI-CI e usato per identificare univocamente i messaggi inviati da o destinati a tale dispositivo. Riguardo ai valori che il MUID può assumere, al fine di poter essere rappresentati su 4 byte, essi devono ricadere nell'intervallo $[00000000, 0FFFFFFF]_{16}$. Va però segnalato che il sotto-intervallo $[0FFFFF00, 0FFFFFFE]_{16}$ è definito come riservato, per cui i valori che vi rientrano al momento non sono assegnabili a MUID. Inoltre il valore $0FFFFFFF_{16}$ viene usato come MUID di *broadcast*, e il suo ruolo verrà presto chiarito.

Il MUID rimane assegnato finché il dispositivo non viene spento o riavviato. L'unica eccezione è data dalla ricezione di un messaggio di invalidazione del MUID, che forza una nuova assegnazione. Quando il dispositivo viene riacceso, la raccomandazione è quella di assegnare un nuovo MUID. Se gli algoritmi di generazione pseudo-casuale contenuti nei vari dispositivi sono efficaci, è altamente improbabile la coesistenza all'interno di un unico *setup* di più identificativi in collisione; il protocollo comunque prevede questa possibilità e contempla meccanismi di risoluzione dei conflitti.

In generale, tutti i messaggi MIDI-CI devono includere il MUID del dispositivo mittente e del dispositivo destinatario. Ma non sempre il MUID di destinazione indicato all'interno del messaggio può essere univoco: alcuni messaggi sono intrinsecamente concepiti per più destinatari nel *setup*, mentre in altri casi il destinatario non è noto a priori. Un esempio è dato dal messaggio di Discovery, trattato nel seguito. Un altro esempio è costituito dal messaggio di invalidazione del MUID, il cui destinatario presenta un problema di identificazione, quindi non può essere raggiunto attraverso il proprio MUID. In questi casi viene utilizzato il MUID di *broadcast* sopra riportato in luogo di un MUID specifico.

6.2.5 Come si stabilisce la connessione MIDI-CI

Per comunicare tra loro, due dispositivi MIDI-CI devono effettuare una reciproca connessione (*pairing*).

Il primo passaggio è costituito dalla transazione *Discovery* (letteralmente: scoperta). Essa richiede lo scambio di più messaggi. In particolare, uno dei dispositivi si pone alla ricerca fungendo da iniziatore e inviando il messaggio di Discovery. Al suo interno, il MUID dell'iniziatore è indicato come origine, mentre il MUID del destinatario non può essere noto a priori, visto che lo scopo del messaggio è proprio quello di individuarlo nel *setup*; per questo motivo viene utilizzato un MUID

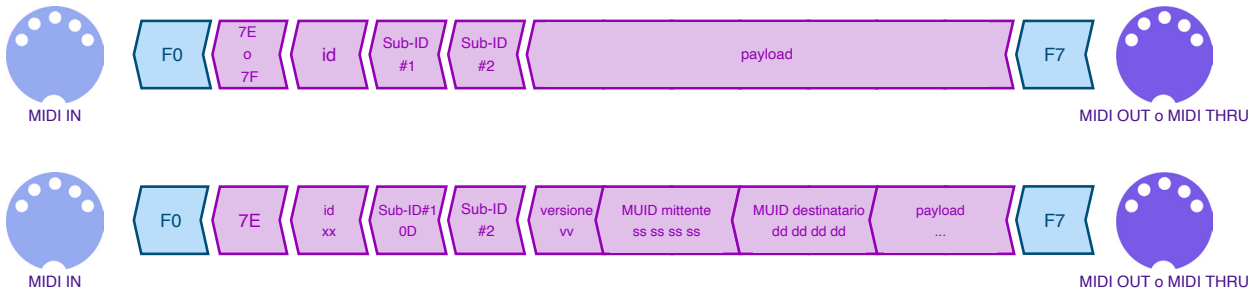


Figura 6.3. Rappresentazione grafica di un generico messaggio *SysEx* universale (sopra) e del formato standard di un messaggio MIDI-CI (sotto).

di *broadcast*. Qualsiasi dispositivo MIDI-CI riceva il messaggio di DISCOVERY deve agire come risponditore, inviando un messaggio REPLY TO DISCOVERY con il proprio MUID come sorgente e il MUID dell'iniziatore come destinatario.

Una volta conclusa la transazione di Discovery tra due dispositivi, ciascuno di essi può assumere il ruolo di iniziatore per una successiva transazione. Questo avviene inviando una MIDI-CI INQUIRY al MUID di destinazione del risponditore individuato. Il risponditore, o i risponditori, agiscono sulla base del contenuto della *inquiry* e rispondono mandando un messaggio di *reply* all'iniziatore.

6.2.6 Formato dei messaggi MIDI-CI e categorie

Ogni transazione MIDI-CI si basa su messaggi *SysEx* universali non in tempo reale, la cui tabella è stata aggiornata con il rilascio delle specifiche MIDI 2.0. Nel formato dati MIDI 1.0, essi richiedono l'invio di un messaggio SYSEX e di un messaggio EOX in apertura e a chiusura del *payload*, e questo aspetto viene mantenuto per retrocompatibilità anche in MIDI 2.0. Tuttavia, si segnala che in alcuni protocolli AMEI/MMA (tra cui lo Universal MIDI Packet descritto più avanti, nel Paragrafo 6.5) i byte di stato che aprono e chiudono una comunicazione *SysEx* vengono omessi. L'argomento dei messaggi *SysEx* universali è stato trattato nel dettaglio nel Paragrafo 2.10.2.

Il formato comune a tutti i messaggi MIDI-CI, espresso in esadecimale e confrontato con quello dei *SysEx* universali nella Figura 6.3, è:

```
F0 7E xx 0D yy vv ss ss ss ss dd dd dd dd [...]  F7.
```

Il byte `xx`, che corrisponde al campo ID nella struttura generale, contiene l'identificativo del dispositivo che funge da origine o destinazione, a seconda del tipo di messaggio; $7F_{16}$ indica l'intera porta MIDI di origine/destinazione (tutti i 16 canali), i valori nell'intervallo $[00, 0F]_{16}$ identificano i canali MIDI da 1 a 16, e l'intervallo [10, 7E] è riservato.

Il successivo byte, che nella struttura dei messaggi *SysEx* universali corrisponde al campo *Sub-ID #1*, è posto a $0D_{16}$per tutti i messaggi.

Il successivo byte, denotato da `yy` e corrispondente al campo *Sub-ID #2*, determina la categoria del messaggio e la sua funzione. L'argomento verrà trattato in maggiore dettaglio nel seguito.

Il successivo campo, rappresentato con punti di sospensione nella Figura 6.3, riguarda i valori da veicolare per lo specifico *SysEx*; per tale motivo nel Paragrafo 2.10.2 non è stato possibile dettagliare la lunghezza di questa parte del messaggio e il significato dei singoli byte. Nel caso di messaggi MIDI-CI, invece, è possibile dettagliare ulteriormente il contenuto di questa parte. Innanzitutto, il byte `vv` permette di specificare la versione/formato del messaggio MIDI-CI. A seguire, si riconoscono due raggruppamenti di 4 byte, indicati come `ss ss ss ss` e `dd dd dd dd`, atti a contenere, rispettivamente, il MUID di origine e destinazione, espressi su 28 bit preponendo a ogni gruppo di 7 bit uno 0 e rappresentando il valore con strategia *LSB first*. Infine, la parte sottesa dai punti di sospensione riguarda l'invio dei dati veri e propri, la cui dimensione e significato dipendono dallo specifico messaggio MIDI-CI.

Attraverso i valori consentiti per il campo *Sub-ID #2*, vengono individuate quattro categorie di messaggi che operano di concerto per portare a termine una determinata funzione MIDI-CI. La categoria viene espressa attraverso il valore del *nibble* di ordine alto nel campo Sub-ID #2 del messaggio SysEx. L'elenco sottostante contiene quest'ultimo valore, l'intervallo di valori ammissibili per *Sub-ID #2* e la denominazione/descrizione della categoria:

0. $[00, 0F]_{16}$ – intervallo riservato (messaggi non ancora definiti);
1. $[10, 1F]_{16}$ – messaggi *Protocol Negotiation*, ossia di negoziazione del protocollo;
2. $[20, 2F]_{16}$ – messaggi *Profile Configuration*, ossia di configurazione dei profili;
3. $[30, 3F]_{16}$ – messaggi *Property Exchange*, ossia di interscambio di proprietà;
4. $[40, 4F]_{16}$ – intervallo riservato (messaggi non ancora definiti);
5. $[50, 5F]_{16}$ – intervallo riservato (messaggi non ancora definiti);
6. $[60, 6F]_{16}$ – intervallo riservato (messaggi non ancora definiti);
7. $[70, 7F]_{16}$ – messaggi *Management*, ossia di gestione.

Oltre ad aver definito solo 4 categorie rispetto alle 8 teoricamente possibili, MIDI-CI sfrutta solo sottoinsiemi di valori anche per le categorie definite, lasciandone altri riservati per usi futuri.

Riguardo alle transazioni, esiste un ordine predefinito nell'elaborazione. Va detto che i dispositivi compatibili MIDI-CI non devono necessariamente supportare tutte le categorie sopra menzionate; il *Property Exchange*, ad esempio, potrebbe non essere implementato. Alla prima connessione MIDI-CI i dispositivi devono procedere per ciascuna categoria supportata nel seguente ordine:

1. esplorazione (*Discovery*);
2. negoziazione del protocollo (*Protocol Negotiation*);
3. configurazione dei profili (*Profile Configuration*);
4. interscambio di proprietà (*Property Exchange*).

Se l'iniziatore o il risponditore non supportano una data categoria, devono procedere con quella successiva. Una volta eseguito questo insieme iniziale di transazioni, ciascun dispositivo può liberamente utilizzare qualsiasi messaggio di *Inquiry* e *Negotiation*.

Una trattazione puntuale di ogni messaggio MIDI-CI esulerebbe dagli scopi del presente testo; al riguardo si rimanda il lettore ai documenti di specifica per approfondimenti. Nel seguito ci si limiterà pertanto a richiamare i concetti fondamentali e ricorrenti e a presentare alcuni esempi significativi di messaggi MIDI-CI.

6.2.7 Esempi

In questo paragrafo si riportano, a titolo esemplificativo, alcuni messaggi MIDI-CI di particolare rilievo. La loro struttura generale ricalca quella illustrata nel paragrafo precedente, trattandosi di messaggi *SysEx* universali inquadrati nelle specifiche MIDI-CI.

Esempio: messaggio DISCOVERY e risposta

Il messaggio DISCOVERY viene inviato quando un iniziatore intende stabilire connessioni verso altri dispositivi MIDI-CI. Tra gli eventi che richiedono la generazione del messaggio, vanno ricordati l'accensione e il completamento della procedura di boot del dispositivo, l'avvio esplicito della modalità MIDI-CI da parte dell'utente, il completamento di una procedura di auto-configurazione, l'invalidazione del proprio MUID e la conseguente adozione di un nuovo MUID con cui ristabilire le connessioni MIDI-CI.

Il messaggio di DISCOVERY, espresso in esadecimale, è il seguente:

`F0 7E 7F 0D 70 vv ss ss ss ss 7F 7F 7F 7F mm mm mm ff ff nn nn rr rr rr rr cc xx xx xx xx F7.`

Rispetto alla forma generale presentata nel Paragrafo 6.2.6, il terzo byte viene fissato a $7F_{16}$, ossia all'intera porta MIDI. Il quinto byte, riservato al *Sub-ID #2*, viene posto a 70_{16}, ossia il messaggio DISCOVERY all'interno della categoria *Management Messages*. Il byte seguente, denotato come `vv`, identifica la versione e/o il formato di messaggio MIDI-CI. I 4 byte seguenti hanno la finalità di identificare il mittente attraverso il proprio MUID, quindi non trovano una valorizzazione di default nella forma generale. I 4 byte del destinatario, posti a $7F\,7F\,7F\,7F_{16}$, contengono il MUID di *broadcast*. Seguono 3 byte, indicati come `mm mm mm`, che servono a identificare il produttore del dispositivo, secondo la logica introdotta nei messaggi *SysEx* del produttore. I successivi 2 byte, `ff ff`, contengono l'identificazione della famiglia del dispositivo, e 2 ulteriori byte, `nn nn`, il numero di modello. Per tutti questi valori, l'LSB viene inviato per primo. I 4 byte `rr rr rr rr` esprimono il livello di revisione del software. Il successivo byte `cc` riguarda le categorie di *Capability Inquiry* supportate, e viene gestito come una mappa di bit. Infine, i 4 byte di *payload* `xx xx xx xx` codificano la dimensione massima per i messaggi *SysEx* (*Receivable Maximum SysEx Message Size*) ricevibili da parte del dispositivo. Questo valore verrà ripreso nel Paragrafo 6.4.4, quando si parlerà delle transazioni per l'interscambio di proprietà tra dispositivi.

A questo messaggio deve far seguito, da parte dei dispositivi compatibili con MIDI-CI presenti nel sistema, un messaggio REPLY TO DISCOVERY, il cui scopo è trasmettere all'iniziatore il proprio MUID. Il formato è molto simile a quello della richiesta. La prima differenza si trova, logicamente, nel quinto byte: il *Sub-ID #2* viene posto, in questo caso, a 71_{16} per identificare il messaggio REPLY TO DISCOVERY. L'altro aspetto distintivo riguarda i 4 byte dedicati al MUID di mittente e destinatario: i primi contengono l'indirizzo del risponditore, e i secondi l'indirizzo dell'iniziatore della transazione anziché l'indicazione di *broadcast*.

Esempio: messaggio INVALIDATE MUID

Il messaggio INVALIDATE MUID trova due applicazioni: viene inviato dal dispositivo identificato dal MUID stesso nei casi in cui si stia spegnendo o – per qualche motivo – non debba più utilizzare il proprio MUID, oppure nel caso di collisione tra i MUID di due o più dispositivi. Tutte le componenti MIDI-CI nel *setup* devono essere in grado di processare i messaggi di questo tipo. Se un dispositivo riconosce come proprio MUID quello specificato nel messaggio, deve terminare qualsiasi transazione attiva e generare un nuovo MUID; se un dispositivo riconosce il MUID da invalidare tra quelli dei dispositivi ad esso noti, deve terminare qualsiasi transazione attiva che li coinvolga ed eliminare dalla propria cache ogni informazione relativa.

Il messaggio, nella fattispecie, è il seguente:

`F0 7E 7F 0D 7E vv ss ss ss ss 7F 7F 7F 7F xx xx xx xx F7.`

Rispetto alla forma generale presentata nel Paragrafo 6.2.6, il terzo byte viene fissato a $7F_{16}$, ossia all'intera porta MIDI; il quinto byte, riservato al *Sub-ID #2*, viene posto a $7E_{16}$, ossia il messaggio INVALIDATE MUID all'interno della categoria *Management Messages*; segue il byte riservato alla versione; i 4 byte del mittente dipendono dal mittente, per cui vengono lasciati non definiti nella forma generale, mentre i 4 byte del destinatario, posti a $7F\,7F\,7F\,7F_{16}$, contengono il MUID di

broadcast. Infine, i 4 byte di *payload* contengono il *Target MUID*, ossia l'identificativo da invalidare, espresso – come di consueto – inviando l'LSB per primo.

6.3 Profili MIDI-CI

Un profilo (*Profile*) è un insieme di messaggi e regole di implementazione volto a raggiungere un particolare scopo o a conformarsi a una specifica applicazione. Oltre a definire le risposte ai messaggi MIDI, un profilo può dettare i requisiti funzionali di un dispositivo, quali l'implementazione di una componente di ricezione (*Receiver*) o di invio (*Sender*). I profili costituiscono una componente importante per abilitare la funzione di auto-configurazione intelligente propria di MIDI 2.0. Le regole comuni per i profili MIDI-CI vengono descritte nel documento di specifica chiamato "Common Rules for MIDI-CI Profiles".

L'approccio che più si è avvicinato ai profili MIDI-CI nella storia del MIDI è stato il General MIDI (Paragrafo 4.1). L'adesione alle sue specifiche, infatti, permette di fare affidamento su un set di suoni predefinito, sulla corrispondenza tra *patch* e identificativi noti a priori, sulla ricezione dei 16 canali MIDI, sull'assegnazione dei timbri percussivi al Canale 10, sul supporto di determinati CONTROL CHANGE e su una risposta definita ad essi, e via dicendo. GM non consente, tuttavia, la comunicazione bidirezionale; infatti l'invio di un messaggio di accensione della modalità GM, che può trovare un riscontro positivo o negativo nei destinatari, non consente al mittente di indagare la sua abilitazione sui singoli dispositivi. Al contrario, i profili MIDI-CI traggono pieno beneficio dalla comunicazione bidirezionale. Si tratta di una forma di conoscenza condivisa tra dispositivi in grado di scambiarsi informazioni sulle proprie funzionalità. L'obiettivo è raggiungere un livello di controllo più integrato e una maggiore prevedibilità dei risultati che deriveranno dall'invio di messaggi MIDI.

6.3.1 Recupero e configurazione dei profili

Determinate *inquiry* consentono a un iniziatore di richiedere e di recuperare un elenco di profili supportati da un risponditore. L'iniziatore può utilizzare queste informazioni per configurare automaticamente la connessione al fine di incrementare l'interoperabilità di applicazioni specifiche. L'iniziatore gestisce la configurazione abilitando e disabilitando particolari profili sul risponditore. Quando un risponditore cambia i suoi profili in risposta a un evento che non è gestito dall'iniziatore, il risponditore deve informare l'iniziatore.

Uno scenario applicativo è dato da un software di editing del suono che riconfigura i controlli mostrati all'utente sulla base del dispositivo da controllare, modificando l'interfaccia a seconda dell'adesione a un profilo di pianoforte elettrico o a un profilo di organo a tiranti [4].

6.3.2 Identificativo del profilo

Il *profiler ID* è un'informazione espressa su 5 byte che ha lo scopo di identificare univocamente il profilo. Essa deve essere inclusa all'interno dei *Profile Configuration Messages*, categoria menzionata nel Paragrafo 6.2.6. Esistono profili standardizzati, il cui elenco è gestito da MMA e AMEI, e profili proprietari dei produttori. Nella Tabella 6.1 vengono descritti nel dettaglio i 5 byte di cui si compone l'identificativo.

Per i profili standard, il valore del byte 1 è $7E_{16}$ (universale), i valori per i byte 2 e 3 vengono assegnati da MMA e AMEI, il byte 4 è definito da ciascuna specifica del profilo e il byte 5 riguarda il supporto di determinati minimi. Focalizzandosi su quest'ultimo byte, 00_{16} indica un supporto parziale (*Partial*), ed è il valore restituito da un destinatario per comunicare che supporta il profilo, ma non in tutte le sue parti; il mittente reagisce inviando un set di messaggi limitato ai requisiti minimi del profilo. Il valore 01_{16} indica il supporto minimo richiesto (*Minimum Required*),

[4] In Inglese, si tratta del *drawbar organ*; un esempio di strumento elettrico che utilizza i tiranti è l'*organo Hammond*.

Byte	Profili standard	Profili proprietari
1	$7E_{16}$	SysEx ID 1 del produttore
2	Numero di banco del profilo	SysEx ID 2 del produttore
3	Numero del profilo	SysEx ID 3 del produttore
4	Versione del profilo	Info 1 specifica del produttore
5	Livello del profilo	Info 2 specifica del produttore

Tabella 6.1. Identificazione dei profili MIDI-CI.

restituito dal destinatario nel caso in cui esso implementi in modo completo le specifiche minime del profilo. I valori nell'intervallo [02_{16}, $7E_{16}$] permettono di definire, in aggiunta al set minimo, una serie di requisiti opzionali e incrementali definiti dai progettisti del profilo (*Minimum Required plus optional, Extended Feature Sets*). Infine, il valore $7F_{16}$ indica il livello più alto (*Highest Possible*), che viene usato dal mittente per istruire il destinatario nell'abilitare tutte le caratteristiche supportate dal profilo.

Come per i messaggi *SysEx* visti nel Paragrafo 2.10.1, i produttori possono utilizzare MIDI-CI per controllare profili proprietari. In tal caso, il proprio ID esclusivo viene espresso nei primi tre byte oppure nel primo byte lasciando i due restanti a zero, a seconda del codice assegnato da MMA al produttore (*Manufacturer ID*). Per questa categoria rimangono solo gli ultimi due byte, in pratica $7 + 7$ bit, per caratterizzare il profilo.

6.3.3 Dispositivi, porte e canali MIDI

Le *inquiry* consentono a un dispositivo di restituire i profili supportati per porta e per canale. Un dispositivo può supportare un singolo profilo per un'intera porta MIDI o solo su canali specifici; analogamente esso può supportare più profili su una porta o più profili solo su canali specifici. Pertanto i messaggi di configurazione del profilo MIDI-CI possono essere indirizzati a livello di porta o per canale. Ricordando che i messaggi MIDI-CI sono SysEx, questa informazione necessaria all'indirizzamento viene veicolata utilizzando l'identificativo del dispositivo dei messaggi *SysEx* universali.

Il campo ID posto a $7F_{16}$ implica l'invio all'intera porta MIDI o la ricezione da parte di essa, mentre l'intervallo [00_{16}, $0F_{16}$] comporta l'indirizzamento ai canali da 1 a 16.

Alcuni profili si rivolgono a dispositivi che utilizzano un solo canale MIDI per il profilo. Un esemplare è dato dal profilo di uno specifico strumento musicale, ad esempio il pianoforte, che non deve essere necessariamente esteso a tutti i canali MIDI supportati; al contrario, selezionare uno specifico canale lascia disponibili i restanti canali per caricare altri profili (si pensi a un modulo sonoro multi-timbrico). A livello di scambio di messaggi, un iniziatore MIDI-CI chiede quali profili siano supportati utilizzando come indirizzo ID l'intera porta, ossia il valore $7F_{16}$. Se un dato profilo è supportato su un solo canale, il risponditore replica con l'indicazione del profilo e del canale specifico. A seguire, l'iniziatore invia il messaggio *Profile Enable* o *Profile Disable* con l'indirizzo del canale MIDI selezionato, e il risponditore invia il messaggio *Profile Enabled* o *Profile Disabled* con l'indirizzo del canale MIDI.

Esistono poi profili multicanale estesi all'intera porta, ossia ai 16 canali. Un esempio tipico è dato dal General MIDI. In questo caso,l'iniziatore MIDI-CI chiede quali profili siano supportati utilizzando l'indirizzo $7F_{16}$, e il risponditore replica con l'indirizzo $7F_{16}$. A questo punto l'iniziatore invia il messaggio di attivazione o di disattivazione del profilo con indirizzo $7F_{16}$, e il risponditore reagisce inviando il messaggio di profilo abilitato o disabilitato con l'indirizzo ID dispositivo $7F_{16}$.

Si considerino, infine, profili rivolti a più canali, ma non all'intera porta MIDI. Un esempio in tal senso è offerto da un tipico profilo di chitarra che utilizza 7 canali: 6 per le corde più un canale principale (*master*) per il controllo di parametri globali, quali il volume complessivo. Come di consueto, l'iniziatore chiede quali profili siano supportati utilizzando l'indirizzo $7F_{16}$, ma il risponditore replica con $7F_{16}$ solo se il profilo è supportato su un qualsiasi sottoinsieme di canali, mentre risponde con l'identificativo del proprio canale *master* (nell'intervallo [00_{16},

$0F_{16}$]) se il profilo è supportato da uno specifico sottoinsieme di canali. Il successivo scambio di messaggi tra iniziatore e risponditore, riguardanti l'abilitazione/disabilitazione del profilo, usa come identificativo quello del canale *master*. Normalmente il canale *master* e i canali membri vengono mantenuti distinti; i canali membri dovrebbero essere numerati consecutivamente, e il canale *master* dovrebbe occupare la posizione disponibile immediatamente sotto o sopra l'intervallo di canali membri. Esistono però varianti contemplate dalle specifiche, tra cui il fatto che un canale *master* funga anche da canale membro, a patto che le funzionalità all'interno del profilo non siano in conflitto.

6.3.4 Messaggi comuni per la gestione dei profili

Le specifiche dei profili e dei dispositivi compatibili con MIDI-CI devono supportare sei messaggi detti *Common Profile Messages*. Con riferimento alla struttura dei messaggi *SysEx* universali (Figura 6.3), essi sono caratterizzati da *Sub-ID #2* nell'intervallo [20, $2F]_{16}$. Specificamente, si tratta di:

- 20_{16} *Profile Inquiry*. Un dispositivo (iniziatore) invia tale messaggio per richiedere un elenco di profili supportati a un dispositivo connesso (risponditore). L'iniziatore può utilizzare queste informazioni per configurare automaticamente la connessione tra i dispositivi ai fini di una maggiore interoperabilità. Il messaggio può essere indirizzato a un singolo canale, per investigare le funzionalità supportate su quel canale o all'intera porta (indirizzo $7F_{16}$) per scoprire le possibilità dell'intero dispositivo;

- 21_{16} *Reply to Profile Inquiry*. Quando un dispositivo riceve il messaggio di PROFILE INQUIRY, assume il ruolo di risponditore e restituisce al chiamante, attraverso molteplici messaggi, un elenco di profili supportati. In casi particolari l'elenco può essere costituito da un solo profilo come pure da nessun profilo. Se la richiesta avviene sull'intera porta MIDI, la risposta viene parcellizzata sotto forma di molteplici messaggi, restituendo dapprima i profili abilitati solo su canali specifici, con indirizzo nell'intervallo [00, $0F_{16}$], e solo in un secondo momento tutti i profili abilitati sull'intera porta, con indirizzo $7F_{16}$. Se il risponditore non supporta alcun profilo per tutti i canali o non supporta alcun profilo, deve inviare un messaggio di risposta all'indirizzo universale senza che venga indicato alcun profilo;

- 22_{16} *Set Profile On*. Il messaggio viene inviato dall'iniziatore al risponditore per chiedere di abilitare un dato profilo;

- 23_{16} *Set Profile Off*. Il messaggio viene inviato dall'iniziatore al risponditore per chiedere di disabilitare un dato profilo;

- 24_{16} *Profile Enabled Report*. Il messaggio viene inviato per segnalare l'abilitazione di un dato profilo, il che potrebbe rappresentare la conferma di un messaggio SET PROFILE ON come pure la notifica di un evento che ha avuto luogo localmente;

- 25_{16} *Profile Disabled Report*. Il messaggio viene inviato per segnalare la disabilitazione di un dato profilo, con le stesse finalità di conferma o di notifica sopra elencate.

Un caso particolare riguarda la richiesta da parte di un iniziatore di abilitare un profilo nel risponditore, ossia *Set Profile On*, cui il risponditore non può dare seguito pur supportando il profilo in oggetto. In tal caso, la mancata conferma avverrà attraverso l'invio del messaggio *Profile Disabled Report*. La situazione complementare, in cui l'iniziatore invia *Set Profile Off* ma il profilo non può essere disabilitato dal parte del risponditore, implica l'invio, da parte di quest'ultimo, del messaggio *Profile Enabled Report*.

Un'ultima situazione da gestire riguarda la richiesta di abilitare o disabilitare un profilo che non è supportato. In tal caso, la risposta da parte del dispositivo è un messaggio *MIDI-CI NAK*. Quest'ultimo rappresenta la risposta standard nel caso in cui un dispositivo non sia in grado di comprendere un messaggio. La sua struttura è:

```
F0 7E xx 0D 7F vv ss ss ss ss dd dd dd dd F7
```

ove F0 è il byte di stato per il messaggio SysEx, complementato dall'ultimo byte F7 per il messaggio EOX, $7E_{16}$ indica un *SysEx* universale, xx è il Device ID della sorgente e identifica l'intera porta o uno specifico canale, 0D è il valore di *Sub-ID #1* per MIDI-CI, 7F è il valore di *Sub-ID #2* che identifica MIDI-CI NAK, il byte vv contraddistingue la versione/formato di MIDI-CI, e, infine, i 4 byte denotati da ss ss ss ss e i successivi 4 byte dd dd dd dd codificano il MUID di mittente e destinatario.

6.3.5 Profili mutuamente esclusivi

Alcune combinazioni di profili possono essere abilitate simultaneamente su un canale MIDI, altre no. Questa combinazione non è predefinita, essa dipende dall'implementazione dei profili da parte del singolo dispositivo.

Partendo da un esempio semplice, si consideri l'adozione del profilo di piano elettrico e del profilo degli effetti della chitarra sullo stesso canale MIDI in un modulo sonoro. Per quanto apparentemente poco sensato, i progettisti del modulo potrebbero aver previsto tale combinazione al fine di ottenere un suono di un pianoforte (o di altro strumento) processato da effetti quali phaser e overdrive.

Si consideri ora la coesistenza per lo stesso canale di profili in qualche modo omogenei, ad esempio pianoforte elettrico e organo a tiranti. Anche in questo caso i profili potrebbero essere in mutua esclusione; si pensi a un software di controllo dei parametri che riconfigura la propria interfaccia a seconda dello strumento musicale target. Alcuni dispositivi, però, ne potrebbero consentire la coesistenza, per quanto controintuitiva; è il caso di moduli sonori che, per scelta progettuale, implementino una stratificazione di più timbri simultanei.

Se un dispositivo ha un profilo abilitato e riceve un messaggio di Set Profile On per un altro profilo che non può essere supportato contemporaneamente al primo, il dispositivo dovrebbe passare al nuovo profilo e inviare due risposte: Profile Disabled Report per il primo profilo e Profile Enabled Report per il secondo.

6.3.6 Specifiche di profilo

Ogni profilo standard è definito in un documento di specifica apposito, detto *Profile Specification*. Lo scopo è descrivere i fattori che consentono un alto livello di interoperabilità tra i dispositivi che supportano il profilo.

Una *Profile Specification* definisce il comportamento di un dispositivo in risposta a messaggi MIDI che includono (ma non si limitano a):

- messaggi Note-On e Note-Off;
- messaggi che influenzano l'accordatura delle note;
- messaggi alcuni Control Change;
- messaggi *Registered Parameter Number*;
- messaggi Bank Select e Program Change;
- messaggi di sistema;
- qualsiasi messaggio *SysEx* universale applicabile;
- qualsiasi messaggio *SysEx* specifico per il profilo.

Riguardo ai Control Change necessariamente supportati, essi vengono elencati in un'appendice del documento "Common Rules for MIDI-CI Profiles". Si nota, ad esempio, che i *CC numbers* nell'intervallo $[00, 0D]_{16}$ da specifiche devono essere coperti, a eccezione dei *controller* non definiti quali 03_{16} e 09_{16}; invece, l'intervallo [0E, 1F], che nell'elenco originario è riservato a funzioni generiche o non definite, logicamente non viene contemplato.

Una *Profile Specification* può anche definire altri requisiti implementativi, che includono (ma non si limitano a):

- blocchi funzionali e aspetti topologici del dispositivo;
- polifonia minima richiesta;
- numero di canali e/o porte MIDI supportati;
- tempi di risposta ai messaggi o altri standard di performance;
- tipi di dati non MIDI supportati (ad esempio, campioni audio);
- timbri specifici.

6.4 Interscambio di proprietà MIDI-CI

L'interscambio di proprietà (*Property Exchange*) è un meccanismo per ottenere e impostare i valori delle proprietà di un dispositivo. Esso consente ai dispositivi MIDI 2.0 di interoperare, fornendo uno schema definito per descrivere come vengono trasferiti i dati delle proprietà. Esempi di possibili applicazioni includono la mappatura dei *CC numbers*, la scelta dei timbri, la modifica dello stato di un dispositivo senza alcuna conoscenza preliminare di quest'ultimo e in assenza di software dedicato a tali funzioni. Ciò significa che, in un contesto MIDI 2.0, i dispositivi possono interagire con un'ampia gamma di soluzioni HW/SW (sistemi operativi desktop, dispositivi mobili, browser Web, e via dicendo) e integrarsi maggiormente con DAW e *controller* hardware.

Per raggiungere tali scopi, MIDI 2.0 utilizza JSON come formato di codifica e i consueti messaggi *SysEx* universali relativi a MIDI-CI per veicolare l'informazione tra dispositivi.

Le regole comuni per lo scambio di proprietà vengono descritte in un apposito documento di specifica, chiamato "Common Rules for MIDI-CI Property Exchange".

6.4.1 Come avviene l'interscambio di proprietà

L'interscambio di proprietà ha luogo attraverso l'esecuzione di una serie di passaggi, e in particolare:

1. una transazione *Discovery* che, come mostrato nel Paragrafo 6.2.7, è finalizzata a:
    - ottenere i MUID di entrambi i dispositivi;
    - assicurarsi che entrambi i dispositivi coinvolti supportino lo scambio di proprietà;
    - ottenere l'identificativo per i *SysEx* del produttore di entrambi i dispositivi;
    - ottenere la famiglia, il numero di modello e il livello di revisione del software per entrambi i dispositivi;
2. una *inquiry* di tipo *Property Exchange Capabilities Transaction* per ottenere i dettagli fondamentali del supporto allo scambio di proprietà;
3. una *inquiry* di tipo *Get Property Data Transaction* per una risorsa "ResourceList" per scoprire il supporto ai dati desiderati;
4. come passaggi opzionali (ma raccomandati) e indipendenti tra loro, una *inquiry* di tipo *Get Property Data Transaction* per una risorsa "DeviceInfo", seguita da una *inquiry* di tipo *Get Property Data Transaction* per una risorsa "ChannelList", finalizzate a raccogliere informazioni sul risponditore;
5. transazioni di *Get*, *Set* o *Subscribe* per la risorsa desiderata, al fine di utilizzare i dati della proprietà associata.

Get e *Set* rappresentano le funzioni fondamentali per lo scambio di proprietà, finalizzate, rispettivamente, a recuperare e impostare i valori di una proprietà. Le rispettive *inquiry* sono dette *Get Property Data* e *Set Property Data*: avviate dall'iniziatore, esse scatenato l'invio di messaggi di risposta da parte del risponditore.

L'iniziatore può inoltre effettuare una *Subscription* riguardante una data risorsa del risponditore. Quando una risorsa sottoscritta dall'iniziatore viene modificata sul risponditore, quest'ultimo lo informa inviando un messaggio *Subscription*. In questo modo iniziatore e risponditore risultano sincronizzati riguardo alla specifica risorsa.

Si osservi che i primi quattro passaggi vanno eseguiti un'unica volta, affinché l'iniziatore possa immagazzinare dati dal risponditore, mentre l'ultimo – *Get*, *Set* e *Subscribe* – può essere ripetuto più volte.

Le risorse (*Resources*) precedentemente menzionate vengono definite, o da MMA/AMEI, o dai produttori, ai fini di descrivere proprietà (*Properties*) che possono essere recuperate o aggiornate attraverso il *Property Exchange*. Le definizioni delle risorse includono l'intento, la logica richiesta per utilizzarle e come esse si rapportino ad altre risorse. Ogni produttore può decidere quali risorse implementare in un dispositivo, ma supportare "ResourceList" è strettamente richiesto per qualsiasi dispositivo gestisca l'interscambio di proprietà.

6.4.2 Formato dei messaggi Property Exchange

L'interscambio di proprietà avviene attraverso messaggi *SysEx* universali con *Sub-ID #1* che identifica MIDI-CI. Come visto in precedenza, il campo *Sub-ID #2* determina la funzione del messaggio. A seguire, il *payload* include un campo di intestazione (*Header Data*) e uno riservato ai valori delle proprietà (*Property Data*). L'intestazione aderisce al formato JSON, mentre le proprietà possono essere a loro volta JSON con opportune restrizioni o qualsiasi altro formato, a seconda della risorsa in uso.

L'approccio generale per l'interscambio di proprietà stabilisce di utilizzare, ove possibile, messaggi MIDI 1.0. Se il valore di una data proprietà può essere impostato o modificato tramite messaggi standard, quali PROGRAM CHANGE o CONTROL CHANGE, allora le specifiche raccomandano di evitare l'impiego di messaggi SYSEX.

I messaggi *Property Exchange* aderiscono alla forma generale dei messaggi MIDI-CI, nuovamente riportata per comodità:

```
F0 7E 7F 0D yy 01 ss ss ss ss dd dd dd dd [...]  F7.
```

Il valore di *Sub-ID #2* (quinto byte), sopra rappresentato da `yy`, dipende dallo specifico messaggio all'interno della categoria dei messaggi *Property Exchange*. Facendo riferimento allo specchietto nel Paragrafo 6.2.6, si tratta della categoria identificata dal primo *nibble* di *Sub-ID #2* posto a 3_{16}. Rispetto alla forma generale dei messaggi MIDI-CI, inoltre, si nota che il campo ID (terzo byte) è posto a 7F, coinvolgendo dunque l'intera porta MIDI, e il byte riservato alla versione (sesto byte) reca il valore 01_{16}.

La parte che contraddistingue la categoria di messaggi *Property Exchange*, rispetto ad altri messaggi MIDI-CI, si concentra nel *payload*, sopra rappresentato attraverso punti di sospensione.

Prima di procedere nella trattazione, si precisa che, in ambito MIDI-CI, prende il nome di *chunk* un singolo messaggio SYSEX che costituisce un segmento di un messaggio completo di *Property Exchange*, potendo quest'ultimo risultare ripartito su più messaggi SYSEX. Si osservi che la definizione differisce considerevolmente rispetto al concetto di *chunk* negli Standard MIDI files.

Tornando ai messaggi *Property Exchange*, il *payload* è strutturato secondo la seguente successione di byte:

- 2 byte. Numero di byte nell'intestazione del presente *chunk* (*Header Data*). Questo valore viene richiamato al punto successivo come `nh`, ed è necessario specificarlo perché, contrariamente all'intestazione di altre strutture dati (ad esempio, i *chunk* degli Standard MIDI Files), in questo caso la dimensione dell'header non è fissa;
- `nh` byte. Dati di intestazione (*Header Data*), in formato JSON con restrizioni;
- 2 byte. Numero di *chunk* nella parte di dataset, posto a 00 00_{16} nel caso di dimensione non nota a priori;

- 2 byte. Identificativo numerico del presente *chunk* all'interno di una numerazione che parte da $00\ 01_{16}$;
- 1 byte. Identificativo della richiesta (*Request ID*), dettagliato nel Paragrafo 6.4.5;
- 2 byte. Numero di byte riservati ai dati della proprietà (*Property Data*) del presente *chunk*. Il valore viene richiamato al punto successivo come `nd`, ed è necessario specificarlo in quanto la dimensione dei dati non è prefissata;
- `nd` byte. Dati relativi alla proprietà (*Property Data*) del presente *chunk*. Può trattarsi di JSON con opportune restrizioni o qualsiasi altro formato, compresso o non compresso.

6.4.3 Messaggi MIDI-CI per l'interscambio delle proprietà

L'insieme di messaggi MIDI-CI utili nelle funzioni di *Property Exchange* include due categorie delle quattro definite nel Paragrafo 6.2.6. La prima di queste è data dai messaggi veri e propri di *Property Exchange*, il cui *Sub-ID #2* è:

- 30_{16} – *Inquiry: Property Exchange Capabilities*
- 31_{16} – *Reply to Property Exchange Capabilities*
- 32_{16} – *Inquiry: Has Property Data* (Reserved)
- 33_{16} – *Reply to Has Property Data* (Reserved)
- 34_{16} – *Inquiry: Get Property Data*
- 35_{16} – *Reply to Get Property Data*
- 36_{16} – *Inquiry: Set Property Data*
- 37_{16} – *Reply to Set Property Data*
- 38_{16} – *Subscription*
- 39_{16} – *Reply to Subscription*
- [3A, 3E] – valori riservati
- 3F – *Notify Message*

Sono inoltre coinvolti messaggi della categoria *Management*, già trattati in precedenza, il cui *Sub-ID #2* è:

- 70_{16} – *Discovery*
- 71_{16} – *Reply to Discovery*
- 72_{16} – *Invalidate MUID*
- $7F_{16}$ – *NAK*

6.4.4 Messaggi su *chunk* multipli

Un dispositivo può decidere di inviare il set di dati relativo a un interscambio di proprietà, come un singolo messaggio SysEx, o attraverso un insieme di messaggi SysEx detti *chunk*.

Si rammenta che, nella transazione di *Discovery* tra dispositivi, viene comunicato un valore detto *Receivable Maximum SysEx Message Size*. (si veda il Paragrafo 6.2.7). Quando un messaggio di *Property Exchange*, con il suo *payload* dedicato ai *Resource Data*, supera il limite noto per l'altro dispositivo, il mittente deve suddividere il messaggio in più *chunk*. Questo comportamento potrebbe anche essere dettato dalle sole caratteristiche del mittente, progettato per ripartire messaggi particolarmente onerosi su più SysEx.

Se l'insieme di dati viene inviato su molteplici *chunk*, all'interno dei messaggi viene specificato il *Number of Chunks in Data Set* e ciascun *chunk* viene etichettato con un numero sequenziale detto *Number of This Chunk*.

6.4.5 Request ID

Il **Request ID** è un numero nell'intervallo [0, 127] che ha tre funzioni:

1. associare tra loro più *chunk* riferibili a un singolo messaggio della famiglia *Property Exchange*. In tal caso, ogni *chunk* deve presentare lo stesso valore per l'identificatore in oggetto;

2. associare una *reply* all'*inquiry* che ha l'ha provocata. In tal caso, l'identificatore deve avere lo stesso valore nella richiesta e nella risposta relativa;

3. supportare l'invio e la ricezione di messaggi multipli da parte di un dispositivo, evitando che un messaggio di dimensioni più grandi, ripartito in molti *chunk*, blocchi richieste più piccole.

Il terzo scenario è utile per consentire a un iniziatore di avviare una nuova *inquiry* prima di ricevere la *reply* dal risponditore, che in alcuni casi potrebbe impiegare un tempo significativo per recuperare e trasferire l'insieme di dati richiesto. Più transazioni possono quindi aver luogo simultaneamente.

I valori impiegati per *Request ID* possono essere riutilizzati al completamento di una transazione. Essi, inoltre, sono univoci per una data connessione tra uno specifico iniziatore (caratterizzato dal proprio MUID) e uno specifico risponditore (caratterizzato dal proprio MUID); l'uso simultaneo dello stesso valore per connessioni tra coppie diverse di dispositivi non genera collisioni.

Una volta trascorso il tempo di *timeout*, le richieste che non ricevono risposta scadono e, a questo punto, il *Request ID* può essere riutilizzato.

6.5 Universal MIDI Packet

Il formato **Universal MIDI Packet** (UMP) è un nuovo formato per i dati definito dalle specifiche MIDI 2.0. Pensato per rappresentare i messaggi del protocollo MIDI, esso introduce il concetto di gruppo per estendere il numero di canali disponibili.

Il formato supporta una risoluzione più fine dei dati per tutti i messaggi della famiglia *Channel Voice*, aggiungendo inoltre alcuni messaggi alla famiglia al fine di fornire un maggiore controllo sulle note e sull'espressività musicale. Inoltre, MIDI 2.0 semplifica la comunicazione aggregando alcune combinazioni di messaggi in un unico messaggio.

I nuovi messaggi dati includono il cosiddetto *System Exclusive 8*, un messaggio molto simile al SysEx di MIDI 1.0 ma con un formato dei dati a 8 bit, e il *Mixed Data Set Message*, utilizzato per trasferire grandi moli di dati, compresa informazione non MIDI. Questi messaggi verranno esemplificati nel seguito.

Altra caratteristica saliente di UMP è l'introduzione di un meccanismo di marcatura temporale (*timestamping*) finalizzato alla riduzione del *jitter*, ossia la mitigazione dell'effetto di variabilità nel ritardo di consegna del pacchetto. A tale scopo è possibile premettere un *timestamp* a ciascun messaggio MIDI per migliorarne l'accuratezza temporale.

Infine, il formato UMP riserva numerose aree attualmente inutilizzate per garantire la futura espandibilità.

6.5.1 Pacchetto base e formato dei messaggi

Ciascun pacchetto in formato UMP è costituito da 1, 2, 3 o 4 parole a 32 bit. Esso contiene un intero messaggio MIDI o, in caso di messaggi che superano la dimensione di $4 \times 32 = 128$ bit, una parte di un messaggio MIDI, senza trasportare ulteriori dati; un messaggio che supera la dimensione massima supportata viene suddiviso su più pacchetti.

Il formato prevede che ogni pacchetto contenga tre campi chiamati *message type*, (*group*) e (*status*), seguiti dalla parte dedicata ai dati dei messaggi.

I 4 bit più significativi di UMP dichiarano il tipo di messaggio (*message type*, MT), che determina anche la dimensione complessiva del pacchetto. Nella fattispecie, i tipi di messaggio ammessi sono:

- MT = 0_{16} – Messaggi di utilità (*Utility Messages*), espressi su 32 bit. Allo stato attuale, questo tipo principalmente implementa il meccanismo per migliorare l'accuratezza temporale attraverso la riduzione del jitter, come mostrato nel Paragrafo 6.5.3;
- MT = 1_{16} – Messaggi di sistema (*System Messages*), espressi su 32 bit. Questo tipo consente di rappresentare in formato UMP i messaggi *System Real Time* e *System Common* propri di MIDI 1.0;
- MT = 2_{16} – Messaggi di canale MIDI 1.0 (*MIDI 1.0 Channel Voice*), espressi su 32 bit. Nel Paragrafo 6.5.2 verrà mostrato l'impacchettamento UMP di messaggi MIDI 1.0, quali NOTE-ON e PROGRAM CHANGE;
- MT = 3_{16} – Messaggi dati (*Data Messages*), espressi su 64 bit. Tra le sue diverse applicazioni, questo tipo consente di impacchettare come UMP i messaggi *System Exclusive* a 7 bit di MIDI 1.0;
- MT = 4_{16} – Messaggi di canale MIDI 2.0 (*MIDI 2.0 Channel Voice*), espressi su 64 bit. A scopo esemplificativo, nel Paragrafo 6.5.2 verrà mostrato l'impacchettamento UMP del messaggio MIDI 1.0 di NOTE-ON, analizzando le estensioni consentite da una sua rappresentazione su 8 byte, in luogo dei 3 byte di MIDI 1.0 e dei 4 byte del già citato tipo *MIDI 1.0 Channel Voice*;
- MT = 5_{16} – Messaggi dati (*Data Messages*), in questo caso espressi su 128 bit. Questo tipo include i messaggi *System Exclusive 8* e i messaggi *Mixed Data Set*;
- MT $\in [6, F]_{16}$ – Combinazioni riservate per usi futuri, ma già dimensionate per numero di bit. Ad esempio, pur non avendo ancora determinato una loro precisa funzione, è già noto che i tipi MT = B_{16} e MT = C_{16} siano riservati a pacchetti a 96 bit.

I successivi 4 bit in UMP sono dedicati al campo del gruppo (*group*). Ciascun gruppo fornisce un insieme di 16 canali MIDI indipendenti cui i messaggi di canale possono afferire. In questo modo il numero di canali supportato da 16 diventa $16 \times 16 = 256$. Ogni gruppo deve usare al proprio interno un unico protocollo MIDI, scelta attualmente limitata a MIDI 1.0 e MIDI 2.0. In un gruppo, i 16 canali MIDI in uso risultano diversi e indipendenti da quelli di qualsiasi altro gruppo, estendendo il numero effettivo di canali disponibili a 256 (Figura 6.4). Pacchetti afferenti a gruppi diversi possono essere trasmessi e ricevuti interfogliati tra loro. All'interno di un dato gruppo, si possono inviare anche messaggi non di canale; in particolare si tratta dei tipi sopra chiamati *Utility Messages*, *System Messages* e *Data Messages*. Mancando informazione di canale, essi si riguardano tutti i canali MIDI nel gruppo.

Il successivo campo incontrato è dedicato allo stato (*status*), il cui scopo è distinguere tra loro i messaggi di uno stesso tipo. Ad esempio, come detto, MT = 2_{16} denota i messaggi *Channel Voice* di MIDI 1.0, al cui interno si trovano NOTE-ON, NOTE-OFF, PROGRAM CHANGE, e via dicendo; il campo dello stato serve a discriminare tra questi. La dimensione in bit del campo dipende dal tipo di messaggio.

I rimanenti bit del pacchetto in formato UMP sono riservati ai dati, e vengono sfruttati per veicolare le restanti informazioni proprie dei messaggi MIDI 1.0, o per renderle più accurate, o per riassumere in un unico pacchetto informazioni normalmente delegate a più messaggi. Nel seguito verranno presentati alcuni casi esemplari.

La Figura 6.5 esemplifica graficamente tre esempi di strutturazione di un pacchetto in formato UMP. Dall'alto verso il basso, si tratta di un pacchetto a 32 bit con campo di stato a 8 bit (e dunque 16 bit riservati ai dati), di un pacchetto a 32 bit con campo di stato a 4 bit (e dunque 20 bit riservati ai dati), e, infine, di un pacchetto a 32 bit con campo di stato a 8 bit, indice a 16 bit e 16 bit riservati

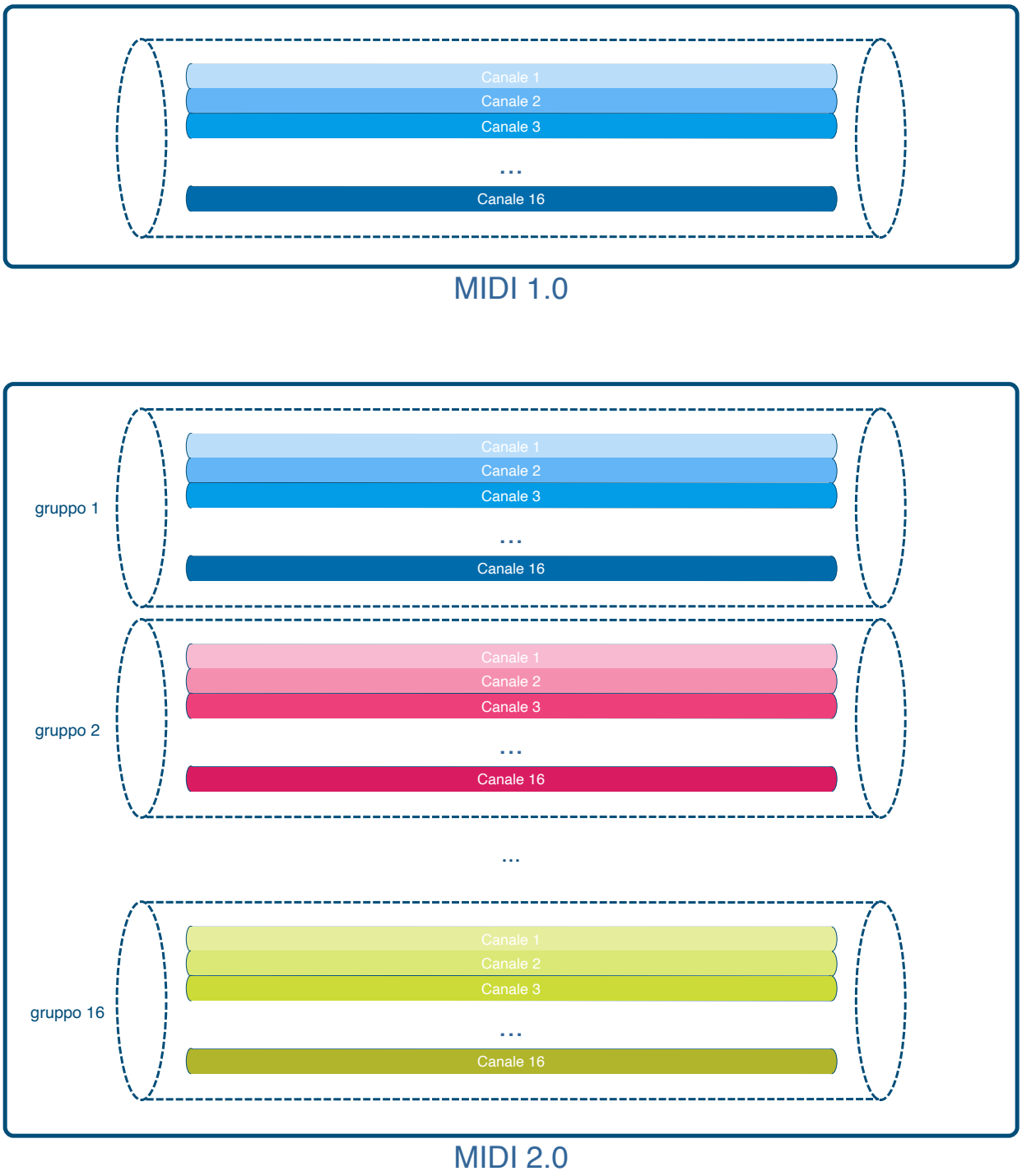


Figura 6.4. Rappresentazione grafica dell'estensione dei 16 canali MIDI 1.0 (sopra) grazie al meccanismo dei gruppi introdotto da UMP in MIDI 2.0 (sotto).

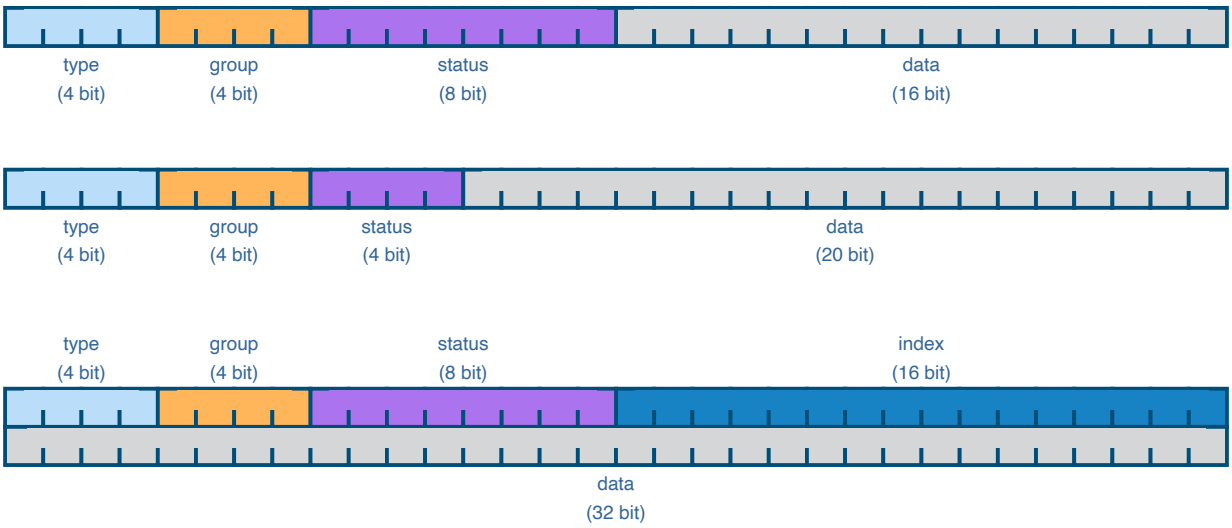


Figura 6.5. Esempi di strutturazione di pacchetti in formato UMP.

ai dati. In tutti gli esempi, si nota che la sequenza di campi si mantiene inalterata, e i primi due cambi occupano ciascuno 4 bit. Il primo esempio risponde alla struttura di un messaggio di tipo 1 – *System Message*, il secondo al tipo 0 – *Utility Message*, e il terzo al tipo 4 – *MIDI 2.0 Channel Voice Message*.

6.5.2 Esempi

In questa sezione verranno presentati alcuni esempi chiarificatori riguardo all'uso del formato UMP per la rappresentazione di messaggi MIDI-CI. Ci si soffermerà, in particolare, sui formati previsti per i messaggi della famiglia *Channel Voice*. Per ulteriori approfondimenti si rimanda al documento di specifica "Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol".

Messaggi *MIDI 1.0 Channel Voice*

Questi messaggi, il cui tipo è caratterizzato da MT = 2, sono sempre a 32 bit. Nel dettaglio, i primi 4 bit riservati al tipo contengono 0010_2, i seguenti 4 bit specificano il gruppo, i successivi 8 bit riguardano lo stato e prevedono il riversamento del byte di stato del messaggio MIDI 1.0 (identificativo della funzione e informazione di canale), infine i 16 bit di dati vengono utilizzati per codificare il primo e il secondo byte di dati. Nel caso in cui il messaggio MIDI 1.0 preveda un unico byte di dati, gli ultimi 8 bit del pacchetto vengono posti a 0.

Di conseguenza, il messaggio NOTE-ON che MIDI 1.0 codifica in base 2 come:

```
10010011 01011110 01000000
```

dà origine a un pacchetto in formato UMP quale:

```
0010gggg 10010011 01011110 01000000
```

ove i bit `gggg` identificano il gruppo.

Per un messaggio *Channel Voice* su soli due byte, quale PROGRAM CHANGE, la forma MIDI 1.0:

```
11000011 01000001
```

dà origine a un pacchetto in formato UMP quale:

```
0010gggg 11000011 01000001 00000000
```

Qualsiasi messaggio della famiglia *Channel Voice* viene ricondotto a una codifica su 32 bit, preponendo un byte che specifica tipo e gruppo. Nel caso di messaggi NOTE-OFF, NOTE-ON,

POLYPHONIC KEY PRESSURE, CONTROL CHANGE e PITCH BEND CHANGE, i tre byte restanti vengono completamente occupati dai rispettivi byte di stato e di dati, mentre, nel caso di PROGRAM CHANGE e CHANNEL PRESSURE, l'ultimo byte del pacchetto contiene un valore riservato e viene riempito di zeri.

Messaggi *MIDI 2.0 Channel Voice*

Questi messaggi, il cui tipo è caratterizzato da MT = 4, sono sempre a 64 bit. Nel dettaglio, i primi 4 bit riservati al tipo contengono 0100_2, i seguenti 4 bit specificano il gruppo, i successivi 8 bit riguardano lo stato e prevedono il riversamento del byte di stato del messaggio MIDI 1.0 (identificativo della funzione e informazione di canale), ulteriori 16 bit fungono da indice, e gli ultimi 32 bit contengono i valori di parametri e proprietà.

L'associazione di un maggior numero di bit ha un duplice impatto: da un lato, consente di inviare informazioni aggiuntive rispetto ai byte di dati MIDI 1.0 e, dall'altro, di migliorare la precisione dei valori o di estenderne l'intervallo.

A questo riguardo è significativo analizzare la rappresentazione dei messaggi di NOTE-ON e NOTE-OFF all'interno di UMP. Considerando il primo di questi, l'originaria codifica MIDI 1.0:

```
10010011 01011110 01000000
```

dà origine a un pacchetto in formato UMP quale:

```
0100gggg 10010011 01011110 tttttttt
00000000 01000000 aaaaaaaa aaaaaaaa
```

ove i bit `gggg` identificano il gruppo, il byte di stato MIDI 1.0 va a occupare il secondo byte UMP e il primo byte di dati MIDI 1.0 diviene il terzo byte UMP. A seguire, si trova un byte, denotato da $\texttt{tttttttt}_2$, che consente di definire il tipo di un attributo il cui valore verrà rappresentato nel seguito del pacchetto. Nei 2 byte successivi è possibile specificare l'informazione del secondo byte di dati MIDI 1.0, ossia la velocity, ma su un numero doppio di bit. Nell'esempio sopra si è scelto di copiare il valore anziché riscalarlo sul nuovo intervallo consentito dall'uso di 16 bit. Gli ultimi 2 byte sono dedicati a ospitare il valore numerico del tipo di attributo precedentemente indicato.

Nel caso di NOTE-ON e NOTE-OFF, le specifiche consentono di non inviare dati aggiuntivi ($\texttt{tt}_{16} = 00_{16}$), o di indicare valori specifici per il produttore ($\texttt{tt}_{16} = 01_{16}$), o di indicare valori specifici per il profilo MIDI-CI in uso ($\texttt{tt}_{16} = 02_{16}$), o, infine, di sfruttare il campo a 16 bit per il cosiddetto *Pitch 7.9* ($\texttt{tt}_{16} = 03_{16}$). In quest'ultimo caso si tratta di rimappare l'altezza della nota attraverso 7 bit per la parte intera, che definiscono il numero di semitoni rispetto alla scala temperata in accordo con MIDI 1.0, e 9 bit per un'ulteriore parte frazionaria, che permettono di suddividere il semitono con risoluzione 1/512, ottenendo così una granularità di 0,2 cent. In MIDI 1.0, gli effetti di microintonazione temporanea o di modifica persistente dell'accordatura richiedevano l'invio di ulteriori messaggi, quali PITCH BEND CHANGE o SYSEX universali di tipo *Single-note Tuning Change*; MIDI 2.0, al contrario, consente di veicolare questa informazione aggiuntiva in un singolo pacchetto.

Il formato UMP prevede per ciascun messaggio della famiglia *Channel Voice* un arricchimento delle possibilità. A titolo di esempio, per POLYPHONIC KEY PRESSURE e per CONTROL CHANGE la risoluzione aumenta da 7 a 32 bit. Per comprendere la portata della novità, basti pensare che MIDI 2.0 introduce 16.384 cosiddetti *Registered Controllers* e altrettanti *Assignable Controllers* attraverso messaggi CONTROL CHANGE di tipo *Registered Parameter Number* (RPN) e *Non-Registered Parameter Number* (NRPN) con risoluzione estesa.

Messaggi *System Exclusive 8*

I messaggi *System Exclusive 8* presentano molte somiglianze con i messaggi SYSEX previsti da MIDI 1.0, ma hanno il vantaggio di consentire un uso completo degli ottetti di bit, senza riservare il bit più

significativo come marcatura del byte di dati. Un messaggio System Exclusive 8 viene trasportato in uno o più UMP con MT = 5_{16}, quindi pacchetti propri della famiglia *Data Messages* a 128 bit.

Non è possibile tradurre un messaggio di questo tipo in un messaggio SysEx secondo il formato MIDI 1.0. Considerando che in un *setup* MIDI potrebbero convivere dispositivi MIDI 1.0 e MIDI 2.0, si prevede che molte applicazioni continueranno a utilizzare per compatibilità il formato a 7 bit, che può essere comunque impacchettato in formato UMP utilizzando messaggi dati con MT = 3_{16}.

In analogia con i precedenti esempi, viene ora illustrata la conversione di un messaggio SysEx in formato MIDI 1.0. Il generico pacchetto si presenta come:

```
0101gggg ssssbbbb xxxxxxxx dddddddd
dddddddd dddddddd dddddddd dddddddd
dddddddd dddddddd dddddddd dddddddd
dddddddd dddddddd dddddddd dddddddd
```

ove il primo *nibble* codifica MT = 5_{16}, il secondo è riservato al gruppo e il terzo allo stato. Per un messaggio *System Exclusive 8*, lo stato può essere posto a 0_{16} se il messaggio è completo, quindi occupa un unico UMP. Nel caso in cui il messaggio richieda più UMP, il valore 1_{16} denota il primo UMP di uno stream, 2_{16} caratterizza tutti gli UMP che proseguono la comunicazione *SysEx*, e 3_{16} individua l'UMP finale.

Trattandosi di un pacchetto dimensionato a 128 bit indipendentemente dalla quantità di informazione da inviare, il *nibble* $\texttt{bbbb}_2$ specifica il numero di byte validi all'interno del pacchetto.

Il byte successivo, ossia $\texttt{xxxxxxxx}_2$ nel formato sopra riportato, è l'identificativo dello stream. Questo consente di gestire più invii simultanei di SysEx differenti da parte di un dato mittente a un dato destinatario. Tra attori differenti, invece, non c'è ambiguità, in quanto la comunicazione MIDI-CI prevede sempre di specificare i MUID coinvolti.

A seguire, vengono riversati i byte tipici dei messaggi SysEx in MIDI 1.0: *Manufacturer ID* (compresi i valori speciali $7D_{16}$ e quelli dei messaggi universali), *Device ID*, *Sub-ID #1* e *Sub-ID #2* (ove presenti).

La parte rimanente del pacchetto viene utilizzata per inviare i dati veri e propri.

Messaggi *Mixed Data Set*

I messaggi *Mixed Data Set* possono trasportare qualsiasi tipo di *payload*, senza ricadere nella limitazione dei 7 bit per byte imposta dal protocollo MIDI 1.0. Si tratta di messaggi della famiglia *Data Messages* a 128 bit, con MT = 5_{16}.

Questo meccanismo è stato ideato principalmente per grandi moli di dati, in particolare per dati non MIDI. Un messaggio *Mixed Data Set* può contenere, ad esempio, informazione in formato XML o aggiornamenti del firmware dei dispositivi.

Se si vuole mantenere la compatibilità con MIDI 1.0, andrebbero utilizzati UMP per messaggi SysEx a 7 bit (MT = 3_{16}), mentre per le applicazioni MIDI che usano il formato UMP si suggerisce l'adozione dei messaggi SysEx a 8 bit, che rientrano nella stessa famiglia (MT = 5_{16}).

Nelle specifiche UMP non viene dettagliato il formato del *payload*, ma solo gli aspetti di intestazione e payload UMP.

Come detto, i dati vengono inviati in pacchetti a 128 bit, normalmente in numero superiore a uno. I vari UMP che si riferiscono a un unico *Mixed Data Set* vengono detti *Mixed Data Set Chunks*. Rispetto al formato generale di tipo 5, presentato per i messaggi *System Exclusive 8*, in questo caso lo stato consente di identificare le funzioni di *Mixed Data Set Header* (8_{16}) e di *Mixed Data Set Payload* (9_{16}).

6.5.3 Riduzione del jitter

I problemi di accuratezza temporale in un sistema MIDI dipendono da due problematiche di natura differente: il *jitter* (per quanto concerne la precisione) e la *latenza* (per quanto riguarda la

Figura 6.6. Formato generale dei messaggi per la riduzione del jitter, ossia *JR Clock* e *JR Timestamp*.

sincronizzazione). In merito a quest'ultimo punto, l'allineamento temporale tra diversi dispositivi in un *setup* è un problema assai complesso che richiederebbe la valutazione delle latenze in gioco e che esula da questa parte delle specifiche.

Un meccanismo basato su invio di messaggi in formato UMP permette invece di ridurre il fenomeno del jitter (*Jitter Reduction*, JR), ossia la variazione del tempo relativo tra due o più eventi. In questo ambito, il jitter rappresenta la variazione nella latenza misurata da un dispositivo che riceve messaggi MIDI.

La riduzione del jitter si basa sulla marcatura temporale dei pacchetti in formato UMP, e si attua attraverso due specifici messaggi: *JR Clock* e *JR Timestamp*. Si tratta di pacchetti in formato UMP di tipo MT = 0 (Utility Messages), quindi di dimensione pari a 32 bit.

Attraverso il messaggio *JR Clock*, il mittente immette sul canale trasmissivo l'informazione sul proprio valore di clock, che si propaga fino al destinatario con una latenza potenzialmente variabile di messaggio in messaggio; il destinatario, che presenta un proprio clock, può stimare il fenomeno del jitter e rilevare il suo valore massimo, agendo di conseguenza per porre rimedio.

I messaggi *JR Timestamp* servono, invece, a marcare temporalmente tutti i pacchetti UMP inviati dal mittente, a parte quelli dedicati a veicolare un nuovo *timestamp*. Nel dettaglio, *JR Timestamp* è un messaggio completo, che dunque occupa un intero UMP a 32 bit; gli UMP seguenti – purché non relativi al timestamping – si considerano allineati all'istante temporale comunicato da *JR Timestamp*, e quindi simultanei tra loro.

Si tratta di un meccanismo tra pari, che ha luogo in una comunicazione tra un mittente e un destinatario. In fase di negoziazione del protocollo, i due attori coinvolti possono stabilire di utilizzare le funzioni di *Jitter Reduction* o meno, e tale decisione rimane in vigore fino a una successiva rinegoziazione del protocollo. Ciascuno dei dispositivi coinvolti deve indicare di supportare il meccanismo, che – una volta attivato – deve essere usato in entrambe le direzioni. Al contrario, se la *Jitter Reduction* non è attiva, nessuno dei dispostivi può inviare *timestamp*. Se un dispositivo supporta il *timestamping* per una data funzione, deve supportare quella stessa funzione anche in assenza di marcatura temporale.

In aggiunta a quanto detto, il meccanismo può essere applicato (o non applicato) gruppo per gruppo, anche con sorgenti di *clock* distinte per ciascun gruppo, sebbene si tratti di una scelta implementativa inconsueta. All'interno del variegato mondo dei dispositivi MIDI 2.0, il mittente potrebbe essere in grado di gestire differenti domini temporali in parallelo, ma il destinatario no; in questo caso, in fase di negoziazione del protocollo, i dispositivi si accorderanno per attivare il meccanismo di *Jitter Reduction* su un unico gruppo.

Il formato generale dei messaggi per la riduzione del jitter, mostrato graficamente nella Figura 6.6, è:

```
0000gggg ssss0000 tttttttt tttttttt
```

Il valore dello stato, denotato da $\texttt{ssss}_2$, è posto a 0001_2 per *JR Clock* e a 0010_2 per *JR Timestamp*. I successivi 4 bit non sono utilizzati in questi messaggi, pertanto recano un valore nullo.

La rappresentazione dell'istante temporale ha luogo su 16 bit, ed è un valore temporale espresso in *ticks* della durata di 1/31250 s, circa 32 µs. Lavorando alla frequenza 1/32 MHz, la sequenza numerica disponibile si esaurisce in un tempo pari a 2,09712 secondi, dopodiché si azzera e ricomincia a crescere. Per evitare ambiguità tra cicli differenti, le specifiche prevedono che il mittente debba inviare un messaggio *JR Clock* almeno una volta ogni 250 ms.

Parte II

Programmazione MIDI

Capitolo 7

Introduzione alla programmazione MIDI

L'universo del MIDI abbraccia molti aspetti eterogenei, tra i quali protocolli per lo scambio di messaggi, specifiche hardware per porte e dispositivi, formati di file. MIDI però non è un linguaggio di programmazione. Fortunatamente al giorno d'oggi è ampio il supporto fornito dai linguaggi di programmazione al protocollo e al formato SMF.

Questa parte del testo esemplifica diversi ambienti e tecnologie per lo sviluppo di applicazioni in grado di ricevere, analizzare, processare, generare e inviare messaggi MIDI. Verranno presi in considerazione diversi approcci, ciascuno paradigmatico nel proprio ambito: la *Web MIDI API* per la programmazione web (Capitolo 8), il package *javax.sound.midi* della Java Sound API per la programmazione in Java (Capitolo 9), l'*Audio Toolbox* di MATLAB per la trattazione numerica (Capitolo 10), Csound per la programmazione timbrica e la sintesi in tempo reale (Capitolo 11) e Pure Data per la programmazione grafica in tempo reale (Capitolo 12).

7.1 Prerequisiti hardware e software

Un elaboratore con un software MIDI in esecuzione può trovare collocazione all'interno di un *setup* come un qualsiasi dispositivo compatibile con il protocollo. Quanto detto vale in generale, tanto per le applicazioni già disponibili, quanto per quelle sviluppate appositamente attraverso le tecnologie illustrate nei prossimi capitoli.

Affinché un computer sia fisicamente in grado di comunicare con altri dispositivi MIDI, non è necessaria la presenza di connettori DIN a 5 pin: allo scopo sono state utilizzate storicamente le porte seriali, e, in tempi più recenti, i protocolli USB, Firewire, Ethernet e Bluetooth.

Altra possibilità è la simulazione di un *setup* MIDI sulla singola macchina attraverso strumenti virtuali, quali *controller* e sintetizzatori software. In questo caso, per far comunicare i diversi moduli, si può rendere necessario installare anche un'applicazione che emuli porte MIDI virtuali. A titolo di esempio, è possibile scaricare e installare gratuitamente i seguenti software:

- per la categoria dei *controller*, *Virtual MIDI Piano Keyboard*,[1] disponibile per sistemi operativi Linux, *Microsoft Windows* e macOS, in grado non solo di generare ma anche di ricevere messaggi MIDI;

- per la categoria dei sintetizzatori, *VirtualMIDISynth*[2] per Windows o *FluidSynth*[3], entrambi dotati di supporto per la tecnologia *SoundFont*;

[1] https://vmpk.sourceforge.io/

[2] https://coolsoft.altervista.org/en/virtualmidisynth

[3] http://www.fluidsynth.org/

- per la categoria delle porte MIDI virtuali, *loopMIDI* per Windows,[4] *Audio MIDI Setup* nativamente incluso in macOS, specifici moduli quali *snd-seq-dummy* del framework *ALSA* per Linux.

Inoltre, per avere visibilità sui messaggi scambiati nel *setup* MIDI, esiste la categoria di software nota come *MIDI monitor*. Si tratta di strumenti che, opportunamente configurati, visualizzano e analizzano il flusso di messaggi scambiati sul canale trasmissivo, ricevendo messaggi su MIDI IN e inoltrandoli a valle su MIDI THRU/OUT. Si tratta di strumenti utili per verificare che un dispositivo o un software stiano correttamente comunicando con il setup a valle. Esistono numerose alternative, tra cui *Morson Pocket MIDI* [5] per macOS e Windows, *Snoize MIDI Monitor* [6] per macOS e *KMidimon MIDI Monitor* per Linux [7].

I software menzionati sono stati riportati a puro titolo di esempio, e le informazioni riguardo a disponibilità, sito web e tipologia di licenza sono state verificate al momento della scrittura del presente testo. Esistono molte altre soluzioni, gratuite o a pagamento, con finalità simili.

7.2 Prerequisiti in termini di competenze

In ordine alle finalità del testo, nei prossimi capitoli non sarà possibile illustrare nel dettaglio caratteristiche e sintassi dei vari linguaggi. Verranno ripresi i soli concetti fondamentali, mentre ci si dedicherà alla trattazione estesa degli argomenti relativi alla programmazione MIDI.

Per tali motivi si considerano prerequisiti la conoscenza di HTML e JavaScript per una piena comprensione del Capitolo 8, la dimestichezza con la programmazione in Java per il Capitolo 9, una basilare capacità di utilizzo di MATLAB per il Capitolo 10, la conoscenza della sintassi Csound per il Capitolo 11 e abilità di base riguardo alla programmazione grafica in Pure Data per il Capitolo 12. Esistono numerosi testi di riferimento e abbondante materiale online per integrare le conoscenze in materia e, ove possibile, il lettore verrà indirizzato verso risorse idonee per colmare le lacune.

Si considereranno acquisiti, inoltre, i concetti illustrati nella prima parte del testo, con particolare riferimento al Capitolo 2 per quanto riguarda sintassi e semantica dei messaggi MIDI e al Capitolo 5 per le specifiche del formato SMF. Il lettore potrà fare riferimento a tali capitoli per riprendere le nozioni richieste.

[4] http://www.tobias-erichsen.de/software/loopmidi.html

[5] https://www.morson.jp/pocketmidi-webpage/

[6] https://www.snoize.com/MIDIMonitor/

[7] https://kmidimon.sourceforge.io/

Capitolo 8

JavaScript: Web MIDI API

La *Web MIDI API* nasce in ambito W3C [1] nel 2015, inizialmente rilasciata dall'Audio Working Group nella forma di *Working Draft*, con l'intenzione di diventare in seguito una *W3C Recommendation*.

L'idea alla base dell'API è consentire alle applicazioni web di enumerare e selezionare i dispositivi di input e output MIDI riconosciuti nel sistema client al fine di inviare e ricevere messaggi MIDI. Lo scopo è consentire l'accesso diretto da parte delle applicazioni per browser ai dispositivi compatibili con lo standard MIDI. Una conseguenza è la possibilità di creare una nuova categoria di applicazioni Web, in grado di rispondere agli input dei *controller* MIDI: ad esempio, giochi musicali o applicazioni educative che prevedono l'interazione dell'utente con interfacce tangibili [25], prototipi per la scrittura collaborativa di partiture [16], piattaforme web per l'esplorazione di collezioni musicali [12]. Si prevede, inoltre, che la *Web MIDI API* possa essere utilizzata in collaborazione con altre API e componenti della piattaforma web, in particolare la Web Audio API, progettata dallo stesso gruppo di lavoro in ambito W3C [19].

La *Web MIDI API* dunque supporta la ricezione e l'invio di messaggi MIDI nel contesto di una performance estemporanea, come se la web app fosse un qualsiasi dispositivo posto all'interno del *setup*. Essa, invece, non è pensata per aprire ed eseguire i file MIDI. Il gruppo di lavoro ipotizza che tale funzione possa essere demandata in futuro a un'estensione dei formati supportati dall'elemento `<audio>` di HTML5.

Attualmente la *Web MIDI API* si trova nello stato di *working draft*, pertanto non costituisce ancora uno standard riconosciuto dal W3C e vincolante per mantenere la compatibilità con gli standard web. Nonostante questo, come mostrato nel Paragrafo 8.1, numerosi browser offrono un buon supporto a tale tecnologia. La versione più recente del documento è la bozza del 16 settembre 2019, disponibile all'indirizzo https://webaudio.github.io/web-midi-api/.

La realizzazione di applicazioni MIDI basate su *Web MIDI API* offre alcuni notevoli vantaggi rispetto al rilascio di software tradizionale. Innanzitutto, un'applicazione web funziona su tutte le piattaforme e i dispositivi dotati di browser, purché compatibili. In secondo luogo, il suo utilizzo non comporta la modifica di un *setup* preesistente: se il computer risulta già fisicamente collegato ad altri dispositivi MIDI, l'app si connetterà a essi senza necessità di ulteriori interventi. Infine, una web app non richiede installazione e i suoi eventuali aggiornamenti risulteranno immediatamente disponibili all'utente finale.

La *Web MIDI API* si basa su codice JavaScript, introducendo alcuni oggetti e metodi appositamente implementati per supportare il protocollo MIDI. La sintassi sarà oggetto del Paragrafo 8.2 e seguenti.

8.1 Supporto da parte dei browser

Tra i browser più diffusi in ambito desktop, la *Web MIDI API* è attualmente supportata da Microsoft Edge, Google Chrome e Opera. Manca, invece, un riscontro da parte di Mozilla Firefox e di Apple

[1] World Wide Web Consortium, https://www.w3.org/TR/webmidi/

Safari, i cui team di sviluppo – al momento della scrittura del presente testo – non sembrano intenzionati a implementarne le specifiche neanche nelle versioni a venire. La situazione attuale è indicata nella Tabella 8.1, sulla base dei dati estratti da https://caniuse.com/midi; un trattino nell'ultima colonna rappresenta il perdurare dello stato di incompatibilità, un punto interrogativo l'assenza di informazioni in materia. Le statistiche di utilizzo dei diversi browser mostrano che poco meno del 75% degli utenti è in grado di utilizzare la *Web MIDI API* sul proprio dispositivo.

Browser	Supporto	Versione più recente	Prima versione compatibile
IE	no	11 (17 ottobre 2013)	-
Edge	sì	91 (27 maggio 2021)	79 (15 gennaio 2020)
Firefox	no	89 (1 giugno 2021)	-
Chrome	sì	91 (26 maggio 2021)	43 (20 maggio 2015)
Safari	no	14.1 (26 aprile 2021)	-
Opera	sì	76 (28 aprile 2021)	30 (9 giugno 2015)
iOS Safari	no	14.6 (26 aprile 2021)	-
Opera Mini	no	all (16 marzo 2015)	-
Android WebView (Chromium)	sì	91 (26 maggio 2021)	?
Opera Mobile	sì	62 (16 febbraio 2021)	46 (23 settembre 2016)
Chrome for Android	sì	91 (26 maggio 2021)	?
Firefox for Android	no	89 (1 giugno 2021)	-
UC Browser for Android	sì	12.14 (10 gennaio 2020)	12.12 (17 agosto 2016)
Samsung Internet	sì	14.0 (16 aprile 2021)	4 (19 aprile 2016)
QQ Browser	no	10.4 (19 maggio 2020)	-
Baidu Browser	sì	43.23 (12 luglio 2019)	7.12 (1 aprile 2017)
KaiOS Browser	no	2.5 (1 giugno 2018)	-

Tabella 8.1. Compatibilità dei principali browser con la *Web MIDI API*.

Attraverso la *Web MIDI API* è del tutto naturale scrivere applicazioni che si contendono in maniera non esclusiva l'accesso alle risorse MIDI del sistema. Questa considerazione va tenuta in debito conto in fase di progettazione, in quanto il lancio simultaneo di codice concorrente potrebbe generare effetti indesiderati e di difficile comprensione. Si pensi, ad esempio, a un'interfaccia grafica dedicata al controllo di un determinato strumento musicale, in cui un dato canale viene associato a un timbro specifico; durante il funzionamento del software tale associazione potrebbe essere alterata da un messaggio PROGRAM CHANGE inviato da un altro applicativo in esecuzione in un secondo browser, il che renderebbe particolarmente complesso risalire all'origine del comportamento indesiderato.

8.2 Accesso ai dispositivi MIDI

Il metodo per accedere a tutti i dispositivi MIDI disponibili nel sistema client, sia in ingresso sia in uscita, è `navigator.requestMIDIAccess()`. Quando invocato, il metodo restituisce un oggetto `Promise`. In generale, gli oggetti `Promise` sono usati per computazioni differite e asincrone; una *promise* rappresenta un'operazione che non è ancora completata, ma lo sarà in futuro. In questo caso specifico, si tratta di una richiesta di accesso ai dispositivi MIDI presenti nel sistema dell'utente.

Il codice mostrato nel Listato 8.1 è un modo standard per richiamare il metodo, verificandone anche l'esistenza. Questo controllo consente di gestire il caso in cui il browser non supporti la *Web MIDI API*.

```
1 if (navigator.requestMIDIAccess) {
2     console.log('Il browser supporta la Web MIDI API.');
3 } else {
4     console.log('Web MIDI API non supportata dal browser.');
5 }
```

Listato 8.1. Accesso ai dispositivi MIDI.

8.3 Funzioni di *callback*

Poiché il metodo `requestMIDIAccess()` restituisce un oggetto `Promise`, si rende necessario implementare le due funzioni di *callback* che rispondono ai casi di successo e fallimento nell'accesso ai dispositivi MIDI, come mostrato nel Listato 8.2. Se il browser riesce a connettersi ai dispositivi MIDI, esso restituisce un oggetto `MIDIAccess` come argomento della funzione di *callback* relativa al successo dell'operazione. Nel listato e nel seguito della trattazione, l'istanza di tale oggetto è chiamata `midiAccess`, ma è ovviamente possibile operare una scelta diversa.

```
1  navigator.requestMIDIAccess().then(onMIDISuccess, onMIDIFailure);
2
3  function onMIDISuccess(midiAccess) {
4    console.log(midiAccess);
5
6    var inputs = midiAccess.inputs;
7    var outputs = midiAccess.outputs;
8  }
9
10 function onMIDIFailure() {
11   console.log('Impossibile accedere ai dispositivi MIDI.');
12 }
```

Listato 8.2. Funzioni di *callback* in caso di successo e fallimento.

Avviso di Google Chrome

All'invocazione di `requestMIDIAccess()`, l'attuale versione di Chrome[2] restituisce il seguente avviso su console:

> [Deprecation] Web MIDI will ask a permission to use even if the sysex is not specified in the MIDIOptions since around M82, around May 2020.

Si tratta di un avvertimento riguardo al potenziale uso non sicuro di Web MIDI: a partire da Chrome 82 si prevedeva di impedire l'esposizione della *Web MIDI API* a sorgenti considerate non sicure. La *Web MIDI API* dovrebbe essere ancora utilizzabile nelle Chrome Extensions, nelle pagine ospitate su server HTTPS e nello sviluppo in locale; negli altri casi, essa non dovrebbe funzionare e la *promise* restituita da `requestMIDIAccess()` dovrebbe essere rifiutata con la motivazione:

> DOMException: An attempt was made to break through the security policy of the user agent.

Va detto che l'implementazione di tale comportamento è già stata rimandata più volte. Ad esempio, il messaggio che imputa alla versione 82 (maggio 2020) l'introduzione della nuova caratteristica viene restituito anche dalla versione 91 (giugno 2021), la più recente tra quelle attualmente disponibili.

Chrome mostra lo stesso avviso anche nel caso di sviluppo e test in locale, teoricamente considerato sicuro secondo la definizione sopra riportata. Il motivo è che le operazioni riferibili ai messaggi SysEx possono riconfigurare i dispositivi esterni e in modo persistente. In passato, il browser chiedeva all'utente un'autorizzazione esplicita per inviare o ricevere i soli messaggi esclusivi, mentre per il futuro l'intenzione degli sviluppatori è richiedere in ogni caso la concessione delle autorizzazioni.

Allo stato attuale, la visualizzazione del *warning* durante i test del codice in locale può essere soppressa tramite un'opzione all'interno del dizionario di impostazioni per `requestMIDIAccess()`. Si tratta di `sysex`, membro booleano che stabilisce se il sistema debba essere in grado o meno di inviare oppure di ricevere messaggi esclusivi. La sintassi di `navigator.requestMIDIAccess()` diventa dunque

[2] Al momento della pubblicazione, si tratta di Google Chrome Versione 91.0.

```
requestMIDIAccess({sysex: true}).
```

facendo una finestra di dialogo in cui l'utente può esprimere il proprio consenso nell'utilizzare i dispositivi MIDI, come mostrato nella Figura 8.1.

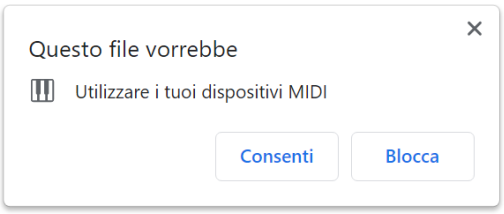


Figura 8.1. La finestra di dialogo di Google Chrome per consentire l'accesso ai dispositivi MIDI.

8.4 Enumerazione degli input e degli output

L'oggetto `MIDIAccess` è più propriamente un'interfaccia che mette a disposizione i metodi per elencare le porte MIDI di ingresso e uscita e permette di accedere ai singoli dispositivi. Come già mostrato nel Listato 8.2, a tale scopo `MIDIAccess` presenta gli attributi di sola lettura `inputs` e `outputs`. Si tratta di strutture dati simili a mappe, in cui la chiave è l'identificativo della porta MIDI e il valore è un'istanza dell'interfaccia `MIDIInput` o `MIDIOutput`.

Una volta ottenute tali collezioni di coppie (chiave, valore), è possibile accedere iterativamente alle informazioni della singola porta di input o di output. I valori rendono disponibili, attraverso la *dot notation*, i seguenti campi:

- `id`, che coincide con la chiave – esempio: "output-0";
- `manufacturer` – esempio: "Microsoft Corporation";
- `name` – esempio: "LoopBe Internal MIDI";
- `version` – esempio: "10.0".

Tra queste informazioni, assume un particolare rilievo l'identificativo della porta, che verrà ripreso più avanti per selezionare l'ingresso e/o l'uscita da connettere all'applicazione.

Il Listato 8.3 riassume quanto visto finora, chiedendo l'accesso ai dispositivi MIDI e implementando le funzioni di *callback* in caso di successo e fallimento; in aggiunta, nella prima ipotesi lo script richiama la funzione `listInputsAndOutputs()`, che mostra su console le informazioni di ogni porta di ingresso e uscita.

```
1  if (navigator.requestMIDIAccess) {
2    navigator.requestMIDIAccess().then(onMIDISuccess, onMIDIFailure);
3  } else {
4    alert('Web MIDI API non supportata dal browser in uso.');
5  }
6
7  function onMIDISuccess(midiAccess) {
8    console.log('Web MIDI API supportata, accesso ai dispositivi MIDI effettuato.');
9    listInputsAndOutputs(midiAccess);
10 }
11
12 function onMIDIFailure() {
13   console.log('Impossibile accedere ai dispositivi MIDI.');
14 }
15
16 function listInputsAndOutputs(midiAccess) {
17   for (var entry of midiAccess.inputs) {
18     var input = entry[1];
19     console.log( "Input port [type:'" + input.type + "'] id:'" + input.id + "'
           manufacturer:'" + input.manufacturer + "' name:'" + input.name + "' version:'" +
           input.version + "'" );
```

```
20    }
21    for (var entry of midiAccess.outputs) {
22      var output = entry[1];
23      console.log( "Output port [type:'" + output.type + "'] id:'" + output.id + "'
            manufacturer:'" + output.manufacturer + "' name:'" + output.name + "' version:'" +
             output.version + "'" );
24    }
25  }
```

Listato 8.3. Enumerazione degli input e degli output disponibili.

Per iterare tra gli elementi delle collezioni in `listInputsAndOutputs()` è stata utilizzata la notazione `for ... of` supportata da ECMAScript 6 (noto anche come ES6 o JavaScript 6). È comunque possibile scegliere altri approcci, come mostrato nei frammenti di codice, tra loro alternativi, del Listato 8.4.

```
1  for (var entry of midiAccess.outputs) {
2    var output = entry[1];
3    console.log("Output port [type:'" + output.type + "'] id:'" + output.id + "'
         manufacturer:'" + output.manufacturer + "' name:'" + output.name + "' version:'" +
         output.version + "'");
4  }
5
6  for (let output of midiAccess.outputs.values()) {
7    console.log("Output port [type:'" + output.type + "'] id:'" + output.id + "'
         manufacturer:'" + output.manufacturer + "' name:'" + output.name + "' version:'" +
         output.version + "'");
8  }
9
10 midiAccess.outputs.forEach(function(output, key) {
11   console.log("Output port [type:'" + output.type + "'] id:'" + output.id + "'
         manufacturer:'" + output.manufacturer + "' name:'" + output.name + "' version:'" +
         output.version + "'");
12 });
```

Listato 8.4. Alternative per iterare sugli elementi di `MIDIInputMap` e `MIDIOutputMap`.

8.5 Selezione della porta di ingresso e di uscita

In analogia con i dispositivi MIDI non virtuali, un'applicazione web MIDI può essere pensata solo per inviare messaggi MIDI, solo per riceverli, o per entrambe le cose. Un esempio della prima categoria è un *controller* MIDI virtuale con interfaccia web, un esempio della seconda un software per la valutazione automatica della performance cui un *controller* fisico invia dati, un esempio della terza un'applicazione di *MIDI monitoring* che – collocata all'interno di un *setup* – visualizza i messaggi in arrivo per poi propagarli a valle.

Una volta opportunamente gestito il caso di browser che non supporta la *Web MIDI API* e il caso di client che non presenta dispositivi MIDI associati, è opportuno consentire all'utente di scegliere le porte MIDI di ingresso e di uscita, ove richieste dall'applicativo. Un controllo comune e appropriato allo scopo è un menù a discesa, che in HTML si implementa con i tag `<select>` e `<option>`, ma chiaramente sono possibili numerose altre varianti: pulsanti di scelta, selettori grafici, ecc.

Il Listato 8.5 mostra come aggiungere le informazioni delle porte di output a un menù a discesa con `id="selectMIDIOutput"`. La prima porta viene preselezionata e il suo identificativo memorizzato nella variabile `defaultOutPort`.

```
1  function getDefaultOutput() {
2    var outputsToString = "";
3    midi.outputs.forEach(function (output, key) {
4      if (outputsToString == "") {
5        outputsToString += "<option selected='selected' value='" + key + "'>" + output.name
             + "</option>";
6        defaultOutPort = key;
7      } else
8        outputsToString += "<option value='" + key + "'>" + output.name + "</option>";
9    });
```

```
10     document.getElementById("selectMIDIOutput").innerHTML = outputsToString;
11   }
```

Listato 8.5. Visualizzazione delle porte di output.

L'ingresso e/o l'uscita a questo punto possono essere selezionati tra quelli disponibili attraverso il metodo `get()`, cui va passato l'identificativo della specifica porta. Ad esempio, chiamato `midiAccess` l'oggetto dell'interfaccia `MIDIAccess` e detto `outId` l'identificativo della porta di uscita selezionata, è possibile scrivere:

```
var output = midiAccess.outputs.get(outId);
```

La variabile così ottenuta è un oggetto dell'interfaccia `MIDIOutput`, illustrata nel prossimo paragrafo.

8.6 Invio di messaggi MIDI

L'invio di messaggi MIDI è delegato all'interfaccia `MIDIOutput`, che implementa due metodi: `send()` e `clear()`.

Il metodo `send()` ha l'obiettivo di accodare il messaggio da inviare alla porta MIDI, passato in ingresso come primo parametro. Tale argomento è un *array* di numeri interi a 8 bit senza segno, in inglese *unsigned 8-bit integer array* ossia `Uint8Array`. I valori all'interno dell'*array* corrispondono al byte di stato e ai byte di dati dei messaggi MIDI da inviare. I messaggi devono essere completi, cioè presentare il byte di stato e tutti i byte di dati attesi per quel particolare messaggio. Inoltre, il sistema non supporta il *running status*. È però possibile inviare più messaggi (completi) tramite un'unica invocazione di `send()`.

Ad esempio, la sintassi

```
output.send([0x90, 0x45, 0x7F]);
```

così come l'equivalente sintassi

```
output.send([144, 69, 127]);
```

implicano l'invio immediato alla porta di output di un messaggio NOTE-ON con pitch 69 e velocity 127. Ovviamente, è possibile all'interno di uno stesso *array* utilizzare forme di rappresentazione numerica differenti, ad esempio la base 16 per il byte di stato e la base 10 per i byte di dati.

Qualsiasi messaggio MIDI descritto nel Capitolo 2 è ammissibile come argomento del metodo `send()`. Va però considerato il caso particolare dei messaggi SYSEX, il cui supporto – per motivi di sicurezza – può essere abilitato o disabilitato attraverso il dizionario di impostazioni di `requestMIDIAccess()`. Se l'accesso esclusivo risulta inibito, il metodo risponde al tentativo di invio di un messaggio SYSEX sollevando un'eccezione `InvalidAccessError`.[3]

Esiste un secondo argomento, opzionale, per il metodo `send()`: si tratta di `timestamp`, che rappresenta l'istante al quale inviare i dati alla porta attraverso un *offset* in millisecondi. Tale valore è relativo al cosiddetto *time origin* del documento, ossia l'inizio del suo tempo di vita all'interno del browser. Un modo comune per recuperarne il valore corrente è il metodo `now()` dell'interfaccia `Performance`: la sintassi

```
window.performance.now()
```

identifica l'istante temporale relativo al *time origin* all'atto della sua invocazione.

Un valore di `timestamp` assente, nullo o precedente a quello corrente implica l'invio di dati alla porta non appena possibile, mentre l'uso di valori positivi consente di predisporre uno o più messaggi da inviare in istanti temporali futuri.

[3] Si evidenzia, per completezza, il fatto che la porta possa trovarsi in uno stato `disconnected`, caso in cui il metodo `send()` solleva un'eccezione `InvalidStateError`, oppure che la porta risulti `connected` ma la connessione sia `closed`, caso in cui il metodo tenta di aprire la porta in modo asincrono. Per ulteriori informazioni si rimanda alla documentazione ufficiale.

Un'alternativa all'invocazione di `send()` con un secondo argomento opportunamente valorizzato consiste nel memorizzare i messaggi presenti e futuri in un'apposita struttura dati, i cui valori vengono letti in modo temporizzato invocando `send()` non differite all'istante opportuno.

Un esempio di invio differito di messaggi viene mostrato nel Listato 8.6. Omettendo la prima occorrenza di `window.performance.now()`, si evidenzia come il valore 500 non rappresenti una deviazione rispetto all'istante di invocazione della funzione (caso in cui la prima nota verrebbe spenta e la seconda verrebbe accesa dopo 500 ms), ma si riferisca al *time origin* del documento. Quindi, se la funzione `playMelody()` non venisse invocata al caricamento della pagina, caso in cui si verifica `window.performance.now()` $\cong 0$, la coppia di NOTE-ON e NOTE-OFF verrebbe inviata a distanza temporale molto ravvicinata, e il secondo NOTE-ON sarebbe immediato.

Il listato esemplifica inoltre la possibilità di inviare più messaggi MIDI in un'unica invocazione di `send()`.

```
1  function playMelody() {
2    var output = midi.outputs.get(defaultOutPort);
3    output.send([0x90, 60, 96])
4    output.send([0x80, 60, 0, 0x90, 62, 96], window.performance.now() + 500);
5    output.send([0x80, 62, 0, 0x90, 64, 96], window.performance.now() + 1000);
6    output.send([0x80, 64, 0, 0x90, 60, 96], window.performance.now() + 1500);
7    output.send([0x80, 60, 0], window.performance.now() + 2500);
8  }
```

Listato 8.6. Invio differito di messaggi.

L'altro metodo dell'interfaccia `MIDIOutput` è `clear()`, che elimina dalla coda di `MIDIOutput` qualsiasi dato non ancora inviato. A titolo di esempio, l'eventuale presenza di `output.clear()` nel Listato 8.6 prima dell'invocazione di alcune `send()` avrebbe impedito l'invio dei successivi messaggi.

Si osservi che, contrariamente a quanto ipotizzabile, non si tratta di un metodo per bloccare l'esecuzione di messaggi già accodati ma differiti nel tempo. A riprova di questo, si implementi una funzione che richiama `clear()` in risposta alla pressione di un pulsante, e la si invochi prima dell'esecuzione di alcune `send()` differite: questo non impedisce ai messaggi MIDI futuri di essere inviati.

Per preimpostare una sequenza di messaggi e mantenere la possibilità di interromperne l'esecuzione, serve un approccio differente, che memorizzi gli eventi futuri in un'opportuna struttura dati i cui messaggi vengono letti e inviati all'istante opportuno con `send()` non differite. In altre parole, ad essere differito è il richiamo di funzioni `send()` che agiscono istantaneamente. In JavaScript, allo scopo, si usano i cosiddetti *timing events* quali `setTimeout()` e `setInterval()`. Per bloccare l'invio dei messaggi si può quindi agire sui *timing events* prima che questi richiamino nuove `send()`.

Al momento della stesura del testo, Google Chrome non riconosce il metodo `clear()`, al contrario di altri browser quali Microsoft Edge.

Si evidenzia un'osservazione generale sulle applicazioni pensate per generare messaggi di performance quali NOTE-ON e NOTE-OFF. In assenza di un messaggio PROGRAM CHANGE iniziale sui canali in uso, il sintetizzatore destinatario assegnerà loro il timbro di default (tipicamente quello con *program number* 0) o manterrà il timbro precedentemente selezionato. Per questo motivo, una volta scelta la porta di output, è sempre opportuno impostare esplicitamente i *program number* desiderati tramite PROGRAM CHANGE. Una gestione ancora migliore consisterebbe nel riportare i valori del sintetizzatore ai default desiderati tramite l'invio di appositi CONTROL CHANGE: si pensi agli effetti di *pan* e *chorus*, al volume di canale, e via dicendo.

8.7 Ricezione di messaggi MIDI

I messaggi MIDI vengono scambiati attraverso l'oggetto `MIDIMessageEvent`. Per raccogliere i messaggi è possibile aggiungere funzioni di *callback* dette *listener* agli ingressi. Affinché l'applicazione sia in grado di reagire alla ricezione di messaggi, il primo passaggio consiste

nell'aggiunta di un *listener* `onmidimessage` a ciascuno degli ingressi da gestire. La relativa funzione di *callback* verrà richiamata ogni volta che giunge un messaggio.

Il Listato 8.7 mostra come iterare sugli ingressi impostando per ciascuno di essi un *listener* che associa un'unica funzione di *callback* chiamata `getMIDIMessage()`. Nell'esempio a quest'ultima viene semplicemente richiesto di mostrare il messaggio `midiMessage` su console. Il Listato 8.9, presentato nel prossimo paragrafo, rappresenta un'evoluzione della funzione di *callback* in cui i messaggi MIDI vengono visualizzati all'interno della finestra del browser e in maniera più intelligibile.

Ovviamente non è necessario associare un *listener* a tutti gli ingressi: in analogia con la selezione dell'output è possibile lasciare all'utente la scelta di quale porta o quali porte "ascoltare".

```
1  function onMIDISuccess(midiAccess) {
2    for (var input of midiAccess.inputs.values()) {
3      input.onmidimessage = getMIDIMessage;
4    }
5  }
6
7  function getMIDIMessage(midiMessage) {
8    console.log(midiMessage);
9  }
```

Listato 8.7. Ricezione e gestione di messaggi MIDI in ingresso.

Un oggetto `MIDIMessageEvent` presenta molte informazioni di potenziale utilità, ma l'aspetto di maggior interesse è la proprietà `data`, una struttura dati di tipo `Uint8Array` che contiene il byte di stato e i byte di dati del messaggio MIDI.

Si osservi che più applicazioni possono risultare contemporaneamente in ascolto della stessa connessione MIDI in ingresso, tanto all'interno di un'unica istanza del browser (ad esempio in pagine aperte su tab differenti) quanto in molteplici istanze dello stesso browser o di browser differenti. Questo consente, ad esempio, di avviare un *MIDI monitor* e, contemporaneamente, un browser game leggendo gli stessi messaggi in ingresso.

8.8 Esempi

In questa sezione vengono presentati e commentati due esempi significativi per mostrare le potenzialità della *Web MIDI API*: un'applicazione web per generare messaggi MIDI a partire dall'interazione dell'utente con i controlli della pagina (Paragrafo 8.8.1) e uno strumento per monitorare i messaggi in ingresso a una porta MIDI (Paragrafo 8.8.2). Infine, nel Paragrafo 8.8.3) vengono resi disponibili i collegamenti ad alcune risorse online, tra cui le implementazioni più complete dei listati commentati e altri semplici esempi.

8.8.1 Tastiera virtuale

Il Listato 8.8 mostra un esempio completo di codice HTML, CSS e JavaScript per la realizzazione di un *controller* virtuale a tastiera da 2 ottave. Lo strumento invia messaggi di NOTE-ON, NOTE-OFF e PROGRAM CHANGE a un sintetizzatore.

Alla pressione del tasto sinistro del mouse su uno dei `<div>` che rappresentano i tasti, viene richiamata la funzione `sendNoteOn()`, mentre, al rilascio del tasto e all'uscita dal `<div>`, viene invocata la funzione `sendNoteOff()`.

Inoltre, è presente un selettore che consente di scegliere il timbro attraverso il richiamo della funzione `sendProgramChange()`, invocata anche all'ultima riga della funzione di *callback* `onMIDISuccess()`, in modo da impostare un timbro prescelto. Questo aspetto è fondamentale, perché garantisce che all'avvio dell'esecuzione dello script il sintetizzatore sia correttamente istruito sulla *patch* da caricare; in altro caso il *synth* continuerebbe a utilizzare l'ultimo strumento impostato per il canale nel corso di una performance precedente, o – in assenza di una esplicita ridefinizione – lo strumento di default, tipicamente la prima *patch* GM disponibile (Acoustic Grand Piano).

Nel listato, la variabile `midi` contiene un oggetto `MIDIAccess` e la variabile `selectedOutput` rappresenta l'oggetto `MIDIOutput` riferito alla porta d'uscita selezionata.

Per questo esempio è anche disponibile una versione online già implementata ed estesa nelle funzionalità (si veda il Paragrafo 8.8.3).

```
<!DOCTYPE html>
<html lang="it">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <style>
    div {
      clear:left;
      padding:0;
      padding-top:30px;
      cursor:default;
    }

    div.white_key {
      clear:none;
      height:100px;
      width:38px;
      border:#000 solid 1px;
      background-color:#EEE;
      float:left;
      cursor:pointer;
    }

    div.black_key {
      position:absolute;
      height:50px;
      width:20px;
      border:0;
      background-color:#000;
      cursor:pointer;
    }
  </style>
  <title>Tastiera virtuale</title>
  <script>
  var midi = null;
  var selectedOutput = null;

  function startMIDI() {
    if (navigator.requestMIDIAccess) {
      navigator.requestMIDIAccess().then(onMIDISuccess, onMIDIFailure);
    } else {
      document.getElementById("div_keyboard").style.display = "none";
      document.getElementById("div_options").style.display = "none";
      document.getElementById("log_string").innerHTML = "Attenzione: browser non
          compatibile!";
    }
  }

  function onMIDISuccess(midiAccess) {
    document.getElementById("log_string").innerHTML = "MIDI rilevato e pronto!";
    midi = midiAccess;
    getDefaultOutput();
    sendProgramChange(0);
  }

  function onMIDIFailure(msg) {
    document.getElementById("log_string").innerHTML = "Impossibile accedere alle risorse
        MIDI. Messaggio di errore: " + msg;
  }

  function changeOutput(outputId) {
    selectedOutput = midi.outputs.get(outputId);
  }

  function getDefaultOutput() {
    var outputsToString = "";
    midi.outputs.forEach(function (output, key) {
      if (outputsToString == "") {
        outputsToString += "<option selected='selected' value='" + key + "'>" + output.
            name + "</option>";
```

```
67          selectedOutput = midi.outputs.get(key);
68        } else {
69          outputsToString += "<option value='" + output.id + "'>" + output.name + "</option>
                ";
70        }
71      });
72      document.getElementById("chooseOutput").innerHTML = outputsToString;
73    }
74
75    function sendNoteOn(pitch) {
76      var noteOnMsg = [0x90, pitch, 96];
77      selectedOutput.send(noteOnMsg);
78    }
79
80    function sendNoteOff(pitch) {
81      var noteOffMsg = [0x80, pitch, 0];
82      selectedOutput.send(noteOffMsg);
83    }
84
85    function sendProgramChange(patch) {
86      var allNotesOffMsg = [0xB0, 123, 0];
87      selectedOutput.send(allNotesOffMsg);
88      var programChangeMsg = [0xC0, patch];
89      selectedOutput.send(programChangeMsg);
90    }
91    </script>
92  </head>
93
94  <body onload="startMIDI()">
95    <h1>Tastiera virtuale</h1>
96    <p id="log_string"></p>
97    <div id="div_keyboard">
98      <div class="white_key" onmousedown="sendNoteOn(60)" onmouseup="sendNoteOff(60)"
            onmouseout="sendNoteOff(60)"></div>
99      <div class="white_key" onmousedown="sendNoteOn(62)" onmouseup="sendNoteOff(62)"
            onmouseout="sendNoteOff(62)"></div>
100     <div class="white_key" onmousedown="sendNoteOn(64)" onmouseup="sendNoteOff(64)"
            onmouseout="sendNoteOff(64)"></div>
101     <div class="white_key" onmousedown="sendNoteOn(65)" onmouseup="sendNoteOff(65)"
            onmouseout="sendNoteOff(65)"></div>
102     <div class="white_key" onmousedown="sendNoteOn(67)" onmouseup="sendNoteOff(67)"
            onmouseout="sendNoteOff(67)"></div>
103     <div class="white_key" onmousedown="sendNoteOn(69)" onmouseup="sendNoteOff(69)"
            onmouseout="sendNoteOff(69)"></div>
104     <div class="white_key" onmousedown="sendNoteOn(71)" onmouseup="sendNoteOff(71)"
            onmouseout="sendNoteOff(71)"></div>
105     <div class="white_key" onmousedown="sendNoteOn(72)" onmouseup="sendNoteOff(72)"
            onmouseout="sendNoteOff(72)"></div>
106     <div class="white_key" onmousedown="sendNoteOn(74)" onmouseup="sendNoteOff(74)"
            onmouseout="sendNoteOff(74)"></div>
107     <div class="white_key" onmousedown="sendNoteOn(76)" onmouseup="sendNoteOff(76)"
            onmouseout="sendNoteOff(76)"></div>
108     <div class="white_key" onmousedown="sendNoteOn(77)" onmouseup="sendNoteOff(77)"
            onmouseout="sendNoteOff(77)"></div>
109     <div class="white_key" onmousedown="sendNoteOn(79)" onmouseup="sendNoteOff(79)"
            onmouseout="sendNoteOff(79)"></div>
110     <div class="white_key" onmousedown="sendNoteOn(81)" onmouseup="sendNoteOff(81)"
            onmouseout="sendNoteOff(81)"></div>
111     <div class="white_key" onmousedown="sendNoteOn(83)" onmouseup="sendNoteOff(83)"
            onmouseout="sendNoteOff(83)"></div>
112     <div class="white_key" onmousedown="sendNoteOn(84)" onmouseup="sendNoteOff(84)"
            onmouseout="sendNoteOff(84)"></div>
113     <div class="black_key" onmousedown="sendNoteOn(61)" onmouseup="sendNoteOff(61)"
            onmouseout="sendNoteOff(61)" style="margin-left:30px"></div>
114     <div class="black_key" onmousedown="sendNoteOn(63)" onmouseup="sendNoteOff(63)"
            onmouseout="sendNoteOff(63)" style="margin-left:70px"></div>
115     <div class="black_key" onmousedown="sendNoteOn(66)" onmouseup="sendNoteOff(66)"
            onmouseout="sendNoteOff(66)" style="margin-left:150px"></div>
116     <div class="black_key" onmousedown="sendNoteOn(68)" onmouseup="sendNoteOff(68)"
            onmouseout="sendNoteOff(68)" style="margin-left:190px"></div>
117     <div class="black_key" onmousedown="sendNoteOn(70)" onmouseup="sendNoteOff(70)"
            onmouseout="sendNoteOff(70)" style="margin-left:230px"></div>
118     <div class="black_key" onmousedown="sendNoteOn(73)" onmouseup="sendNoteOff(73)"
            onmouseout="sendNoteOff(73)" style="margin-left:310px"></div>
119     <div class="black_key" onmousedown="sendNoteOn(75)" onmouseup="sendNoteOff(75)"
```

```
            onmouseout="sendNoteOff(75)" style="margin-left:350px"></div>
120       <div class="black_key" onmousedown="sendNoteOn(78)" onmouseup="sendNoteOff(78)"
            onmouseout="sendNoteOff(78)" style="margin-left:430px"></div>
121       <div class="black_key" onmousedown="sendNoteOn(80)" onmouseup="sendNoteOff(80)"
            onmouseout="sendNoteOff(80)" style="margin-left:470px"></div>
122       <div class="black_key" onmousedown="sendNoteOn(82)" onmouseup="sendNoteOff(82)"
            onmouseout="sendNoteOff(82)" style="margin-left:510px"></div>
123     </div>
124     <div id="div_options">
125       <p>
126         <label for="chooseOutput">Output MIDI</label><br>
127         <select id="chooseOutput" onchange="changeOutput(value)">
128         <option>Nessuno</option>
129         </select>
130       </p>
131       <p>
132         Strumento<br>
133         <input type="radio" name="instrument" id="piano" checked="checked" onclick="
              sendProgramChange(0)" />
134         <label for="piano">Pianoforte</label>
135         <input type="radio" name="instrument" id="harpsichord" onclick="sendProgramChange(6)
              " />
136         <label for="harpsichord">Clavicembalo</label>
137         <input type="radio" name="instrument" id="organ" onclick="sendProgramChange(19)" />
138         <label for="organ">Organo</label>
139         <input type="radio" name="instrument" id="celesta" onclick="sendProgramChange(8)" />
140         <label for="celesta">Celesta</label>
141       </p>
142     </div>
143   </body>
144   </html>
```

Listato 8.8. Codice completo per la realizzazione di una tastiera virtuale.

8.8.2 MIDI Monitor

In questo paragrafo si mostra un esempio di *MIDI monitor*, ossia un'applicazione che presenta in maniera intelligibile all'utente l'elenco di messaggi MIDI in arrivo sulla porta di ingresso selezionata. L'interfaccia grafica è volutamente minimale, rappresentata da sole stringhe di testo.

Per brevità ci si limita a gestire i messaggi di NOTE-ON, NOTE-OFF, PROGRAM CHANGE e PITCH BEND CHANGE, ma il prototipo potrebbe essere facilmente esteso ad altri messaggi. Le stringhe di testo vengono aggiunte a `<div id="console">` posto all'interno di `<body>`.

```
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4     <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5     <title>MIDI Monitor</title>
6     <script>
7     function clearConsole() {
8       document.getElementById("latest").innerHTML = "Latest message received:<br /><br />
            none";
9       document.getElementById("console").innerHTML = "";
10    }
11
12    function startMIDI() {
13      if (navigator.requestMIDIAccess) {
14        navigator.requestMIDIAccess().then(onMIDISuccess, onMIDIFailure);
15      } else {
16        document.getElementById("console").innerHTML = "Il browser in uso non supporta la
              Web MIDI API";
17      }
18    }
19
20    function onMIDISuccess(midiAccess) {
21      midiAccess.inputs.forEach(function (input, key) {
22        input.onmidimessage = getMIDIMessage;
23      });
24    }
25
26    function onMIDIFailure() {
```

```
27      console.log('Could not access your MIDI devices.');
28    }
29
30    function getMIDIMessage(midiMessage) {
31      var stringToAdd = "<p>";
32
33      stringToAdd = "From: " + midiMessage.srcElement.name + "<br />";
34
35      // valori in formato binario ed esadecimale
36      for (var i = 0; i < midiMessage.data.length; i++) {
37        var pad = "00000000";
38        var value2nopad = midiMessage.data[i].toString(2);
39        var value2 = pad.substring(0, pad.length - value2nopad.length) + value2nopad;
40        var value16 = midiMessage.data[i].toString(16);
41        stringToAdd += "<tt>" + value2 + " [0x" + value16 + "]" + "</tt>   ";
42      }
43
44      stringToAdd += "<br />";
45
46      // interpretazione dei valori
47      var isNoteOnNoteOff = false;
48      var isProgramChange = false;
49      var isPitchBend = false;
50      for (var i = 0; i < midiMessage.data.length; i++) {
51        if (i == 0) { // byte di stato
52          var firstNibble = Math.floor(midiMessage.data[i] / 16); //int division by 16
53          switch (firstNibble) {
54            case 8: stringToAdd += "NoteOff"; isNoteOnNoteOff = true; break;
55            case 9: stringToAdd += "NoteOn"; isNoteOnNoteOff = true; break;
56            case 12: stringToAdd += "Program Change"; isProgramChange = true; break;
57            case 14: stringToAdd += "Pitch Bend Change"; isPitchBend = true; break;
58            default: stringToAdd += "Unmanaged message"; break;
59          }
60          var secondNibble = midiMessage.data[i] % 16 + 1;
61          stringToAdd += ", Channel " + secondNibble;
62        } else if (i == 1) { // primo byte di dati
63          if (isNoteOnNoteOff) {
64            stringToAdd += ", Pitch " + midiMessage.data[i];
65          } else if (isProgramChange) {
66            stringToAdd += ", Program #" + (midiMessage.data[i] + 1);
67          }
68        } else if (i == 2) { // secondo byte di dati
69          if (isNoteOnNoteOff) {
70            stringToAdd += ", Velocity " + midiMessage.data[i];
71          } else if (isPitchBend) {
72            var aggregatedValue = midiMessage.data[i] * 128 + midiMessage.data[i-1];
73            stringToAdd += ", MSB and LSB aggregated value " + aggregatedValue + " (" + (
                  aggregatedValue - 8192) + " with respect to center)";
74          }
75        }
76      }
77      stringToAdd += "</p><hr />";
78      document.getElementById("latest").innerHTML = "Latest message received:";
79      document.getElementById("console").innerHTML = stringToAdd + document.getElementById("
            console").innerHTML;
80    }
81    </script>
82  </head>
83
84  <body onload="startMIDI()">
85    <h1>MIDI Monitor</h1>
86    <a href="#" onclick="clearConsole()">Clear the console</a>
87    <p id="latest">Latest message received:<br /><br />none</p>
88    <div id="console"></div>
89  </body>
90  </html>
```

Listato 8.9. Codice completo per la realizzazione di un *MIDI Monitor*.

8.8.3 Esempi online

Come materiale supplementare per lo studio e la comprensione da parte dell'utente delle potenzialità della *Web MIDI API*, vengono qui forniti i collegamenti ad alcune risorse disponibili

online.

Il codice – volutamente racchiuso in un unico file sorgente che comprende HTML, JavaScript ed eventuali CSS – può essere osservato utilizzando gli strumenti per sviluppatori messi a disposizione dei browser. Google Chrome, ad esempio, presenta le funzionalità "Visualizza sorgente pagina" e "Ispeziona".

Xilofono

URL: https://www.lim.di.unimi.it/download/midi/web/xilofono.html
Nell'esempio, si mostra la generazione di messaggi MIDI di NOTE-ON e NOTE-OFF a partire dal'interazione via mouse con aree sensibili di un'immagine mappata.

Tastiera virtuale

URL: https://www.lim.di.unimi.it/download/midi/web/tastiera.html
L'applicativo genera messaggi MIDI di NOTE-ON e NOTE-OFF a partire dall'interazione via mouse con aree sensibili di un'interfaccia di tastiera ricostruita attraverso elementi `<div>`. Viene inoltre gestito il messaggio di PROGRAM CHANGE. Si tratta di un'implementazione più estesa dell'esempio mostrato nel Paragrafo 8.8.1.

Armonizzatore

URL: https://www.lim.di.unimi.it/download/midi/web/armonizzatore.html
Questo esempio costituisce un ampliamento del prototipo della tastiera virtuale. Alla pressione di un singolo tasto della tastiera viene associata la generazione non di singoli suoni, bensì di accordi il cui modello può essere scelto dall'utente. Il prototipo si limita a triadi maggiori e minori, nello stato fondamentale e di I e II rivolto, ma può essere facilmente esteso ad altri modelli di accordo (ad esempio, settime di varia specie, ecc.).

Melodia

URL: https://www.lim.di.unimi.it/download/midi/web/melodia.html
Il presente prototipo esemplifica una possibile strada per risolvere il problema principale introdotto dal `send()` differito, ossia l'impossibilità di svuotare il buffer dei messaggi MIDI futuri in risposta a un comando di stop. Per farlo, la melodia immessa dall'utente viene salvata all'interno di un'opportuna struttura dati, ed è stato introdotto un meccanismo di temporizzazione per leggere i contenuti di tale struttura al momento opportuno ed effettuare `send()` istantanee. La pressione del pulsante Stop non agisce dunque su una sequenza di messaggi MIDI, bensì interrompe la lettura temporizzata della struttura dati.

Capitolo 9

Java: javax.sound.midi

Il package *javax.sound.midi* si colloca all'interno della cosiddetta *Java Sound API* e presenta un insieme di classi e interfacce dedicate al MIDI. Esso fornisce supporto all'implementazione di operazioni di I/O, *sequencing* e sintesi dei dati MIDI.

Un primo aspetto riguarda le interfacce specializzate per il MIDI. Le interfacce Java rappresentano uno degli strumenti volto a introdurre astrazione nella progettazione del codice. Un'interfaccia è una classe completamente astratta, utilizzata per racchiudere metodi dai corpi vuoti. Per poter accedere a tali metodi, l'interfaccia deve essere "implementata" da una classe, il che avviene, nel codice, attraverso la parola riservata `implements`. Tale meccanismo richiama, dal punto di vista sintattico, quello dell'ereditarietà tipico della programmazione orientata agli oggetti, ma con una differenza sostanziale: la nuova classe implementa (`implements`) l'interfaccia anziché estendere (`extends`) la classe genitrice. Dal punto di vista concettuale, invece, le differenze tra i due approcci sono molto più profonde, ma una loro trattazione esaustiva esulerebbe dalle finalità del testo.

Il package presenta le seguenti **interfacce**:

- `ControllerEventListener`. Interfaccia da implementare da parte delle classi le cui istanze richiedono di ricevere notifiche quando un *sequencer* [1] ha processato un certo tipo di evento CONTROL CHANGE;
- `MetaEventListener`. Interfaccia da implementare da parte delle classi le cui istanze richiedono di ricevere notifiche quando un *sequencer* ha processato un certo tipo di evento METAEVENT;
- `MidiChannel`. Interfaccia le cui istanze rappresentano singoli canali MIDI;
- `MidiDevice`. Interfaccia di base per tutti i dispositivi MIDI;
- `MidiDeviceReceiver`. Estensione dell'interfaccia `Receiver` per descrivere un ricevitore MIDI, ossia un connettore di input per un `MidiDevice`;
- `MidiDeviceTransmitter`. Estensione dell'interfaccia `Transmitter` per descrivere un trasmettitore MIDI, ossia un connettore di output per un dispositivo MIDI (`MidiDevice`);
- `Receiver`. Interfaccia le cui istanze ricevono oggetti `MidiEvent` e, in generale, rispondono svolgendo una determinata operazione su questi, come, ad esempio, interpretare tali eventi per generare suono o produrre output MIDI;
- `Sequencer`. Interfaccia per rappresentare un dispositivo hardware o software che manda in esecuzione una sequenza MIDI;

[1] In questo contesto, per *sequencer* si intende un oggetto istanza dell'interfaccia `Sequencer` descritta nel seguito.

- `Soundbank`. Interfaccia che descrive un set di strumenti che può essere caricato all'interno di un sintetizzatore;
- `Synthesizer`. Interfaccia per rappresentare un generatore di suono;
- `Transmitter`. Interfaccia le cui istanze inviano oggetti `MidiEvent` a uno o più ricevitori (`Receiver`).

Le **classi** contenute nel package sono:

- `Instrument`. Classe le cui istanze rappresentano algoritmi di sintesi del suono, normalmente progettati per emulare uno specifico strumento musicale reale o per ottenere un determinato effetto sonoro;
- `MetaMessage`. Classe per rappresentare meta-messaggi MIDI che, pur non essendo significativi per i sintetizzatori, possono trovare posto nei file MIDI ed essere interpretati dai *sequencer*;
- `MidiDevice.Info`. Classe contenente informazioni sui dispositivi MIDI, quali il nome, il produttore e una descrizione testuale;
- `MidiEvent`. Classe per rappresentare eventi MIDI, ossia messaggi MIDI e relativi *timestamp* espressi in *ticks*. Tali informazioni possono essere memorizzate in un file MIDI o in una sequenza, ossia – in questo contesto – un oggetto `Sequence`;
- `MidiFileFormat`. Classe che si focalizza sul formato dei file MIDI, descrivendone il tipo, la lunghezza e informazioni sulla temporizzazione;
- `MidiMessage`. Classe base per i messaggi MIDI;
- `MidiSystem`. Classe che fornisce accesso alle risorse del sistema MIDI presenti sulla macchina, elencando i dispositivi disponibili quali sintetizzatori, *sequencer* e porte MIDI di ingresso e uscita;
- `Patch`. Classe per rappresentare locazioni di memoria all'interno di un sintetizzatore in cui vengono caricati e salvati singoli strumenti;
- `Sequence`. Classe finalizzata a descrivere una sequenza, ossia una struttura dati che contiene informazione musicale che può essere eseguita da un *sequencer*;
- `Sequencer.SyncMode`. Classe per rappresentare uno dei modi in cui la nozione di tempo di un *sequencer* può essere sincronizzata con un dispositivo *master* o *slave*;
- `ShortMessage`. Classe dedicata ai messaggi MIDI costituiti da un byte di stato e, al più, due byte di dati;
- `SoundbankResource`. Classe per descrivere qualsiasi risorsa audio memorizzata in un banco di suoni [2];
- `SysexMessage`. Classe per rappresentare i messaggi esclusivi di sistema;
- `Track`. Classe che descrive flussi indipendenti di eventi MIDI, ossia dati MIDI con *timestamp*. Un file MIDI può contenere al proprio interno più tracce, ossia più istanze della classe in oggetto;
- `VoiceStatus`. Classe che descrive lo stato corrente di una delle voci prodotte da un sintetizzatore [3].

[2] In questo contesto, per banco di suoni si intende un oggetto istanza dell'interfaccia `Soundbank`.

[3] In questo contesto, per sintetizzatore si intende un oggetto istanza dell'interfaccia `Synthesizer`.

Le **eccezioni** introdotte dal package, infine, sono:

- `InvalidMidiDataException`. Eccezione sollevata in caso di dati MIDI non validi;
- `MidiUnavailableException`. Eccezione sollevata quando il componente MIDI invocato non può essere aperto o creato per via della sua indisponibilità.

Una trattazione dettagliata di tutte le classi e interfacce elencate esulerebbe dagli scopi del presente testo. Nel seguito ci si limiterà a descrivere le operazioni più comuni, quali la connessione di trasmettitori e ricevitori, la creazione, ricezione, gestione e invio dei messaggi MIDI e il *sequencing* a partire da sequenze generate appositamente o contenute in un file MIDI. Per maggiori dettagli si rimanda il lettore alla documentazione ufficiale Java SE 10 & JDK 10.[4]

Tutti gli esempi presentati e commentati nel seguito del capitolo sono a disposizione del lettore all'URL https://ludovico.lim.di.unimi.it/download/libri/midi/java.zip.

9.1 Dispositivi, trasmettitori e ricevitori

La *Java Sound API* specifica un'architettura di *routing* dei messaggi MIDI basata sulla connessione di più moduli. Ciascuno di questi è specializzato nel portare a termine un determinato compito e può essere interconnesso ad altri moduli per consentire il flusso di dati MIDI.

Il modulo principale è costituito dall'interfaccia `MidiDevice`. L'obiettivo è rappresentare diverse categorie di dispositivi MIDI, tra cui i *sequencer* (che registrano, mandano in esecuzione, caricano e manipolano sequenze di messaggi MIDI con *timestamp*), i sintetizzatori (che generano suono quando ricevono opportuni messaggi MIDI) e le porte di ingresso e uscita (attraverso le quali i dati MIDI fluiscono da e verso dispositivi esterni). Le interfacce `Synthesizer` e `Sequencer`, trattate, rispettivamente, nei Paragrafi 9.3 e 9.8, estendono l'interfaccia `MidiDevice` per specializzarne le funzioni.

Un oggetto `MidiDevice` tipicamente presenta uno o più oggetti che implementano le interfacce `Receiver` o `Transmitter`, il cui scopo è dotare il dispositivo di connettori per collegarlo ad altri `MidiDevice`, permettendo così un flusso dei dati MIDI in ingresso e uscita. Il *routing* viene realizzato connettendo un `Transmitter` di un `MidiDevice` a un `Receiver` di un altro `MidiDevice`. L'interfaccia `MidiDevice` offre metodi per determinare quanti trasmettitori e ricevitori siano supportati in modo concorrente e per accedere a tali oggetti.

Una porta MIDI di output normalmente presenta, quanto meno, un `Receiver` per ricevere i messaggi in uscita dal dispositivo. Una porta MIDI di input, analogamente, deve avere almeno un `Transmitter` per propagare i messaggi in arrivo. Si tratta in apparenza di un comportamento controintuitivo, che va però rivalutato nell'ottica del *routing*, come mostrato graficamente nella Figura 9.1 per un *sequencer*. Anche se una porta di output è evidentemente pensata per inviare messaggi a una catena a valle, e quindi per trasmetterli in uscita, dal punto di vista del *routing* essa presenta uno o più ricevitori, perché – per svolgere la propria funzione – deve essere connessa a un generatore di messaggi che funge da sorgente (in questo caso, un *sequencer*). Per le porte di input vale l'approccio complementare: esse devono acquisire informazioni MIDI dalla catena a monte, e quindi ricevere messaggi, ma dal punto di vista del *routing* presentano dei trasmettitori per inoltrare tali messaggi al dispositivo.

Dopo aver trattato trasmettitori e ricevitori riguardo alle porte di ingresso e uscita, si considerino ora altre due rilevanti categorie di dispositivi MIDI. Un sintetizzatore normalmente risponde ai messaggi inviati a uno o più dei componenti `Receiver` di cui dispone. Un *sequencer* completo, invece, deve avere uno o più `Receiver` per svolgere la funzione di registrazione e uno o più `Transmitter` per inviare messaggi durante la riproduzione.

L'interfaccia `Transmitter` presenta metodi per impostare e interrogare i ricevitori ai quali il trasmettitore manda i propri messaggi MIDI. L'interfaccia `Receiver`, invece, presenta un metodo

[4] https://docs.oracle.com/javase/10/docs/api/javax/sound/midi/package-summary.html

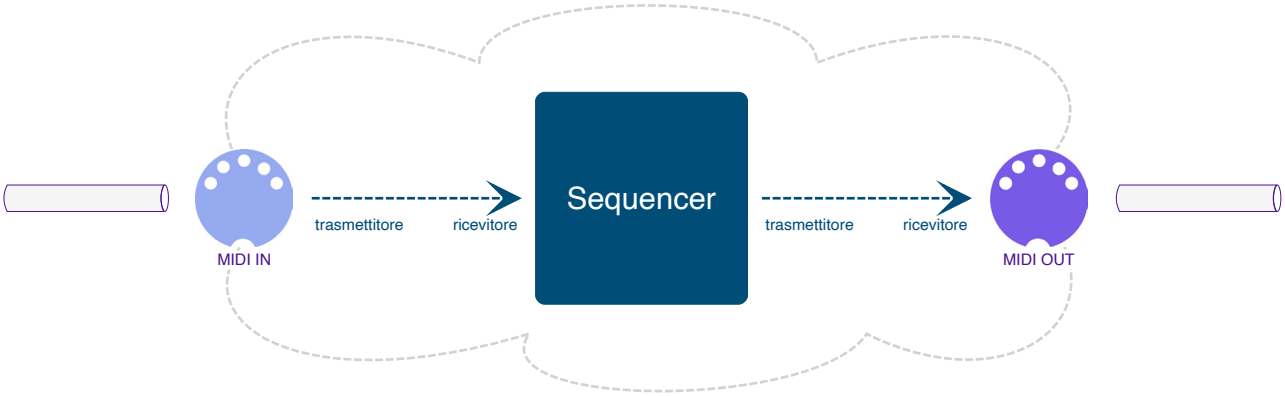


Figura 9.1. Relazione tra trasmettitori e ricevitori per il *routing* della comunicazione MIDI.

per inviare al ricevitore stesso un messaggio. Quindi, sebbene l'invio di un messaggio costituisca – dal punto di vista logico – un'operazione propria di un trasmettitore, il metodo `send()` deve essere invocato sul ricevitore destinatario del messaggio. Riguardo alla sintassi di `send()`, essa prevede come argomenti non solo il messaggio MIDI (istanza della classe `MidiMessage` o, più precisamente, di una sua specializzazione), ma anche un valore numerico intero che permette di differire l'invio di un certo numero di microsecondi. Esempi di `send()` verranno mostrati nel Paragrafo 9.4.6.

La connessione tra `Transmitter` e `Receiver` viene stabilita quando un trasmettitore imposta il proprio ricevitore con una sintassi quale:

```
trasmettitore.setReceiver(ricevitore);
```

La connessione può essere interrotta richiamando i metodi `close()` di cui dispongono entrambe le interfacce, in modo da liberare le risorse e renderle disponibili per nuove connessioni.

I concetti ora esposti richiedono esempi completi di codice per acquisire maggiore chiarezza. Nel seguito verranno mostrati, innanzitutto, casi in cui la comunicazione tra dispositivi MIDI viene stabilita grazie alla connessione tra i loro trasmettitori e ricevitori; sarà prima necessario, però, introdurre la classe `MidiSystem`. Per questo motivo i primi esempi inerenti a `Transmitter` e `Receiver` saranno mostrati solo nel Paragrafo 9.2.1. Altro aspetto da approfondire è l'invio e la ricezione di messaggi MIDI sulla connessione così stabilita, ma – per poter essere comprese – tali operazioni richiedono la trattazione della classe `MidiMessage`; i relativi esempi saranno dunque presentati nel Paragrafo 9.4.6.

9.2 Le classi MidiSystem e MidiDevice.Info

La classe `MidiSystem` riveste una fondamentale importanza per la parte di *Java Sound API* dedicata al MIDI, in quanto essa fornisce accesso alle risorse MIDI presenti all'interno del sistema: sintetizzatori, *sequencer* e connettori MIDI di ingresso e uscita. Un'applicazione MIDI, all'avvio, invoca tipicamente i metodi di `MidiSystem` per apprendere quali dispositivi risultino installati e ottenere l'accesso a quelli richiesti dall'applicazione stessa. La classe presenta inoltre metodi per leggere file, flussi di dati e URL che contengono file MIDI o banchi di suoni.

Non è possibile istanziare un oggetto di classe `MidiSystem`. Tutti i metodi della classe sono statici, quindi vengono richiamati usando la notazione puntata (*dot notation*) a partire dal nome della classe stessa.

All'interno del sistema è possibile identificare alcuni dispositivi MIDI di default, in particolare un ricevitore (istanza di `Receiver`), un *sequencer* (istanza di `Sequencer`), un sintetizzatore (istanza di `Synthesizer`) e un trasmettitore (istanza di `Transmitter`). Essi sono elencati nel file "sound.properties", contenuto in una sottocartella dell'installazione del *Java Runtime Environment* (JRE). Se tali proprietà non risultano definite all'interno del file subentrano le proprietà di sistema. In entrambi i casi i dispositivi di default vengono recuperati attraverso i metodi

`getReceiver()`, `getSequencer()`, `getSynthesizer()` e `getTransmitter()`. Le chiamate dei metodi elencati restituiscono istanze di classi specializzate che saranno descritte nel seguito. Ad esempio, `getReceiver()` restituisce un oggetto di classe `Receiver`, che – a sua volta – rappresenta un'estensione dell'interfaccia `MidiDevice`. Se una di tali chiamate non ha successo, viene sollevata un'eccezione `MidiUnavailableException`. Nel Paragrafo 9.2.1 verrà mostrato un esempio completo.

Un altro metodo fondamentale messo a disposizione da `MidiSystem` è `getMidiDeviceInfo()`, che restituisce una lista di oggetti di classe `MidiDevice.Info`. Attraverso un opportuno ciclo che scorra la struttura dati è possibile ottenere informazioni sulle risorse MIDI presenti nel sistema. Su un oggetto `MidiSystem` è inoltre possibile invocare il metodo `getMidiDevice()`, cui è necessario passare un'istanza di `MidiDevice.Info` per recuperare uno specifico dispositivo.

Anche la classe `MidiDevice.Info`, al pari di `MidiSystem`, è statica. Essa contiene principalmente metodi per investigare le proprietà dei dispositivi presenti nel sistema. In particolare, `getDescription()` restituisce una descrizione del dispositivo in formato stringa, `getName()` il suo nome, `getVendor()` il suo produttore, `getVersion()` la versione, `toString()` una descrizione testuale estesa dell'oggetto. Esistono, inoltre, metodi di servizio, quali `equals()`, che verifica l'identità rispetto a un altro oggetto, e `hashCode()`, che ne estrae il codice di *hash*[5].

9.2.1 Esempi

Gli esempi qui riportati riguardano alcune operazioni fondamentali inerenti a `MidiSystem`, classe statica che dà accesso ai dispositivi MIDI presenti nel sistema.

Informazioni su una specifica risorsa

Il codice mostrato nel Listato 9.1 recupera alcune informazioni del sintetizzatore di default, e in particolare il suo nome e il numero massimo di ricevitori e trasmettitori supportati.

```
1  import javax.sound.midi.*;
2  import javax.sound.midi.MidiDevice.Info;
3
4  public class GetInfo {
5    public static void main(String[] args) throws MidiUnavailableException {
6      Synthesizer synth = MidiSystem.getSynthesizer();
7
8      System.out.println("Nome: " + synth.getDeviceInfo().getName());
9      System.out.println("Numero massimo di ricevitori: " + synth.getMaxReceivers());
10     System.out.println("Numero massimo di trasmettitori: " + synth.getMaxTransmitters());
11
12     Info deviceToSearch = synth.getDeviceInfo();
13     MidiDevice device = MidiSystem.getMidiDevice(deviceToSearch);
14     System.out.println("\nNome: " + device.getDeviceInfo().getName());
15     System.out.println("Numero massimo di ricevitori: " + device.getMaxReceivers());
16     System.out.println("Numero massimo di trasmettitori: " + device.getMaxTransmitters());
17   }
18 }
```

Listato 9.1. Recupero di informazioni sul sintetizzatore di default.

Il codice nell'esempio è volutamente ridondante, vista la sua natura didattica. Infatti, una volta recuperato il sintetizzatore di default (riga 6), viene mostrata la possibilità di ottenere informazioni su di esso attraverso due strade normalmente alternative:

- per via diretta, ossia invocando i metodi a partire dall'istanza di `Synthesizer` (righe 8–10);
- recuperando la corrispondente istanza di `MidiDevice` a partire da `getMidiDevice()` (riga 13), e mostrando le relative informazioni a partire da essa (righe 14–16). Al di fuori di un contesto didattico, il processo di estrarre da un oggetto `Synthesizer` le sue informazioni (riga 12)

[5] In gergo matematico e informatico, viene detta *hash* una funzione non invertibile che mappa una stringa di lunghezza arbitraria in una stringa di lunghezza predefinita.

e sfruttarle per recuperare da `MidiSystem` un oggetto `MidiDevice` risulterebbe inutilmente artificioso.

Si osservi che la *signature* di `main()` consente al corpo della funzione di sollevare eccezioni `MidiUnavailableException`. Questo permette di gestire il caso in cui non sia disponibile un sintetizzatore di default.

Osservando i contenuti dello standard output, il sintetizzatore di default tipicamente presenterà 0 trasmettitori, il che è coerente con le funzioni attese da questa categoria di dispositivi. Il numero di ricevitori potrebbe essere -1, valore in apparenza sorprendente, ma usato in realtà per indicare una quantità virtualmente infinita di ricevitori.

Risorse disponibili nel sistema

Il codice mostrato nel Listato 9.2 elenca le informazioni di tutti i dispositivi MIDI presenti nel sistema.

```
1  import javax.sound.midi.*;
2
3  public class MidiDevices {
4    public static void main(String[] args) throws MidiUnavailableException {
5      MidiDevice.Info[] midiDevices = MidiSystem.getMidiDeviceInfo();
6      System.out.println("DISPOSITIVI PRESENTI NEL SISTEMA\n");
7      for (MidiDevice.Info info : midiDevices) {
8        System.out.println("Nome: " + info.getName());
9        System.out.println("Descrizione: " + info.getDescription());
10       System.out.println("Produttore: " + info.getVendor());
11       System.out.println("Versione: " + info.getVersion());
12       System.out.println("Classe: " + info.getClass().toString());
13       MidiDevice device = MidiSystem.getMidiDevice(info);
14       if (device instanceof Receiver)
15         System.out.println("Tipologia: ricevitore\n");
16       else if (device instanceof Sequencer)
17         System.out.println("Tipologia: sequencer\n");
18       else if (device instanceof Synthesizer)
19         System.out.println("Tipologia: sintetizzatore\n");
20       else if (device instanceof Transmitter)
21         System.out.println("Tipologia: trasmettitore\n");
22       else
23         System.out.println("Tipologia: sconosciuta\n");
24     }
25
26     System.out.println("\nDISPOSITIVI DI DEFAULT\n");
27     Receiver rec = MidiSystem.getReceiver();
28     System.out.println("Ricevitore di default: " + rec);
29     Sequencer seq = MidiSystem.getSequencer();
30     System.out.println("Sequencer di default: " + seq);
31     Synthesizer synth = MidiSystem.getSynthesizer();
32     System.out.println("Sintetizzatore di default: " + synth);
33     Transmitter trans = MidiSystem.getTransmitter();
34     System.out.println("Trasmettitore di default: " + trans);
35   }
36 }
```

Listato 9.2. Recupero di tutte le risorse MIDI disponibili nel sistema.

Il codice è in generale autoesplicativo. Gli aspetti di maggior rilievo si trovano a riga 5, ove vengono recuperati i dispositivi e memorizzati in un *array*, e a riga 7, ove un ciclo scorre gli oggetti all'interno della struttura dati.

Il blocco di codice alle righe 26–34 mostra le informazioni sui dispositivi di default del sistema.

Collegamento di un trasmettitore a un ricevitore

Il Listato 9.3 mostra il collegamento tra il trasmettitore del *sequencer* di default e il ricevitore del sintetizzatore di default.

```
1  import java.io.File;
2  import java.io.IOException;
```

```
3  import javax.sound.midi.*;
4
5  public class TransmitterToReceiver {
6    public static void main(String[] args) throws MidiUnavailableException {
7      Sequencer seq = MidiSystem.getSequencer();
8      seq.open();
9      System.out.format("Sequencer di default: %s (%s)\n", seq.getDeviceInfo().getName(),
           seq.getDeviceInfo().getVendor());
10     Transmitter seqTrans = seq.getTransmitter();
11     Synthesizer synth = MidiSystem.getSynthesizer();
12     synth.open();
13     System.out.format("Sintetizzatore di default: %s (%s)\n", synth.getDeviceInfo().
           getName(), synth.getDeviceInfo().getVendor());
14     Receiver synthRcvr = synth.getReceiver();
15     seqTrans.setReceiver(synthRcvr);
16     try {
17       seq.setSequence(MidiSystem.getSequence(new File("midisong.mid")));
18     } catch (InvalidMidiDataException | IOException ex) {}
19     seq.start();
20   }
21 }
```

Listato 9.3. Collegamento di un trasmettitore a un ricevitore.

Sequencer e sintetizzatore di default vengono determinati a partire da `MidiSystem` attraverso la sintassi alle righe 7 e 11. Poiché nell'esempio si vuole che il *sequencer* invii messaggi al sintetizzatore, è necessario recuperare un trasmettitore del *sequencer* (riga 10), un ricevitore del sintetizzatore (riga 14) e creare il collegamento tra essi (riga 15). L'eventuale assenza di *sequencer* e sintetizzatore di default viene gestita sollevando un'eccezione `MidiUnavailableException` (riga 6).

Le interfacce `Synthesizer` e `Sequencer` verranno affrontate nei Paragrafi 9.3 e 9.8. Per il momento, al lettore basti sapere che i due dispositivi, per poter interagire, devono essere "aperti" attraverso la chiamata di `open()` (righe 8 e 12). Al termine della comunicazione essi dovrebbero essere "chiusi" esplicitamente tramite l'invocazione di `close()`; tale azione, però, è stata volutamente omessa dall'esempio in quanto inibirebbe all'istante la riproduzione del file MIDI.

Le righe 16–19 hanno il fine di verificare il collegamento tra trasmettitore e ricevitore, ma presuppongono la conoscenza della classe `Sequence` (Paragrafo 9.7) e dell'interfaccia `Sequencer` (Paragrafo 9.8) e la disponibilità di un file MIDI chiamato "midisong.mid". Al momento è possibile ignorare queste righe di codice e differire una loro piena comprensione. L'apertura e l'esecuzione di un file MIDI da parte di un *sequencer* verranno illustrate nel Paragrafo 9.8.2.

Collegamento di un trasmettitore a più ricevitori

Il Listato 9.4 mostra come i messaggi MIDI provenienti da una sorgente possano essere inoltrati simultaneamente a più ricevitori.

```
1  import javax.sound.midi.*;
2
3  public class TransmitterToReceivers {
4    public static void main(String[] args) throws MidiUnavailableException {
5      // sintetizzatore
6      Synthesizer synth = MidiSystem.getSynthesizer();
7      synth.open();
8      // sequencer
9      Sequencer seq = MidiSystem.getSequencer();
10     seq.open();
11     // sorgente dei messaggi
12     MidiDevice inputPort;
13     // qui va selezionata e aperta la sorgente dei dati...
14     Transmitter inPortTrans1, inPortTrans2;
15     Receiver synthRcvr;
16     Receiver seqRcvr;
17     try {
18       // primo dispositivo: sintetizzatore
19       inPortTrans1 = inputPort.getTransmitter();
20       synthRcvr = synth.getReceiver();
21       inPortTrans1.setReceiver(synthRcvr);
```

```
22          // secondo dispositivo: sequencer
23          inPortTrans2 = inputPort.getTransmitter();
24          seqRcvr = seq.getReceiver();
25          inPortTrans2.setReceiver(seqRcvr);
26        } catch (MidiUnavailableException e) {
27          // gestione delle eccezioni
28        }
29      }
30  }
```

Listato 9.4. Collegamento di un trasmettitore a più ricevitori.

Nell'esempio si sceglie arbitrariamente di inviare i messaggi al sintetizzatore e al sequencer di default, mentre non viene determinato il dispositivo trasmettitore, operazione da eseguire alla riga 13. Chiaramente, è possibile operare scelte diverse, ad esempio attraverso un costrutto iterativo sui dispositivi di ingresso e uscita MIDI nel sistema.

Il cuore dell'esempio consiste nell'assegnare un trasmettitore del dispositivo sorgente a differenti istanze di `Transmitter` (righe 19 e 23), cui vengono poi associati i ricevitori dei due dispositivi destinatari (righe 21 e 25). In realtà, ogni invocazione di `getTransmitter()` restituisce un nuovo trasmettitore, fino a quando il dispositivo non esaurisce quelli a propria disposizione, il che genera un'eccezione.

La tecnica illustrata permette di inviare gli stessi messaggi MIDI a più destinatari, creando una forma di connessione spesso chiamata in letteratura *fan out* o *splitter*.

9.3 L'interfaccia Synthesizer

L'interfaccia `Syntehsizer`, una delle principali estensioni dell'interfaccia `MidiDevice`, è specializzata per rappresentare le informazioni e le funzionalità tipiche di un sintetizzatore.

9.3.1 Metodi principali

Innanzi tutto, l'interfaccia `Syntehsizer` eredita dall'interfaccia `MidiDevice` i metodi `open()` e `close()`, necessari per allocare risorse di sistema all'apertura del dispositivo e per liberarle quando non sono più necessarie.

In aggiunta alle informazioni ricavabili su qualsiasi oggetto `MidiDevice` e contenute in `MidiDevice.Info`, tale interfaccia presenta un insieme di metodi che hanno a che vedere, da un lato, con le caratteristiche di un sintetizzatore e, dall'altro lato, con gli strumenti musicali (timbri) messi a disposizione.

Nella prima categoria rientrano metodi quali `getLatency()`, che restituisce la latenza in microsecondi nel processare i comandi, o `getMaxPolyphony()`, che dà il numero massimo di voci simultanee. Tramite opportune chiamate è possibile investigare altri parametri caratteristici. Ad esempio, `getChannels()` restituisce i canali controllati dal sintetizzatore, e `getVoiceStatus()` lo stato corrente delle voci prodotte dal dispositivo (voce attiva o meno, numero di banco, numero di canale, numero di nota, numero di programma, volume). A quest'ultimo riguardo, per motivi di brevità, non sarà possibile addentrarsi maggiormente nell'argomento, pertanto si rimanda il lettore interessato alla documentazione della classe `VoiceStatus`.

Alla seconda categoria appartengono metodi operanti sui banchi di suoni, oggetti della classe `Soundbank`, quali `getDefaultSoundbank()` e `isSoundbankSupported()`. Sono presenti metodi per caricare e rimuovere singoli strumenti – `loadInstrument()`, `unloadInstrument()` – o più strumenti posti all'interno di uno specifico banco – `loadInstruments()`, `loadAllInstruments()`, `unloadInstruments()`, `unloadAllInstruments()`.

Il metodo `getAvailableInstruments()` recupera l'elenco di timbri supportati dal sintetizzatore, mentre `getLoadedInstruments()` restituisce l'elenco dei soli strumenti attualmente caricati.

9.3.2 Esempi

In questa sezione verranno mostrati esempi riguardanti il recupero di informazioni sui sintetizzatori e alcune basilari operazioni sui timbri.

Recupero di informazioni dai sintetizzatori

Il Listato 9.5 ha l'obiettivo di scorrere tutte le risorse MIDI disponibili nel sistema, filtrare solo quelle che ricadono nella categoria dei sintetizzatori e fornire informazioni specifiche su tali dispositivi, quali la latenza e la polifonia massima supportata.

```
1  import javax.sound.midi.*;
2
3  public class SynthInfo {
4    public static void main(String[] args) throws MidiUnavailableException {
5      Instrument[] instrList;
6      MidiDevice.Info[] infos = MidiSystem.getMidiDeviceInfo();
7      for (MidiDevice.Info deviceInfo : infos) {
8        MidiDevice device = MidiSystem.getMidiDevice(deviceInfo);
9        if (device instanceof Synthesizer) {
10         Synthesizer synth = (Synthesizer)device;
11         System.out.println("Sintetizzatore corrente: " + synth.getDeviceInfo().toString())
               ;
12         System.out.println("Numero massimo di voci supportate (polifonia): " + synth.
               getMaxPolyphony() + " voci");
13         System.out.println("Latenza di elaborazione: " + synth.getLatency() + "
               microsecondi\n");
14         System.out.println("Elenco degli strumenti disponibili");
15         instrList = synth.getAvailableInstruments();
16         System.out.println("Numero totale: " + instrList.length);
17         int counter = 1;
18         for (Instrument currInstr : instrList)
19           System.out.println("   " + counter++ + ". " + currInstr.getName());
20         System.out.println("-----");
21       }
22     }
23   }
24 }
```

Listato 9.5. Recupero di informazioni dai sintetizzatori.

Si osservi che, per recuperare informazioni sul sintetizzatore, non è necessario invocare il metodo `open()` né, di conseguenza, il metodo `close()`. Difatti, nell'esempio corrente, il sintetizzatore non svolge alcuna funzione attiva (emissione di suono, sostituzione del banco, caricamento di strumenti, ecc.), ma viene semplicemente interrogato riguardo alle proprie peculiarità.

Sostituzione degli strumenti caricati

Il Listato 9.6 mostra alcuni esempi di operazioni sui timbri caricati nel sintetizzatore.

```
1  import javax.sound.midi.*;
2
3  public class SynthChangeInstruments {
4    static void showInstrumentList(Instrument[] instrList) {
5      System.out.println("Numero totale: " + instrList.length);
6      int counter = 1;
7      for (Instrument currInstr : instrList) {
8        System.out.println("   " + counter + ". " + currInstr.getName());
9        counter++;
10     }
11   }
12
13   public static void main(String[] args) throws MidiUnavailableException {
14     Synthesizer synth = MidiSystem.getSynthesizer();
15     System.out.println("Il sintetizzatore di default nel sistema e' " + synth.
           getDeviceInfo().toString());
16     System.out.println("Numero massimo di voci supportate (polifonia): " + synth.
           getMaxPolyphony() + " voci");
```

```
17      System.out.println("Latenza di elaborazione: " + synth.getLatency() + " microsecondi\n
            ");
18      System.out.println("Elenco degli strumenti disponibili");
19      showInstrumentList(synth.getAvailableInstruments());
20      System.out.println("\nElenco degli strumenti caricati");
21      showInstrumentList(synth.getLoadedInstruments());
22
23      synth.open();
24      synth.unloadAllInstruments(synth.getDefaultSoundbank());
25      Instrument[] instrList = synth.getAvailableInstruments();
26      synth.loadInstrument(instrList[13]);
27      synth.loadInstrument(instrList[60]);
28      System.out.println("\nNuovo elenco di strumenti caricati");
29      showInstrumentList(synth.getLoadedInstruments());
30      synth.close();
31    }
32  }
```

Listato 9.6. Caricamento di strumenti in un sintetizzatore.

Il codice della funzione `main()` è organizzato in due blocchi. Il primo di questi (righe 14–21) riprende nella sostanza l'esempio precedente. La visualizzazione dell'elenco degli strumenti disponibili e di quelli attualmente caricati, demandata alla funzione `showInstrumentList()` (righe 4–11), è funzionale ad apprezzare le differenze introdotte dal secondo blocco (righe 23–30). Vengono infatti rimossi tutti gli strumenti del banco di suoni di default, e, tra quelli disponibili, ne vengono selezionati due da aggiungere nuovamente al sintetizzatore [6]. Infine, viene mostrata la nuova collezione di strumenti caricati (riga 29).

Si osservi che, in assenza della chiamata di `open()` alla riga 23, la seconda parte del codice non avrebbe consentito di caricare i due strumenti. Quindi l'operazione di "apertura" del sintetizzatore non è finalizzata solo alla produzione di suono, ma anche al richiamo di operazioni che ne modificano lo stato.

9.4 La classe MidiMessage

La classe `MidiMessage` supporta la rappresentazione di qualsiasi messaggio MIDI, considerando non solo i messaggi MIDI 1.0 descritti nel Capitolo 2, ma anche i meta-eventi trattati nel Capitolo 5, significativi per i *sequencer* e tipicamente salvati all'interno degli Standard MIDI Files. La classe dà accesso a tre tipi di informazione relativi a un messaggio MIDI:

1. il byte di stato;
2. la lunghezza complessiva espressa in byte;
3. l'*array* di byte contenente il messaggio completo.

La classe è astratta e include metodi per recuperare tali valori, ma non per impostarli. Quest'ultima funzione è delegata alle sottoclassi specializzate `MetaMessage`, `ShortMessage` e `SysexMessage`. In questo modo le sottoclassi si occupano di verificare che l'*array* di dati specifichi un messaggio MIDI valido e completo.

9.4.1 Rappresentazione dei byte

Talvolta le applicazioni Java devono scambiare dati MIDI con sorgenti esterne al codice, quali *sequencer* e sintetizzatori (fisici o virtuali). Poiché lo standard MIDI rappresenta i dati in byte, sembrerebbe del tutto naturale utilizzare tanto per la rappresentazione interna quanto per l'interscambio di informazioni il tipo *byte* messo a disposizione da Java. Si tratta, evidentemente, di valori esprimibili attraverso sequenze di 8 bit, ma interpretati come numeri con segno rappresentati in complemento a 2.

[6] Si segnala al lettore l'opportunità di verificare, all'interno del codice, che gli strumenti da aggiungere (righe 26–27) effettivamente esistano, controllo omesso dal Listato 9.6 per motivi di brevità.

Per via di questa difformità nell'interpretazione dei bit, la *Java Sound API* codifica i dati MIDI adottando, piuttosto, il tipo *integer*. In tal modo, i metodi che recuperano messaggi MIDI o loro sottoparti restituiscono valori interi; e, quando si rende necessario passare byte di stato e di dati come parametri a un metodo, è possibile utilizzare valori numerici interi, peraltro espressi in qualsiasi base.

Rimane da risolvere il problema dell'interoperabilità con le sorgenti esterne, che inviano e ricevono dati come sequenze di byte nell'accezione più generale. Esiste una soluzione semplice per trasformare i dati MIDI da byte a interi:

```
int intData = (int)(byteData & 0xFF);
```

Di base, si tratta di un'operazione di *and bit a bit* che coinvolge come operandi il valore da trasformare e una sequenza di 8 bit posti a 1. Si può notare come, prima dell'assegnamento a una variabile intera, venga forzato un *cast* al tipo *integer*.

9.4.2 Metodi principali

Il metodo `getLength()` restituisce la lunghezza in byte dell'intero messaggio, ossia byte di stato e byte di dati. Il metodo `getMessage()` recupera tutti i dati che costituiscono il messaggio, incluso il byte di stato, nella forma di un *array* di byte. Il metodo `getStatus()` si focalizza sul solo byte di stato, restituito come valore intero per i motivi sopra descritti. La classe `MidiMessage` presenta, inoltre, un metodo `clone()` per creare un nuovo oggetto con gli stessi contenuti di quello da clonare, che è l'istanza su cui viene invocato il metodo. Tutte le sottoclassi di `MidiMessage` descritte nel seguito ereditano i metodi `getLength()`, `getMessage()` e `getStatus()` e implementano una versione specializzata del metodo `clone()`.

Il metodo `setMessage()` presenta come argomenti un *array* di byte relativo al messaggio e una lunghezza espressa come intero. Si tratta però di un metodo `protected`, anch'esso ridefinito dalle sottoclassi che specializzano `MidiMessage`.

9.4.3 La classe ShortMessage

La classe `ShortMessage`, sottoclasse di `MidiMessage`, si rivolge alla rappresentazione di messaggi MIDI detti brevi, ossia contenenti al più due byte di dati. Essa copre tutte le famiglie di messaggi descritte nel Capitolo 2, a esclusione dei messaggi *System Exclusive* e dei meta-eventi, cui sono dedicate altre due sottoclassi specializzate di `MidiMessage`.

La classe prevede costanti numeriche intere per richiamare il valore dei byte di stato più comuni, quali `NOTE_ON` (il cui valore è 144 o 90_{16}), `PROGRAM_CHANGE` (192 o $C0_{16}$) e `ACTIVE_SENSING` (254 o FE_{16}). Si osservi che, nel caso di messaggi di canale, questi valori rappresentano il byte di stato al netto dell'informazione di canale, come pure i byte di stato per messaggi inviati sul Canale 1. L'informazione di canale può essere agevolmente integrata sommando al valore iniziale la rappresentazione numerica del canale (+1 per il Canale 2, +2 per il Canale 3 e via dicendo). Esistono anche varianti sintattiche per la creazione di messaggi brevi che prevedono un argomento aggiuntivo dedicato all'informazione di canale, disaccoppiando quindi il tipo di messaggio dal numero di canale.

La classe `ShortMessage` mette a disposizione il costruttore e il metodo `setMessage()` per impostare i messaggi brevi, entrambi dotati di varianti richiamabili in base alla *signature*. Si tratta della forma di polimorfismo detta *overloading*. Passando un unico parametro intero viene creato o impostato un messaggio con il solo byte di stato. L'utilizzo di tre argomenti interi specifica il byte di stato e i due byte di dati; questa variante può essere impiegata anche per i messaggi con un solo byte di dati, in cui il terzo valore – necessariamente presente nella chiamata per via dei vincoli sintattici – viene ignorato. Infine, la versione con quattro argomenti interi permette di specificare il comando (quindi il solo *nibble* di ordine alto del byte di stato), il canale (il *nibble* di ordine basso potenzialmente dedicato all'informazione di canale), il primo e il secondo byte di dati.

I metodi di `ShortMessage` includono, oltre ai già citati `clone()` e `setMessage()`, 4 funzioni per recuperare parti di informazione sul messaggio: `getChannel()`, `getCommand()`, `getData1()` e `getData2()`. Per i motivi sopra esposti, i valori dei byte vengono restituiti come interi.

9.4.4 La classe MetaMessage

Un'istanza della classe `MetaMessage`, sottoclasse di `MidiMessage`, rappresenta un meta-messaggio memorizzabile in un file MIDI e interpretabile da parte di un *sequencer*. L'argomento è stato trattato nel Paragrafo 5.4.

Il costruttore prevede solo due varianti: una senza argomenti, e una con tre argomenti, che rappresentano, rispettivamente, il tipo di meta-evento (intero minore di 128), l'*array* di byte del messaggio e la sua lunghezza in numero di byte. Quest'ultima, in generale, rappresenta esattamente la dimensione dell'*array*passato, il che renderebbe superfluo specificarla; esistono però casi particolari, che esulano dagli scopi del testo, in cui si rivela utile poter indicare una dimensione inferiore rispetto a quella dell'*array*. Invece, non è mai possibile specificare un numero di byte superiore.

Anche per questa specializzazione della classe `MidiMessage` è presente un metodo `setMessage()` che permette di definire o ridefinire i contenuti del messaggio. Esiste un'unica *signature* per tale metodo, analoga a quella a 3 argomenti del costruttore.

La classe implementa alcuni metodi specializzati: `clone()`, la cui funzione dovrebbe essere oramai nota al lettore; `getData()`, che restituisce una copia dei dati del meta-messaggio sotto forma di *array* di byte; `getType()`, che fornisce il tipo di meta-messaggio espresso come un valore intero.

9.4.5 La classe SysexMessage

La classe `SysexMessage`, sottoclasse di `MidiMessage`, si rivolge alla rappresentazione e gestione dei messaggi della famiglia *System Exclusive*. La *Java Sound API* codifica come messaggio esclusivo di sistema la sequenza data dal delimitatore di apertura SYSEX ($F0_{16}$), dal *payload* e dal delimitatore di chiusura EOX ($F7_{16}$).

Il costruttore presenta quattro varianti:

1. senza argomenti;
2. con un unico argomento, ossia l'*array* di byte che costituisce il *payload*. Questa variante del costruttore garantisce la costruzione di un messaggio midimsgSysEx valido;
3. con due argomenti, ossia l'*array* di byte e la sua dimensione in numero di byte;
4. con tre argomenti che specificano, rispettivamente, il byte di stato in formato intero, il *payload* come *array* di byte e la sua dimensione.

Il metodo specializzato `setMessage()` supporta solo le ultime due *signature* tra quelle previste per l'*overloading* del costruttore. Anche questa classe implementa `clone()`, la cui funzione è ormai nota al lettore. Infine, viene messo a disposizione del programmatore il metodo `getData()`, che restituisce sotto forma di *array* di byte il *payload* del messaggio SYSEX.

9.4.6 Esempi

In questa sezione verrà mostrato come creare messaggi MIDI, modificarli e recuperare informazioni da essi.

Si tratta del primo passaggio verso la realizzazione di una performance MIDI in Java, perché, per poterla finalizzare, i messaggi devono poi essere indirizzati su connessioni in uscita (MIDI OUT) a un sintetizzatore (performance estemporanea); in alternativa essi possono essere organizzati in eventi, gli eventi in tracce e le tracce in sequenze per alimentare un *sequencer* (performance temporizzata). Questi ulteriori passaggi verranno trattati nel seguito.

Creazione e gestione di messaggi MIDI brevi

Si vogliano creare algoritmicamente i messaggi di NOTE-ON e NOTE-OFF, relativi a una scala cromatica ascendente nell'ottava centrale, i cui pitch coprano dunque con continuità l'intervallo [60, 72]. Riguardo al numero di canale, si scelga di operare sul Canale 5, cui corrisponde il valore intero 4. Il Listato 9.7 mostra la generazione dei messaggi, la loro aggiunta a una struttura dati dinamica e un report finale per l'utente via *standard output*.

Si evidenzia che, nell'ottica di una performance MIDI, tale struttura dati andrebbe letta a determinati intervalli temporali, aspetto volutamente tralasciato dal presente esempio.

```
1  import java.util.ArrayList;
2  import javax.sound.midi.*;
3
4  public class ShortMessages {
5    private static String toString(byte[] array) {
6      String s = "[";
7      for (int i = 0; i < array.length; i++) {
8        if (i > 0)
9          s += ", ";
10       int intData = (int)(array[i] & 0xFF);
11       s += intData;
12     }
13     s += "]";
14     return s;
15   }
16
17   public static void main(String[] ar) {
18     ArrayList<ShortMessage> scale = new ArrayList();
19     for(int i = 60; i <= 72; i++) {
20       try {
21         ShortMessage msg1 = new ShortMessage(ShortMessage.NOTE_ON, 4, i, 64);
22         ShortMessage msg2 = new ShortMessage(ShortMessage.NOTE_OFF, 4, i, 0);
23         scale.add(msg1);
24         scale.add(msg2);
25       } catch (InvalidMidiDataException ex) {
26         System.out.println(ex);
27       }
28     }
29
30     for (ShortMessage msg : scale) {
31       System.out.println("Messaggio completo: " + toString(msg.getMessage()));
32       System.out.println("Lunghezza: " + msg.getLength());
33       System.out.println("Byte di stato: " + msg.getStatus());
34       System.out.println("    Comando: " + msg.getCommand());
35       System.out.println("    Canale: " + msg.getChannel());
36       System.out.println("Primo byte di dati: " + msg.getData1());
37       System.out.println("Secondo byte di dati: " + msg.getData2() + "\n");
38     }
39   }
40 }
```

Listato 9.7. Creazione e gestione di messaggi MIDI brevi.

All'interno del metodo `main()` si riconoscono due macro-blocchi. Il primo di questi (righe 18–28) è finalizzato a creare i messaggi NOTE-ON e NOTE-OFF, mentre il secondo (righe 30–38) mostra il contenuto della struttura dati esemplificando le molteplici possibilità offerte dai metodi della classe `ShortMessage`.

Si osservi che, al fine di rendere intelligibili le informazioni contenute nell'*array* di byte restituito da `getMessage()`, è stata implementata un'apposita funzione `toString()` (righe 5–15). A riga 10 è stato usato il già citato meccanismo di *cast* da byte a interi per evitare una rappresentazione del valore con segno. Si verifica agevolmente che `intData` assume il valore 132 quando `array[i]` vale -124 e il valore 148 quando `array[i]` vale -108, come tipico della rappresentazione in complemento a 2 su 8 bit.

Creazione e gestione di messaggi SysEx

Questo esempio mostra come costruire il messaggio MIDI per accendere la modalità XG su un sintetizzatore Yamaha compatibile con tale standard. A titolo di esempio, la seguente sequenza di byte è tratta dal manuale operativo del modello *Yamaha MU-80*:

```
F0 43 1n 4C 00 00 7E 00 F7
```

ove il byte $1n_{16}$ (ossia $0001nnnn_2$) denota l'identificativo del dispositivo all'interno del *setup*. Nel Listato 9.8 il byte viene arbitrariamente posto a 10_{16}.

```
1  import javax.sound.midi.*;
2
3  public class SysExMessages {
4    private static String toString(byte[] array) {
5      String s = "[";
6      for (int i = 0; i < array.length; i++) {
7        if (i > 0)
8        s += ", ";
9        int intData = (int)(array[i] & 0xFF);
10       s += intData;
11     }
12     s += "]";
13     return s;
14   }
15
16   public static void main(String[] ar) {
17     SysexMessage msg = new SysexMessage();
18     byte[] data = {0x43, 0x10, 0x4C, 0, 0, 0x7E, 0, (byte)0xF7};
19     try {
20       msg.setMessage(0xF0, data, data.length);
21       System.out.println("Messaggio completo: " + toString(msg.getMessage()));
22     } catch (InvalidMidiDataException ex) {
23       System.out.println(ex);
24     }
25   }
26 }
```

Listato 9.8. Creazione e gestione di messaggi SysEx.

Si sceglie, tra le molte varianti consentite, di creare un'istanza di `SysexMessage` vuota (riga 17), di definire l'*array* di byte del *payload* includendo il messaggio EOX (riga 18) e, infine, di assegnare al messaggio la sequenza di byte risultante (riga 20).

I valori costanti inseriti nell'array di byte alla riga 18 vengono considerati di default di tipo intero. Il *cast* del valore $F7_{16}$ si rende necessario per non incorrere in un errore "Incompatible types: possible lossy conversion from int to byte". Questo problema non si pone per i restanti valori, in quanto essi sfruttano al più 7 degli 8 bit e la conversione da intero a byte non introduce criticità.

Invio di messaggi MIDI a un sintetizzatore

Il Listato 9.9 genera, al pari del Listato 9.7, una scala cromatica ascendente con pitch nell'intervallo [60, 72], ma, in aggiunta, invia i corrispettivi messaggi di Note-On e Note-Off al sintetizzatore di default.

```
1  import javax.sound.midi.*;
2
3  public class SendMessages {
4    public static void main(String[] ar) throws MidiUnavailableException,
         InvalidMidiDataException, InterruptedException {
5      ShortMessage myMsg = new ShortMessage();
6      Synthesizer synth = MidiSystem.getSynthesizer();
7      synth.open();
8      Receiver rcvr = synth.getReceiver();
9      myMsg.setMessage(ShortMessage.PROGRAM_CHANGE, 0, 16, 0);
10     rcvr.send(myMsg, -1);
11     for (int i = 0; i < 13; i++) {
12       myMsg.setMessage(ShortMessage.NOTE_ON, 0, 60 + i, 96);
13       rcvr.send(myMsg, 500000 * i);
14       myMsg.setMessage(ShortMessage.NOTE_OFF, 0, 60 + i, 0);
```

```
15        rcvr.send(myMsg, 500000 * (i + 1));
16      }
17      Thread.sleep(13 * 500 + 1);
18      synth.close();
19    }
20  }
```

Listato 9.9. Invio di messaggi MIDI a un sintetizzatore.

Si rende necessario, innanzitutto, individuare il sintetizzatore di default (riga 6), aprirlo (riga 7) e recuperare l'oggetto `Receiver` cui inviare i messaggi (riga 8).

Alla riga 10 si mostra l'invocazione di una `send()` immediata, il cui *timestamp* viene appositamente impostato a -1; le successive chiamate presentano invece un *timestamp* non negativo espresso in microsecondi, finalizzato a disporre gli eventi nel tempo distanziando i NOTE-ON e i NOTE-OFF corrispondenti di 0,5 s.

Trattandosi di un'applicazione su console, è necessario impiegare un artificio per evitare che il sintetizzatore venga immediatamente chiuso dall'invocazione di `close()` o che le risorse vengano liberate dal raggiungimento della fine del codice. Nel listato si è scelto di interrompere l'esecuzione per un numero opportuno di millisecondi (riga 17); in alternativa si sarebbe potuto richiedere un input da tastiera all'utente.

Ricezione e visualizzazione di messaggi MIDI

Il Listato 9.10 mostra un semplice esempio di *MIDI monitor*, limitando, per questioni di brevità, la gestione dei messaggi a quelli principali.

```
1  import javax.sound.midi.*;
2  import java.util.List;
3
4  public class ReceiveMessages {
5    public static void main(String args[]) {
6      MidiDevice device = null;
7      MidiDevice.Info[] infos = MidiSystem.getMidiDeviceInfo();
8      for (int i = 0; i < infos.length; i++) {
9        try {
10         device = MidiSystem.getMidiDevice(infos[i]);
11         device.getTransmitter();
12         List<Transmitter> transmitters = device.getTransmitters();
13         for(int j = 0; j < transmitters.size();j++)
14           transmitters.get(j).setReceiver(new MidiInputReceiver());
15         device.open();
16         System.out.println("Dispositivo " + device.getDeviceInfo().getName() + " aperto");
17       } catch (MidiUnavailableException e) {
18         if (device != null)
19           System.out.println("Dispositivo " + device.getDeviceInfo().getName() + " non
                 aperto");
20       }
21     }
22   }
23 }
24
25 class MidiInputReceiver implements Receiver {
26   private static String toString(byte[] array) {
27     String s = "[";
28     for (int i = 0; i < array.length; i++) {
29       if (i > 0)
30         s += ", ";
31       int intData = (int)(array[i] & 0xFF);
32       s += intData;
33     }
34     s += "]";
35     return s;
36   }
37
38   @Override
39     public void send(MidiMessage msg, long timeStamp) {
40     if (msg instanceof ShortMessage) {
41       ShortMessage shortMsg = (ShortMessage)msg;
42       switch(shortMsg.getCommand()) {
```

```
43          case ShortMessage.NOTE_ON:
44            System.out.println("Note-On, canale " + shortMsg.getChannel());
45            System.out.println("Pitch: " + shortMsg.getData1());
46            System.out.println("Velocity: " + shortMsg.getData2() + "\n");
47            break;
48          case ShortMessage.NOTE_OFF:
49            System.out.println("Note-Off, canale " + shortMsg.getChannel());
50            System.out.println("Pitch: " + shortMsg.getData1());
51            System.out.println("Velocity: " + shortMsg.getData2() + "\n");
52            break;
53          case ShortMessage.PROGRAM_CHANGE:
54            System.out.println("Program Change, canale " + shortMsg.getChannel());
55            System.out.println("Numero di programma: " + (shortMsg.getData1() + 1) + "\n");
56            break;
57          default:
58            System.out.println("Messaggio non gestito");
59            System.out.println(toString(shortMsg.getMessage()) + "\n");
60        }
61      } else if (msg != null) {
62        System.out.println("Messaggio non gestito");
63        System.out.println(toString(msg.getMessage()) + "\n");
64      }
65    }
66
67    @Override
68    public void close() {}
69  }
```

Listato 9.10. Ricezione e visualizzazione di messaggi MIDI.

Il listato si compone di due blocchi: la classe principale, chiamata `ReceiveMessages` (righe 4–23), e una classe di appoggio, detta `MidiInputReceiver` (righe 25–69).

All'interno del ciclo *for* (righe 8–21) si cerca di recuperare il trasmettitore di default del dispositivo; nel caso non fosse disponibile, la chiamata solleverebbe un'eccezione gestita dal blocco *catch*. A ciascun trasmettitore del dispositivo si associa dunque una nuova istanza di `MidiInputReceiver`.

La classe `MidiInputReceiver` implementa l'interfaccia `Receiver`, il che richiede l'*override* dei metodi `send()` e `close()`. Ai fini dell'esempio, riveste interesse solo il metodo `send()`, che verifica innanzitutto se il messaggio sia istanza di `ShortMessage` o, più genericamente, di `MidiMessage`; nel primo caso la funzione cerca di presentare le informazioni in modo più leggibile, mentre, nel secondo, si limita a mostrare la sequenza di byte. Il numero di casi gestiti dal costrutto *switch-case* (righe 42–60) può essere facilmente ampliato.

In assenza di una catena MIDI fisica collegata in ingresso al computer, è possibile testare il codice installando un *controller* software (ad esempio una tastiera) e una porta MIDI virtuali, quindi collegando il MIDI OUT del *controller* alla porta virtuale e lasciando che l'applicativo in questione rilevi i messaggi provenienti dalla porta virtuale. Quest'ultima verrà riconosciuta tra i dispositivi dotati di trasmettitori e, di conseguenza, le verrà automaticamente associata un'istanza di `MidiInputReceiver`.

9.5 La classe MidiEvent

Gli **eventi MIDI** si compongono di un messaggio MIDI, nell'accezione estesa vista nel Paragrafo 9.4, e di un *timestamp* espresso in *ticks*. Il concetto di evento assume significato per l'informazione salvata in un file MIDI o all'interno di una sequenza; in una performance estemporanea, invece, sarebbe assente il concetto di temporizzazione, ossia la componente di *timestamp* associata al messaggio per generare l'evento nella sua interezza.

Il *timestamp* agisce come una distanza rispetto all'inizio della performance, istante caratterizzato da un numero di *ticks* pari a 0. La durata dei *ticks* non ha un valore assoluto, ma va rapportata all'informazione contenuta nell'*header* di un file MIDI o in un oggetto della classe `Sequence`. Si osservi che i *timestamp* degli eventi MIDI differiscono dai Δt degli eventi

negli Standard MIDI Files: questi ultimi costituiscono distanze relative agli eventi immediatamente precedenti anziché fare tutti riferimento al punto iniziale.

Gli oggetti `MidiEvent` possono essere organizzati in tracce (istanze della classe `Track`), che, a loro volta, possono formare sequenze (istanze della classe `Sequence`), che, infine, possono essere associate a un *sequencer* (istanza della classe Sequencer). Le nuove classi menzionate verranno trattate nel seguito del capitolo. La Figura 9.2 mostra la gerarchia completa, da intendersi non in termini di ereditarietà e specializzazione delle classi propria di Java, bensì come strutturazione logica nella rappresentazione dell'informazione.

9.5.1 Metodi principali

Il costruttore presenta un'unica *signature* che prevede due argomenti: i) il messaggio MIDI, istanza di `MidiMessage`, e ii) il *timestamp*, espresso come numero di *ticks* attraverso un valore *long*.

I metodi implementati dalla classe sono: `getMessage()`, che enuclea il solo messaggio MIDI contenuto nell'evento; `getTick()`, che ne restituisce il *timestamp*; `setTick()`, che permette di ridefinire il *timestamp* in termini di numero di *ticks*.

9.5.2 Esempio

Nel Listato 9.11 viene mostrata la creazione di una scala esatonale ascendente che copre un'ottava, eseguita da un timbro di flauto e le cui note durano 40 *ticks* e sono distanziate tra loro da silenzi di 8 *ticks*. Si tratta di un'evoluzione dell'esempio sulla creazione di messaggi brevi (Listato 9.7).

```
1  import java.util.ArrayList;
2  import javax.sound.midi.*;
3
4  public class MidiEvents {
5    public static void main(String[] ar) throws InvalidMidiDataException {
6      ArrayList<MidiEvent> scale = new ArrayList();
7      ShortMessage msg0 = new ShortMessage(ShortMessage.PROGRAM_CHANGE, 73, 0);
8      MidiEvent ev0 = new MidiEvent(msg0, 0);
9      scale.add(ev0);
10     for(int i = 0; i <= 6; i++) {
11       int pitch = 60 + i * 2;
12       ShortMessage msg1 = new ShortMessage(ShortMessage.NOTE_ON, pitch, 64);
13       long timestamp1 = i * 48;
14       MidiEvent ev1 = new MidiEvent(msg1, timestamp1);
15       ShortMessage msg2 = new ShortMessage(ShortMessage.NOTE_OFF, pitch, 0);
16       long timestamp2 = timestamp1 + 40;
17       MidiEvent ev2 = new MidiEvent(msg2, timestamp2);
18       scale.add(ev1);
19       scale.add(ev2);
20     }
21
22     long lastTimestamp = 0;
```

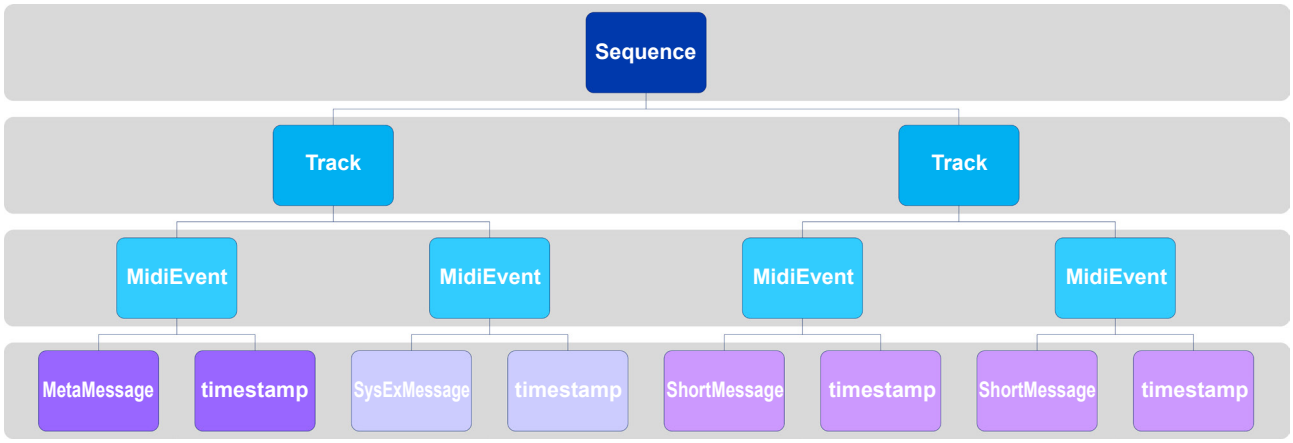


Figura 9.2. Strutturazione dell'informazione relativa a sequenze, tracce, eventi e messaggi.

```
23      MidiMessage currentMsg = new ShortMessage();
24      for (MidiEvent ev : scale) {
25        long delta = ev.getTick() - lastTimestamp;
26        System.out.println("Delta-time: " + delta);
27        currentMsg = ev.getMessage();
28        String msgDescription = "";
29        switch (currentMsg.getStatus()) {
30          case ShortMessage.NOTE_ON: msgDescription = "Note-On"; break;
31          case ShortMessage.NOTE_OFF: msgDescription = "Note-Off"; break;
32          case ShortMessage.PROGRAM_CHANGE: msgDescription = "Program Change"; break;
33          default: msgDescription = "messaggio sconosciuto";
34        }
35        System.out.println("Byte di stato: " + currentMsg.getStatus() + " (" +
              msgDescription + ")");
36        int dataByte = currentMsg.getMessage()[1];
37        System.out.println("Primo byte di dati: " + dataByte + "\n");
38        lastTimestamp = ev.getTick();
39      }
40    }
41  }
```

Listato 9.11. Creazione di eventi MIDI.

Rispetto al Listato 9.7, ora la struttura dati dinamica `scale` è una collezione di oggetti `MidiEvent`. Per migliorare la leggibilità del codice si è preferito procedere gradualmente nella creazione di un'istanza di `ShortMessage` e di `MidiEvent` e nell'aggiunta alla struttura dati, ma tali passaggi sarebbero riassumibili in una sola riga di codice. Le righe 7–9 potrebbero essere sostituite, ad esempio, da

```
scale.add(new MidiEvent(new ShortMessage(ShortMessage.PROGRAM_CHANGE,73,0),0));
```

Si osservi che, decidendo di sfruttare il primo canale, è possibile utilizzare, per il costruttore di `ShortMessage`, la sintassi a 3 argomenti e costanti intere per richiamare mnemonicamente i byte di stato (righe 7, 12 e 15).

Dalla riga 22 in avanti si costruisce il report da visualizzare su *standard output*. Viene implementato un semplice meccanismo per ricondurre i *timestamp* (relativi a un istante iniziale) a Δt(relativi all'evento precedente), storicizzando, di iterazione in iterazione, il più recente valore di *ticks* incontrato (righe 25 e 38).

Si noti che in questo esempio viene validamente impiegata la classe più generica `MidiMessage` (righe 23 e 27) in luogo della sua specializzazione `ShortMessage`, anche se tale scelta impedisce di utilizzare un metodo specializzato quale `getData1()` a riga 36.

9.6 La classe Track

Una **traccia MIDI** è un flusso indipendente di eventi MIDI (messaggi MIDI dotati di *timestamp*). Una traccia può essere salvata con altre tracce all'interno di uno Standard MIDI File. Si ricorda al lettore l'ortogonalità tra i concetti di traccia e di canale MIDI. Anche se le specifiche del protocollo MIDI 1.0 consentono solo 16 canali, le tracce possono trovarsi in qualunque numero all'interno di un file MIDI; inoltre ciascuna traccia, presa singolarmente, può contenere informazione riconducibile a un solo canale, a più canali o a nessun canale MIDI.

All'interno della gerarchia che lega, in ultima analisi, un *sequencer* ai dati MIDI (Figura 9.2), una traccia si trova in posizione intermedia: un *sequencer* suona sequenze, le sequenze contengono tracce, le tracce a loro volta sono costituite da eventi MIDI e gli eventi sono coppie di dati MIDI e relativi *timestamp*.

La classe `Track` mette a disposizione metodi per gestire i contenuti di una traccia aggiungendo o rimuovendo oggetti della classe `MidiEvent` o delle sue specializzazioni. Tali operazioni mantengono la lista degli eventi in un corretto ordine temporale. Altri metodi servono a recuperare la dimensione della traccia in termini di numero di eventi o di durata in *ticks*.

Esiste un certo numero di operazioni sulla traccia che vengono delegate ad altre classi. Le funzioni di *mute* o *solo*, ad esempio, vanno richiamate da parte del *sequencer* (istanza della classe

`Sequencer`) che gestisce la sequenza (istanza della classe `Sequence`) contenente la traccia. Le informazioni generali di temporizzazione e risoluzione temporale, che devono accomunare tutte le tracce all'interno di una sequenza, sono contenute nella sequenza cui afferisce la traccia. Per via di tale dipendenza della traccia rispetto alla sequenza, una traccia non può essere creata invocando direttamente il costruttore della classe `Track`; è necessario richiamare il metodo `createTrack()` della classe `Sequence`.

9.6.1 Metodi principali

I principali metodi messi a disposizione sono `add()` e `remove()`, rispettivamente finalizzati ad aggiungere alla traccia e rimuovere dalla traccia un evento. A tali funzioni va passata come unico argomento un'istanza di `MidiEvent`. Nel primo caso, l'evento viene aggiunto alla traccia in corrispondenza dell'istante temporale specificato dal suo *timestamp*. Un evento MIDI, ossia una coppia {dati MIDI, *timestamp*} già presente nella traccia, non viene nuovamente aggiunto. Riguardo al metodo `remove()`, esso fallisce nel caso gli venga passato un evento non presente nella traccia.

Il metodo `get()` restituisce un oggetto `MidiEvent` corrispondente al un dato indice intero da passare come argomento.

Vanno citate, infine, le funzioni che recuperano la lunghezza della traccia in termini di numero di eventi e di *ticks*. Si tratta, rispettivamente, di `size()`, che produce in uscita un valore intero, e di `ticks()`, che restituisce un valore *long*.

9.6.2 Esempio

Nel Listato 9.12 viene generata una sequenza di messaggi NOTE-ON e NOTE-OFF con altezze pseudo-casuali nell'intervallo [60, 72] e *timestamp* incrementali. Tale sequenza viene assegnata a una traccia inizialmente vuota, cui viene poi applicato un filtro per eliminare i messaggi riferiti a un qualsiasi Do naturale (o equivalenti scritture enarmoniche). Il codice mostra su *standard output* i contenuti della traccia prima e dopo l'operazione di filtraggio.

```
1  import java.util.Random;
2  import javax.sound.midi.*;
3
4  public class GenerateFilterTrack {
5    public static void main(String[] ar) throws InvalidMidiDataException {
6      Random rand = new Random();
7      int upperbound = 13; // valori nell'intervallo [0, 12]
8      int ppqn = 48;
9      long currentTimestamp = 0;
10     Sequence s = new Sequence(Sequence.PPQ, ppqn);
11     Track t = s.createTrack();
12     for (int i = 0; i < 40; i++) {
13       int randomPitch = 60 + rand.nextInt(upperbound);
14       ShortMessage msg1 = new ShortMessage(ShortMessage.NOTE_ON, randomPitch, 64);
15       ShortMessage msg2 = new ShortMessage(ShortMessage.NOTE_OFF, randomPitch, 0);
16       MidiEvent ev1 = new MidiEvent(msg1, currentTimestamp);
17       currentTimestamp += ppqn * (rand.nextInt(2) + 1);
18       MidiEvent ev2 = new MidiEvent(msg1, currentTimestamp);
19       currentTimestamp += ppqn * (rand.nextInt(2) + 1);
20       t.add(ev1);
21       t.add(ev2);
22     }
23     System.out.println("TRACCIA ORIGINALE");
24     System.out.println("Durata in ticks: " + t.ticks());
25     System.out.println("Numero di eventi: " + t.size());
26
27     for (int i = t.size() - 1; i >= 0; i--) {
28       MidiEvent currentEvent = t.get(i);
29       MidiMessage currentMessage = currentEvent.getMessage();
30       if ((currentMessage.getStatus() == ShortMessage.NOTE_ON || currentMessage.getStatus
             () == ShortMessage.NOTE_OFF) && currentMessage.getMessage()[1] % 12 == 0)
31         t.remove(currentEvent);
32     }
33     System.out.println("-----");
```

```
34        System.out.println("TRACCIA FILTRATA");
35        System.out.println("Durata in ticks: " + t.ticks());
36        System.out.println("Numero di eventi: " + t.size());
37      }
38  }
```

Listato 9.12. Inserimento e rimozione di eventi MIDI da una traccia.

Si rimanda alla documentazione della classe `Random` riguardo alla generazione di valori pseudo-casuali (righe 6–7, 13, 17 e 19).

A riga 10 viene istanziato un oggetto di classe `Sequence`; anche se questa verrà descritta solo nel prossimo paragrafo, l'anticipazione si rende necessaria per poter creare una traccia attraverso il metodo `createTrack()` (riga 11).

Nella traccia originaria vengono inseriti 40 messaggi NOTE-ON e altrettanti NOTE-OFF. Potrebbe sorprendere, dunque, la presenza di 81 messaggi nel report finale. Il motivo è la presenza di un evento di fine traccia implicitamente aggiunto alla creazione della traccia stessa.

Alla riga 29 si istanzia un generico oggetto `MidiMessage` in luogo di un più specifico `ShortMessage`, il che – a sua volta – costringe a richiamare il primo byte di dati di `currentMessage` con la sintassi `getMessage()[1]` invece di `getData1()` (riga 30). Potrebbe sorprendere il fallimento di un eventuale *cast* a `ShortMessage` di messaggi che, per costruzione, sono NOTE-ON o NOTE-OFF; tuttavia, come detto, la traccia contiene l'evento di fine traccia che non rientra nella casistica degli `ShortMessage`. Anche se non strettamente necessaria, essendo l'evento di fine traccia costituito dai byte FF $2F_2$, l'aggiunta delle prime due condizioni in *or* alla riga 30 limita il controllo della condizione sul pitch ai soli messaggi di NOTE-ON e NOTE-OFF.

Il comportamento del codice è pseudo-casuale, quindi non è possibile affermare con certezza che la traccia filtrata differisca da quella originaria. Una riduzione nel numero di eventi è comunque statisticamente probabile, essendo legata alla presenza di almeno un Do naturale tra i 40 pitch sorteggiati. La durata della traccia in *ticks*, invece, non cambia, in quanto l'evento di fine traccia non viene rimosso e dunque mantiene il proprio *timestamp*. Se pure tale evento venisse rimosso, per modificare la durata della traccia in *ticks* l'ultimo messaggio di canale dovrebbe essere un NOTE-OFF relativo a un Do naturale.

9.7 La classe Sequence

Una **sequenza** è una struttura dati che contiene informazione musicale pensata per essere eseguita da un *sequencer*. Il contenuto di una sequenza abbraccia non solo un elenco di tracce, ma anche l'informazione di temporizzazione che le accomuna. Grazie a quest'ultima, la marcatura temporale degli eventi si traduce in unità di tempo assolute. L'informazione di temporizzazione include il tipo di suddivisione dell'unità di tempo e, in caso di approccio *pulse per quarter*, determina il numero di *ticks* per pulsazione.

Una sequenza può essere creata partendo da un oggetto vuoto e associando a esso una o più oggetti `Track`, oppure leggendo un file MIDI e invocando una delle varianti di `getSequence()` messe a disposizione da `MidiSystem`. Nell'esempio di questo paragrafo ci si concentrerà sul primo scenario, mentre nel Paragrafo 9.8 verrà affrontato il secondo.

9.7.1 Metodi principali

Il costruttore della classe presenta due *signature*. La prima variante ha 2 argomenti, uno di tipo *float* per definire la tipologia di suddivisione (`divisionType`), e un altro di tipo intero per specificare la risoluzione temporale (`resolution`). La seconda variante, a 3 argomenti, aggiunge alla *signature* precedente un ulteriore parametro di tipo intero per determinare il numero di tracce nella sequenza.

Valori ammissibili per `divisionType` sono le seguenti costanti *float* definite dalla classe stessa: `PPQ` per una temporizzazione basata sulle pulsazioni, dipendente dal tempo metronomico, e

`SMPTE_24`, `SMPTE_25`, `SMPTE_30` e `SMPTE_30DROP` per una temporizzazione assoluta di tipo SMPTE, rispettivamente, a 24, 25, 30 e 29,97 frame al secondo. Il significato del secondo argomento, `resolution`, varia a seconda dei casi: se il tipo di suddivisione è `PPQ`, si tratta del numero di *ticks* per pulsazione; negli altri casi il valore rappresenta il numero di *ticks* per frame.

Oltre ai metodi per creare ed eliminare tracce, `createTrack()` e `deleteTrack()`, la classe permette di recuperare il tipo di suddivisione con `getDivisionType()`, la risoluzione temporale tramite `getResolution()`, la durata della sequenza in microsecondi attraverso `getMicrosecondLength()` e la sua durata in *ticks* con `getTickLength()`.

Il metodo `getTracks()` ottiene l'*array* di tracce (oggetti `Track`) che costituiscono la sequenza. Similmente, il metodo `getPatchList()` restituisce l'*array* di *patch* (oggetti `Patch`) all'interno della sequenza.

9.7.2 Esempio

Il Listato 9.13 mostra su *standard output* gli eventi MIDI contenuti in una sequenza e disposti su più tracce, riordinati per *timestamp* crescente. L'esempio potrebbe essere considerato una sorta di *MIDI monitor*, ma, anziché agire in tempo reale, esso effettua una scansione del contenuto già temporizzato di una sequenza.

```
1  import java.util.ArrayList;
2  import java.util.TreeMap;
3  import javax.sound.midi.*;
4
5  public class GenerateSequence {
6    private static String toString(byte[] array) {
7      String s = "[";
8      for (int i = 0; i < array.length; i++) {
9        if (i > 0)
10         s += ", ";
11       int intData = (int)(array[i] & 0xFF);
12       s += intData;
13     }
14     s += "]";
15     return s;
16   }
17
18   public static void main(String[] ar) throws InvalidMidiDataException {
19     int ppqn = 48;
20     Sequence s = new Sequence(Sequence.PPQ, ppqn, 2);
21     s.createTrack();
22     for (int i = 0; i < s.getTracks().length; i++) {
23       for (int j = 0; j < 4; j++) {
24         int pitch = 48 + 12 * i + j;
25         int duration = ppqn / (2 * (i + 1));
26         ShortMessage smOn = new ShortMessage(ShortMessage.NOTE_ON + i, pitch, 90);
27         ShortMessage smOff = new ShortMessage(ShortMessage.NOTE_OFF + i, pitch, 0);
28         MidiEvent evOn = new MidiEvent(smOn, j * duration);
29         MidiEvent evOff = new MidiEvent(smOff, j * duration + duration); // (j + 1) *
               duration
30         s.getTracks()[i].add(evOn);
31         s.getTracks()[i].add(evOff);
32       }
33     }
34
35     TreeMap<Long, ArrayList<MidiMessage>> map = new TreeMap();
36     for (Track t: s.getTracks()) {
37       for (int i = 0; i < t.size(); i++) {
38         long currentTime = t.get(i).getTick();
39         MidiMessage currentMsg = t.get(i).getMessage();
40         if (!map.containsKey(currentTime)) {
41           ArrayList<MidiMessage> msgList = new ArrayList<>();
42           msgList.add(currentMsg);
43           map.put(currentTime, msgList);
44         } else {
45           map.get(currentTime).add(currentMsg);
46         }
47       }
48     }
```

```
49
50      for (long timestamp : map.keySet()) {
51        System.out.println("Timestamp: " + timestamp);
52        for (MidiMessage mm : map.get(timestamp)) {
53          System.out.println(toString(mm.getMessage()));
54        }
55        System.out.println();
56      }
57    }
58  }
```

Listato 9.13. Aggiunta e rimozione di eventi MIDI da una traccia.

Tralasciando la funzione `toString()`, già adottata nel Listato 9.7, il codice è ripartito in tre blocchi: le righe 19–33 creano la sequenza, generano le tracce e popolano queste ultime con messaggi MIDI; le righe 35–48 predispongono una struttura dati adeguata per riordinare i messaggi MIDI in base al *timestamp*; le righe 50–56, infine, sono dedicate all'output.

Alla riga 20 si mostra l'uso del costruttore di `Sequence` nella variante che specifica il numero di tracce. Le tracce vengono quindi contestualmente create e, all'inizio, risultano prive di messaggi associati. A riga 21 si crea un'ulteriore traccia vuota attraverso la sintassi `createTrack()`. Il successivo doppio ciclo annidato inizializza il contenuto delle 3 tracce usando pitch che si collocano su ottave differenti (riga 24) e durate in *PPQN* differenti (riga 25). La prima traccia dispone, per costruzione, messaggi ogni $48/(2\times(0+1)) = 24$ *ticks*, la seconda ogni $48/(2\times(1+1)) = 12$ *ticks*, e la terza ogni $48/(2\times(2+1)) = 8$ *ticks*. Gli eventi vengono aggiunti alla traccia corrente alle righe 30–31. Le temporizzazioni e le durate sono state calcolate algoritmicamente, anziché generate randomicamente, per offrire un maggior controllo sul buon esito dello script.

Il secondo blocco si focalizza sulla struttura dati più adeguata agli scopi dell'esempio. Si è scelta una `TreeMap`, ossia una collezione dinamica di coppie {chiave, valore} che mantiene l'ordinamento per chiave. Si è pensato di sfruttare la chiave per la rappresentazione dei *timestamp* e di inserire come valore un'ulteriore struttura dati dinamica (`ArrayList`), pensata per contenere tutti i messaggi (`MidiMessage`) che hanno luogo a quel dato *timestamp*. Il doppio ciclo che ha luogo alle righe 36–48 ha lo scopo di passare in rassegna – traccia per traccia – ogni evento, e di collocarlo nella mappa in corrispondenza del *timestamp* opportuno. La mappa potrebbe già contenere la chiave, caso in cui sarebbe sufficiente aggiungere il messaggio alla lista corrispondente (riga 45), oppure non contemplare ancora il valore di *timestamp* tra le sue chiavi, il che richiederebbe di creare la lista, associarla al *timestamp* e aggiungere la coppia alla mappa (righe 40–44).

L'apparente macchinosità del secondo blocco rende estremamente agevole il compito del terzo: è sufficiente iterare sulle chiavi e, per ciascuno dei *timestamp* così recuperati, iterare sui messaggi MIDI corrispondenti. Una visualizzazione più leggibile dell'*array* di byte è affidata alla chiamata del metodo `toString()`.

9.8 L'interfaccia Sequencer

L'interfaccia `Sequencer` presenta metodi per effettuare le normali operazioni di un *sequencer* MIDI, tra cui ottenere una sequenza da un file MIDI, avviare e interrompere la sua esecuzione, riposizionarsi in un punto arbitrario della sequenza, modificare il tempo (ossia la velocità di riproduzione), sincronizzare la performance rispetto a un *clock* interno o ai messaggi MIDI ricevuti e controllare la temporizzazione di un altro dispositivo.

In modo diretto o indiretto, ossia attraverso oggetti cui hanno accesso gli oggetti dell'interfaccia `Sequencer`, è possibile inoltre aggiungere o cancellare singoli eventi MIDI o tracce intere, applicare effetti di *mute* e *solo* sulle singole tracce, interloquire con oggetti *listener* in merito a meta-eventi o CONTROL CHANGE incontrati durante l'esecuzione.

L'interfaccia `Sequencer` gestisce oggetti di classe `Sequence`, quindi interagisce con il livello più alto della gerarchia mostrata nella Figura 9.2.

9.8.1 Metodi principali

L'interfaccia `Sequencer` eredita dall'interfaccia `MidiDevice` i metodi `open()` e `close()`, necessari per allocare risorse di sistema all'apertura del dispositivo e per liberarle quando non sono più necessarie.

Il metodo `setSequence()` assegna a un'istanza di `Sequencer` la sequenza passata come argomento, oggetto dell'interfaccia `Sequence` (Paragrafo 9.7).

La performance viene avviata dal principio, o ripresa dal punto corrente, attraverso l'invocazione del metodo `start()`. Essa viene interrotta, ma non riavvolta, dal metodo `stop()`; un'ulteriore invocazione di `start()`, dunque, riprenderebbe dall'istante temporale corrente. La posizione di esecuzione della sequenza può essere recuperata o impostata, tanto in *ticks*, quanto in microsecondi, attraverso metodi che operano con valori di tipo *long*. Si tratta, in particolare, di `getTickPosition()`, `getMicrosecondPosition()`, `setTickPosition()` e `setMicrosecondPosition()`.

Il tempo metronomico del brano può essere impostato secondo varie strategie. Il metodo `setTempoFactor()` consente di specificare un valore di tipo *float* come fattore moltiplicativo rispetto al tempo attuale. Il metodo `setTempoInBPM()` permette di impostare il metronomo attraverso un valore di tipo *float*. Un'ulteriore possibilità è offerta dal metodo `setTempoInMPQ()`, cui viene passato il numero di microsecondi al quarto. Esistono metodi corrispondenti per recuperare tali valori: `getTempoFactor()`, `getTempoInBPM()`, `getTempoInMPQ()`. La durata complessiva della traccia è ottenibile, in sola lettura, attraverso il metodo `getMicrosecondLength()`; il suo valore può variare durante l'esecuzione in risposta a modifiche sul metronomo.

È possibile realizzare *loop*, ossia ripetizioni cicliche nella lettura degli eventi MIDI, grazie ai metodi `setLoopStartPoint()` e `setLoopEndPoint()`. A essi va passato un valore di tipo *long*, al fine di identificare il punto iniziale o finale tramite un numero di *tick*. Attraverso il metodo `setLoopCount()` è possibile determinare il numero di *loop* da eseguire complessivamente. L'interfaccia presenta anche i corrispettivi metodi `getLoopStartPoint()`, `getLoopEndPoint()` e `getLoopCount()`.

In un dato istante si può trovare, al più, una sequenza associata al *sequencer*, la quale, a sua volta, dà informazioni e accesso alle rispettive tracce. Lo stato di *mute* e *solo* sono proprietà che caratterizzano lo stato della singola traccia, tuttavia per accedervi è necessario utilizzare metodi della classe `Seqeuncer`. Innanzitutto è necessario recuperare l'indice numerico intero che identifica la traccia all'interno della collezione di tracce proprie della sequenza. In seguito è possibile invocare i metodi `getTrackMute()` e `getTrackSolo()` propri dell'interfaccia `Sequencer`, che richiedono come unico parametro tale identificativo numerico. I corrispettivi metodi per impostare lo stato di *mute* e *solo*, `setTrackMute()` e `setTrackSolo()`, in aggiunta all'indice intero della traccia, richiedono un valore boolean per abilitare o disabilitare la funzionalità.

9.8.2 Esempi

Apertura ed esecuzione di un file MIDI

Lo *snippet* di codice riportato nel Listato 9.14 mostra come aprire un file MIDI da disco, avviarne e interromperne l'esecuzione.

```
1  import java.io.*;
2  import javax.sound.midi.*;
3  import java.util.Scanner;
4
5  public class OpenMidiFile {
6    public static void main(String[] ar) {
7      try {
8        File midiFile = new File("midisong.mid");
9        Sequence mySequence = MidiSystem.getSequence(midiFile);
10       Sequencer mySequencer = MidiSystem.getSequencer();
11       mySequencer.open();
12       mySequencer.setSequence(mySequence);
13       mySequencer.start();
```

```
14          System.out.println("Premere il tasto INVIO interrompere l'esecuzione...");
15          Scanner scanner = new Scanner(System.in);
16          scanner.nextLine();
17          mySequencer.stop();
18          mySequencer.close();
19        } catch(MidiUnavailableException | InvalidMidiDataException | IOException e) {
20          System.out.println(e);
21        }
22      }
23  }
```

Listato 9.14. Apertura ed esecuzione di un file MIDI.

Alla riga 9 viene caricato un file MIDI utilizzando un percorso relativo che punta alla cartella individuata da `System.getProperty("user.dir")`[7]. In alternativa, è possibile usare percorsi assoluti che presenteranno una sintassi dipendente dal sistema operativo in uso.

Alla riga 10 la sequenza contenuta nel file MIDI viene assegnata a un oggetto `mySequence` di classe `Sequence`.

Si rende poi necessario creare un oggetto di classe `Sequencer`, chiamato `mySequencer`, ottenibile a partire da `MidiSystem` attraverso l'invocazione del metodo `getSequencer()` (riga 10).

Le successive operazioni consistono nell'apertura del *sequencer*(riga 11), nell'associazione a esso della sequenza `mySequence` (riga 12) e, infine, nell'avvio della performance (riga 13), che ha luogo al lancio del programma; l'interruzione dell'esecuzione è affidata alla naturale conclusione della sequenza oppure alla pressione del tasto "Invio" su console, che invoca esplicitamente il metodo `mySequencer.stop()`. In un programma dotato di interfaccia grafica sarebbe più corretto associare le funzioni di avvio e interruzione dell'esecuzione a pulsanti o controlli simili.

Si osservi che il metodo `stop()` non comporta un riavvolgimento della sequenza, richiamando in realtà la funzione di pausa, e il metodo `start()` riavvia la riproduzione dal punto corrente.

Assegnare una seconda sequenza al *sequencer* non comporta un missaggio degli eventi già presenti con quelli di nuova introduzione, bensì la sostituzione della sequenza corrente. Se il *sequencer* si trovasse in stato di *play* all'atto della sostituzione della sequenza, la performance generale non verrebbe interrotta, ma procederebbe con i primi eventi contenuti nelle nuove tracce.

Recupero e modifica del tempo metronomico

Lo *snippet* di codice riportato nel Listato 9.15 recupera le informazioni sulla durata della *song* e sul suo tempo metronomico ed esemplifica varie strategie per modificare quest'ultimo durante la performance.

```
1   import java.io.*;
2   import java.util.Scanner;
3   import javax.sound.midi.*;
4
5   public class ReadChangeTempo {
6     public static void main(String[] ar)
7     {
8       try {
9         File midiFile = new File("midisong.mid");
10        Sequence mySequence = MidiSystem.getSequence(midiFile);
11        Sequencer mySequencer = MidiSystem.getSequencer();
12        mySequencer.open();
13        mySequencer.setSequence(mySequence);
14        System.out.println("Durata originale: " + (mySequencer.getMicrosecondLength() /
              1000000) + "s");
15        float originalBPM =  mySequencer.getTempoInBPM();
16        System.out.println("Tempo (bpm): " + originalBPM);
17        mySequencer.start();
18        System.out.println("Premere il tasto INVIO per dimezzare il BPM...");
19        Scanner scanner = new Scanner(System.in);
20        scanner.nextLine();
21        mySequencer.setTempoInBPM(originalBPM / 2);
```

[7] In caso di dubbio, è possibile aggiungere al codice la riga `System.out.println(System.getProperty("user.dir"));` che farà apparire il nome della cartella su *standard output* testuale.

```
22        System.out.println("Tempo (bpm): " + mySequencer.getTempoInBPM());
23        System.out.println("Premere il tasto INVIO per triplicare la velocita' di
              riproduzione...");
24        scanner.nextLine();
25        mySequencer.setTempoFactor(3);
26        System.out.println("Premere il tasto INVIO per interrompere...");
27        scanner.nextLine();
28        mySequencer.stop();
29        mySequencer.close();
30      }
31      catch(MidiUnavailableException | InvalidMidiDataException | IOException e){
32        System.out.println(e);
33      }
34    }
35  }
```

Listato 9.15. Apertura ed esecuzione di un file MIDI.

Riga 14 visualizza sullo *standard output* la durata del file MIDI in secondi e riga 15 restituisce il tempo metronomico originale.

Alla prima pressione del tasto "Invio", il tempo viene dimezzato attraverso un metodo che agisce sul bpm (riga 21). La successiva pressione triplica il valore corrente di metronomo tramite l'applicazione di un fattore correttivo (riga 25).

Elenco delle tracce, *mute* e *solo*

Il codice riportato nel Listato 9.16 mostra come un *sequencer* possa interagire con le varie tracce di cui si compone una sequenza caricata da file MIDI.

```
1   import java.io.*;
2   import javax.sound.midi.*;
3
4   public class TracksReport {
5     public static void main(String[] ar) {
6       try {
7         File midiFile = new File("midisong.mid");
8         Sequence mySequence = MidiSystem.getSequence(midiFile);
9         Track[] trackArray = mySequence.getTracks();
10        for (int i = 0; i < trackArray.length; i++) {
11          if (i > 0)
12          System.out.println("-----");
13          System.out.println("TRACCIA " + i);
14          System.out.println("Lunghezza in ticks: " + trackArray[i].ticks());
15          System.out.println("Numero di eventi contenuti: " + trackArray[i].size());
16        }
17
18        System.out.println("---------------\n");
19
20        Sequencer mySequencer = MidiSystem.getSequencer();
21        mySequencer.open();
22        mySequencer.setSequence(mySequence);
23        Track t1 = mySequence.createTrack();
24        Track t2 = mySequence.createTrack();
25        ShortMessage msg1 = new ShortMessage(ShortMessage.NOTE_ON, 60, 64);
26        MidiEvent event1 = new MidiEvent(msg1, 0);
27        t2.add(event1);
28        ShortMessage msg2 = new ShortMessage(ShortMessage.NOTE_OFF, 60, 0);
29        MidiEvent event2 = new MidiEvent(msg2, 96);
30        t2.add(event2);
31        mySequencer.setTrackMute(1, true);
32        Track[] newTrackArray = mySequencer.getSequence().getTracks();
33        for (int i = 0; i < newTrackArray.length; i++) {
34          if (i > 0)
35          System.out.println("-----");
36          System.out.println("TRACCIA " + i);
37          System.out.println("Lunghezza in ticks: " + newTrackArray[i].ticks());
38          System.out.println("Numero di eventi contenuti: " + newTrackArray[i].size());
39          boolean isMuted = mySequencer.getTrackMute(i);
40          boolean isSolo = mySequencer.getTrackSolo(i);
41          System.out.println("Mute: " + isMuted);
42          System.out.println("Solo: " + isSolo);
43        }
```

```
44          mySequencer.close();
45        } catch(InvalidMidiDataException | MidiUnavailableException | IOException ex){
46          System.out.println(ex);
47        }
48      }
49  }
```

Listato 9.16. Elenco delle tracce di una sequenza.

Il corpo della funzione `main()` si compone di due parti. La prima di queste (righe 7–16) carica la sequenza contenuta nel file "midisong.mid" e scrive sullo *standard output* testuale un report sulle tracce ivi contenute. Si osservi che gli oggetti `Track` sono accessibili a partire da un oggetto `Sequence` prima ancora di associare quest'ultimo a un oggetto `Sequencer` attraverso `setSequence()`, il che avrà luogo solo a riga 22.

La seconda parte del codice (righe 20–44) crea il *sequencer*, gli associa la sequenza precedentemente caricata, aggiunge due tracce alla sequenza, pone la seconda traccia (a questo punto sicuramente esistente) in *mute* e, infine, stampa su *standard output* testuale un nuovo report delle tracce, corredato dall'informazione sullo stato di *mute* e *solo*. Si sottolinea che l'operazione di *muting* è delegata a un metodo della classe `Sequencer` (riga 31).

Riguardo alla creazione e al popolamento delle tracce, si nota che la prima di queste non ha eventi esplicitamente assegnati, motivo per cui il rapporto restituisce lunghezza in *ticks* pari a zero. Viene segnalata, tuttavia, la presenza di un evento; si tratta dell'evento di fine traccia, aggiunto di default all'invocazione di `createTrack()`. Le righe 25–30 mostrano la creazione di due istanze di `ShortMessage`, la loro trasformazione in `MidiEvent`, posizionati, rispettivamente, al *tick* 0 e al *tick* 96, e la loro aggiunta alla seconda delle nuove tracce. Il rapporto segnala, coerentemente, la presenza di tre eventi e la durata complessiva di 96 *ticks*.

Capitolo 10

MATLAB: Audio Toolbox

Il supporto di MATLAB al protocollo MIDI si basa principalmente su un modulo software aggiuntivo chiamato *Audio Toolbox*. Questo *add-on*, come indica il nome stesso, ha una valenza molto più generale, fornendo anche strumenti per l'elaborazione dell'audio, la generazione diretta della forma d'onda, l'analisi del tratto vocale, la misurazione in ambito acustico, la spazializzazione del suono e via dicendo. Il *toolbox* presenta, inoltre, interfacce di streaming per schede audio mediante protocolli ASIO, WASAPI, ALSA e CoreAudio e mette a disposizione strumenti per la generazione e l'*hosting* di plug-in VST e Audio Units.

Di particolare interesse per le finalità del presente libro è il supporto offerto al protocollo MIDI in termini di collegamento con dispositivi esterni e di ricezione, creazione, manipolazione e trasmissione dei messaggi.

I vantaggi di porre in comunicazione l'ambito MIDI con un software per il calcolo numerico quale MATLAB sono molteplici. Si spazia dall'analisi di sequenze di valori provenienti da una performance MIDI attraverso gli strumenti statistici di MATLAB alla possibilità di generare forme d'onda complesse e rappresentarle all'interno di messaggi SysEx da inoltrare ai dispositivi MIDI a valle.

Nel corso del capitolo verranno illustrate le funzioni per elencare i dispositivi MIDI presenti nel sistema (Paragrafo 10.1), per creare messaggi MIDI e gestirne la temporizzazione (Paragrafo 10.2), per riceverli da e inviarli a un *setup* esterno a MATLAB (Paragrafi 10.3 e 10.4), presentando inoltre alcuni casi esemplari corredati da codice (10.5). Infine, verrà menzionato il *MIDI Toolbox*, un modulo software per MATLAB finalizzato alla musicologia computazionale, che supporta funzioni di gestione degli Standard MIDI Files e metodi avanzati di visualizzazione dell'informazione MIDI (Paragrafo 10.6).

Tutti i listati introdotti e commentati nel seguito del capitolo sono a disposizione del lettore all'URL https://ludovico.lim.di.unimi.it/download/libri/midi/matlab.zip.

10.1 Elenco dei dispositivi MIDI

Il primo passaggio per mettere in comunicazione MATLAB con l'ecosistema MIDI è rilevare l'elenco dei dispositivi disponibili nel sistema. Questa operazione iniziale viene realizzata invocando il comando `mididevinfo`. La specifica risposta è chiaramente dipendente dalle caratteristiche del sistema in uso. Un esempio può essere il seguente:

```
MIDI devices available:
ID  Direction  Interface   Name
0   output     MMSystem   'Microsoft MIDI Mapper'
1    input     MMSystem   'LoopBe Internal MIDI'
2   output     MMSystem   'VirtualMIDISynth #1'
3   output     MMSystem   'Microsoft GS Wavetable Synth'
4   output     MMSystem   'LoopBe Internal MIDI'
```

Dal punto di vista di MATLAB, i dispositivi elencati rappresentano ingressi (input) o uscite (output). Un oggetto[1] `mididevice` può quindi essere utilizzato, rispettivamente, per ascoltare messaggi in arrivo o per inviare messaggi in uscita. Esiste anche la possibilità che uno stesso dispositivo svolga entrambe le funzioni, come nel caso sopra riportato di "LoopBe Internal MIDI". In questa evenienza vengono assegnati all'oggetto `mididevice` due codici identificativi diversi.

Ora è possibile creare un oggetto `mididevice`, arbitrariamente chiamato `device`, specificandone o il nome o l'identificativo attraverso una delle seguenti sintassi alternative:

```
device = mididevice('VirtualMIDISynth #1')
device = mididevice(2)
```

il cui riscontro nella *command window*, in caso di successo, risulta

```
mididevice connected to
Output: 'VirtualMIDISynth #1' (2)
```

Quando la chiamata di `mididevice()` avviene passando il nome anziché l'identificativo e il dispositivo coinvolto presenta funzioni sia di input sia di output, l'oggetto `mididevice` risultante dispone tanto di ingressi quanto di uscite. In riferimento al precedente esempio, il comando

```
device = mididevice('LoopBe Internal MIDI')
```

genera il seguente riscontro nella console:

```
mididevice connected to
Input: 'LoopBe Internal MIDI' (1)
Output: 'LoopBe Internal MIDI' (4)
```

10.2 Creazione e rappresentazione dei messaggi

In MATLAB i messaggi MIDI sono impacchettati come oggetti `midimsg` e possono essere manipolati, secondo le consuetudini dell'ambiente, come valori scalari, vettori o matrici. L'elenco completo delle sintassi supportate per la creazione e inizializzazione di oggetti di tipo `midimsg`, arbitrariamente chiamati `msg`, è il seguente:

```
msg = midimsg('Note',channel,note,velocity,duration,timestamp)
msg = midimsg('NoteOn',channel,note,velocity,timestamp)
msg = midimsg('NoteOff',channel,note,velocity,timestamp)
msg = midimsg('ControlChange',channel,ccnumber,ccvalue,timestamp)
msg = midimsg('ProgramChange',channel,program,timestamp)
msg = midimsg('SystemExclusive',bytes,timestamp)
msg = midimsg('SystemExclusive',timestamp)
msg = midimsg('Data',bytes,timestamp)
msg = midimsg('EOX',timestamp)
msg = midimsg('TimingClock',timestamp)
msg = midimsg('Start',timestamp)
msg = midimsg('Continue',timestamp)
msg = midimsg('Stop',timestamp)
msg = midimsg('ActiveSensing',timestamp)
```

[1] Qui e nel seguito verrà utilizzato il termine "oggetto" per coerenza con la documentazione dell'*Audio Toolbox*. Tuttavia è opportuno precisare che si tratti di un tipo di dato strutturato comunemente detto "struct" in ambito informatico.

```
msg = midimsg('SystemReset',timestamp)
msg = midimsg('TuneRequest',timestamp)
msg = midimsg('MIDITimeCodeQuarterFrame',seq,value,timestamp)
msg = midimsg('SongPositionPointer',position,timestamp)
msg = midimsg('SongSelect',song,timestamp)
msg = midimsg('AllSoundOff',channel,timestamp)
msg = midimsg('ResetAllControllers',channel,timestamp)
msg = midimsg('LocalControl',channel,localcontrol,timestamp)
msg = midimsg('PolyOn',channel,timestamp)
msg = midimsg('MonoOn',channel,monoChannels,timestamp)
msg = midimsg('OmniOn',channel,timestamp)
msg = midimsg('OmniOff',channel,timestamp)
msg = midimsg('AllNotesOff',channel,timestamp)
msg = midimsg('PolyKeyPressure',channel,note,pressure,timestamp)
msg = midimsg('ChannelPressure',channel,pressure,timestamp)
msg = midimsg('PitchBend',channel,change,timestamp)
msg = midimsg
msg = midimsg(size)
msg = midimsg(0)
```

La maggior parte dei parametri fa riferimento alla sintassi dei messaggi MIDI presentata nel Capitolo 2 e, a questo punto della trattazione, dovrebbe risultare autoesplicativa. Al momento si suggerisce al lettore di trascurare il parametro `timestamp`, ripreso nel seguito e comunque opzionale per le chiamate che lo contemplano.

In tutte le sintassi, a esclusione delle ultime tre, il primo parametro specifica il tipo di messaggio MIDI. Tale valore può essere espresso tramite un vettore di caratteri, una stringa o un membro dell'enumerazione `midimsgtype`. Ad esempio, le seguenti sintassi sono tra loro alternative e assegnano a `msg` lo stesso messaggio MIDI:

```
msg = midimsg('NoteOn',1,60,96)
msg = midimsg("NoteOn",1,60,96)
msg = midimsg(midimsgtype.NoteOn,1,60,96)
```

Un oggetto `midimsg` si presenta come una struttura dati con un insieme predefinito di campi, detti elementi o proprietà. Alcuni di questi campi risultano sempre compilati; è il caso del tipo (`Type`), del numero di byte che costituiscono il messaggio (`NumMsgBytes`), del vettore di tali byte (`MsgBytes`) e della marcatura temporale (`Timestamp`). Altre proprietà sono sempre e comunque presenti, ma – a seconda dello specifico messaggio – possono essere valorizzate o meno. Ad esempio, il valore di `Note` e `Velocity` ha significato per messaggi NOTE-ON e NOTE-OFF, ma non per PROGRAM CHANGE o CHANNEL PRESSURE. Analogamente, il valore del campo `Channel` non viene compilato per messaggi che non siano di canale.

Una volta creato un oggetto `midimsg`, la notazione puntata (*dot notation*) consente l'accesso in lettura alle sue proprietà. Riguardo l'accesso scrittura, è possibile reimpostare alcune proprietà, ma altre risultano di sola lettura. Ad esempio, il numero di programma associato a un PROGRAM CHANGE può essere modificato tramite accesso in scrittura alla proprietà `Program`, ma un messaggio NOTE-ON non può essere trasformato in un CHANNEL PRESSURE in quanto `Type` e `MsgBytes` sono di sola lettura.

10.2.1 Messaggi non di tipo MIDI

Alcuni dei comandi sopra descritti non trovano immediata corrispondenza nei messaggi MIDI trattati nel Capitolo 2.

La prima delle sintassi elencate si riferisce al tipo `Note` e riassume in sè i messaggi MIDI NOTE-ON e NOTE-OFF, distanziandoli sulla base di `duration`. L'istante temporale a cui si verifica il NOTE-OFF corrisponde alla temporizzazione di NOTE-ON più il valore di `duration` espresso in secondi. Tecnicamente la struttura dati `msg` valorizzata da questa sintassi è un vettore costituito da due elementi, ciascuno contenente un oggetto `midimsg`. Seguendo le convenzioni di *naming* delle variabili MATLAB sarebbe più opportuno attribuire a essa un nome al plurale quale, ad esempio, `msgs` in luogo di `msg`.

Tornando all'elenco sopra riportato, si nota che le ultime tre sintassi non specificano alcun tipo di messaggio, creando dunque oggetti `midimsg` non immediatamente riconducibili a funzioni MIDI. La chiamata

```
msg = midimsg
```

restituisce un `midimsg` il cui `Type` vale 0, ossia è di tipo `Data`. L'oggetto contiene 8 byte posti a 0 e nessun'altra proprietà impostata.

La sintassi

```
msg = midimsg(size)
```

restituisce una matrice quadrata di `midimsg` con tutti i byte posti a 0. L'argomento in ingresso, ossia `size`, si rifà alla sintassi MATLAB comunemente in uso per specificare le dimensioni di una matrice; può trattarsi di uno scalare, andando quindi a determinare una matrice quadrata con numero di righe e colonne fissato a `size`, di un vettore, che consente di differenziare il numero di righe e colonne, o anche di una generica matrice, che crea strutture dati multidimensionali. Ad esempio, la sintassi

```
messages = midimsg(3)
```

crea una matrice 3×3 di oggetti `midimsg`, mentre

```
messages = midimsg([2,3,4])
```

crea una matrice tridimensionale $2 \times 3 \times 4$ generando in tutto 24 oggetti `midimsg`.

Sebbene si tratti di matrici, la visualizzazione dei loro elementi viene resa lineare e l'accesso può avvenire sia tramite una coppia di indici, sia attraverso uno scalare. In una matrice 2×2, ad esempio, le due scritture

```
msg(1,2).Timestamp = 5
```

e

```
msg(3).Timestamp = 5
```

sono alternative equivalenti per accedere in scrittura alla proprietà `Timestamp` dello stesso elemento.

Infine, la sintassi

```
msg = midimsg(0),
```

può essere considerata un caso particolare del comando precedente in cui l'argomento `size` è posto a 0, creando dunque una matrice di 0×0 oggetti `midimsg` vuoti.

10.2.2 Temporizzazione dei messaggi

Il protocollo MIDI non definisce aspetti di temporizzazione dei messaggi, assumendo che questi agiscano istantaneamente nel contesto di una performance estemporanea. Molte applicazioni, invece, si basano su informazione temporale per accodare e processare i messaggi.

L'*Audio Toolbox* consente di conferire una temporizzazione agli oggetti `midimsg`, tanto all'atto della loro creazione, quanto a posteriori, ma in quest'ultimo caso l'effetto è efficace solo se precede l'istante di invio del messaggio.

Nelle varianti di `midimsg()` sopra elencate si nota la frequente presenza dell'argomento `timestamp`. Questo parametro può sempre essere omesso, e, in tal caso, viene considerato come posto a 0. I valori temporali vengono espressi in secondi e fanno riferimento a un istante 0 la cui determinazione da parte del sistema dipende da come i messaggi MIDI vengono creati e utilizzati.

Nel caso di ricezione dei messaggi, dal punto di vista concettuale, il *timing clock* viene attivato nell'istante in cui un oggetto `mididevice` viene creato e connesso attraverso un *listener* a una data porta d'ingresso. Se un'altra istanza di `mididevice` viene collegata allo stesso input, essa si vede assegnare marcature temporali riferite allo stesso *timing clock*.

L'aspetto complementare concerne l'invio di messaggi MIDI. In questo caso, i *timestamp* consentono l'invio di messaggi differito nel tempo. Il meccanismo di marcatura temporale dei messaggi in uscita fa riferimento a una stessa porta di output; temporizzazioni e distanze vengono dunque calcolati porta per porta.

Un messaggio con marcatura temporale posta a 0 o non espressa implica un invio immediato. Una marcatura temporale positiva esprime, in generale, una deviazione temporale in secondi rispetto al *timestamp* più alto specificato nell'invio immediatamente precedente. Si osservi che l'invio precedente potrebbe a sua volta contenere uno o più *timestamp* non nulli di cui è necessario tenere conto.

Si è parlato finora di invii multipli di messaggi singoli dotati di marcatura temporale, ma – come si mostrerà nel Paragrafo 10.4 – è anche possibile invocare una singola operazione di invio su un vettore di `midimsg`, ciascuno potenzialmente dotato di un *timestamp* differente. In tal caso, tutte le marcature temporali "interne" si relazionano a un unico riferimento, anziché essere interpretate in modo incrementale. Per il momento ci si limiterà a esemplificare i diversi scenari per via grafica (Figura 10.1), ma l'argomento verrà ripreso nel Paragrafo 10.4.

Rispetto al criterio generale ora esposto fa eccezione lo scenario in cui il primo invio presenti un *timestamp* non nullo. In questo caso, il riferimento è dato dall'orologio di sistema (*real-world time*).

10.3 Ricezione dei messaggi

La ricezione dei messaggi MIDI è legata a una funzione degli oggetti `mididevice` chiamata `midireceive()`. Quando si crea un oggetto `mididevice` dotato di ingressi, questo inizia a ricevere dati dal proprio input e a scriverli all'interno di un buffer. L'invocazione di `midireceive()` scarica il contenuto del buffer in una variabile che contiene un vettore di oggetti `midimsg`.

A titolo di esempio, dato un dispositivo in ingresso chiamato "Virtual MIDI port" e dotato di input, la riga

```
mydev = mididevice('Virtual MIDI port')
```

crea un oggetto `mididevice` chiamato `mydev`. Ora l'oggetto è pronto per ricevere messaggi in ingresso, che vengono memorizzati in un buffer. Si ipotizzi la pressione e il rilascio di un tasto in un *controller* collegato alla porta. L'invocazione di

```
receivedMessages = midireceive(mydev)
```

comporta la creazione e l'inizializzazione di un buffer di oggetti `midimsg`. Questa operazione svuota il buffer, che però rimane pronto per ricevere nuovi messaggi in ingresso. Il contenuto di `receivedMessages`, dal punto di vista logico, è dato da un messaggio NOTE-ON con un determinato *timestamp* e un messaggio NOTE-OFF con timestamp superiore; dal punto di vista implementativo potrebbe invece trattarsi di un messaggio NOTE-ON con velocity non nulla e un ulteriore messaggio NOTE-ON con velocity nulla, ma questo dipende esclusivamente dalla strategia adottata nella generazione dei messaggi da parte del *controller*.

10.4 Invio dei messaggi

L'invio di messaggi MIDI ha luogo invocando la funzione `midisend()` degli oggetti `mididevice`. La sintassi è la seguente:

```
midisend(device,msg)
```

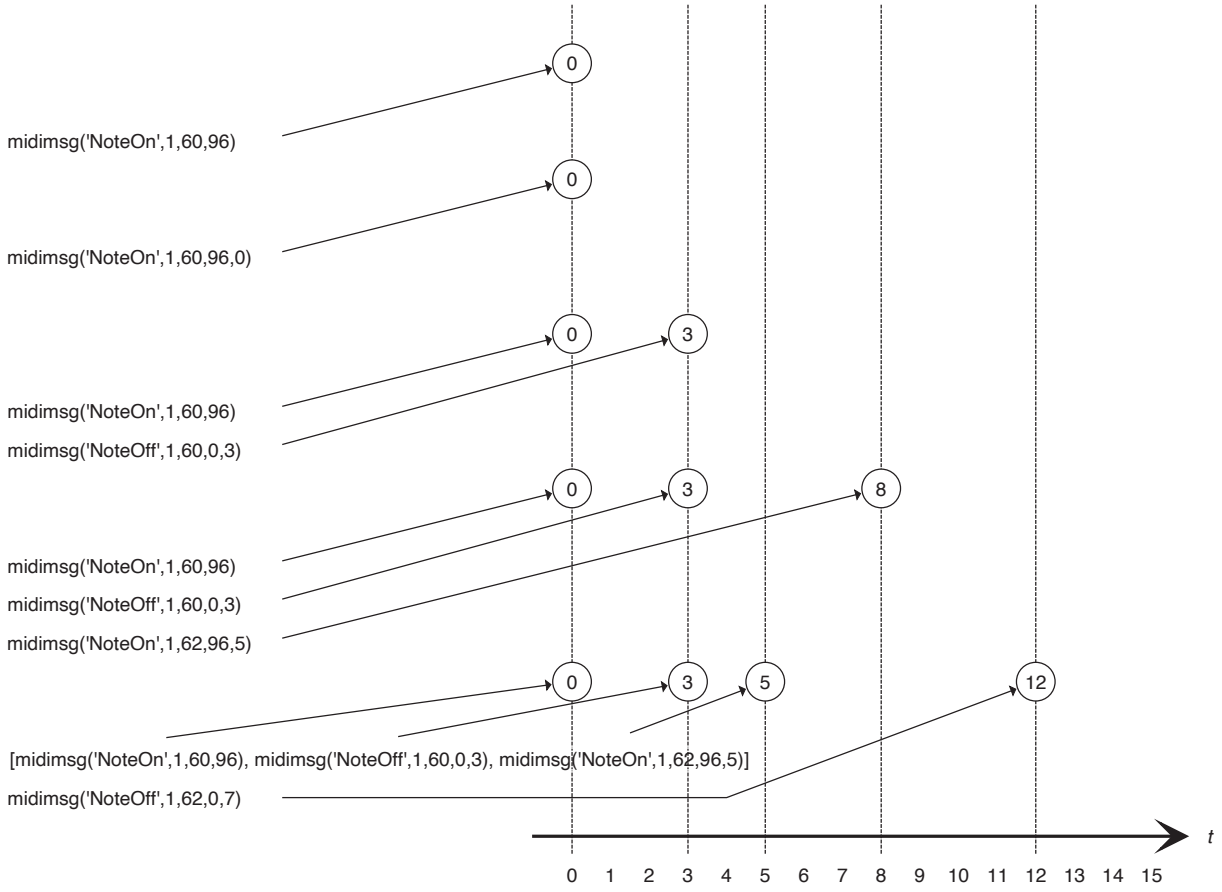


Figura 10.1. Distanze temporali in secondi introdotte dall'argomento `timestamp`. Ogni riga corrisponde all'invio del messaggio, o dei messaggi, ivi contenuti.

ove `device` indica un dispositivo MIDI dotato di uscite e `msg` rappresenta un singolo oggetto o un insieme di oggetti `midimsg`. Come per la ricezione dei messaggi, gli eventi vengono accumulati in un buffer di output per poi essere instradati, all'istante opportuno, sulla catena a valle.

Si voglia accendere e spegnere una nota con il timbro di organo da chiesa della durata di 2,5 s in un sintetizzatore virtuale il cui indice è 2 nell'elenco dei dispositivi MIDI installati nel sistema. Il Listato 10.1 mostra una prima versione, estremamente semplice da comprendere, del codice che svolge tali operazioni. Il suo funzionamento è chiaramente subordinato all'esistenza di un dispositivo di uscita all'indice 2.

```
1  device = mididevice(2);
2  programChangeMsg = midimsg('ProgramChange',1,20);
3  midisend(device,programChangeMsg);
4  onMsg = midimsg('NoteOn',1,69,64);
5  offMsg = midimsg('NoteOn',1,69,0);
6  offMsg.Timestamp = 2.5;
7  midisend(device,onMsg);
8  midisend(device,offMsg);
```

Listato 10.1. Codice per l'invio di messaggi MIDI.

Si noti la determinazione esplicita di un *timestamp* per il solo oggetto `offMsg` (riga 6), il che sarebbe potuto avvenire già alla riga 5 usando la sintassi a 5 parametri. L'attribuzione di un *timestamp* non nullo all'oggetto `onMsg` avrebbe comportato per la nota un'accensione differita nel tempo, ma non ne avrebbe modificato la durata.

Un aspetto ottimizzabile del codice è la duplice invocazione di `midisend()`, il che comporta due svuotamenti del buffer. Invece, come tipico di MATLAB, la funzione `midisend()` supporta anche l'invio di più messaggi tramite un'unica invocazione, utilizzando come secondo argomento un vettore di oggetti `midimsg`. Il Listato 10.2 mostra una versione alternativa.

```
1  device = mididevice(2);
2  msgs = [midimsg('ProgramChange',1,20); midimsg('NoteOn',1,69,64); midimsg('NoteOn',1,69,0)
       ];
3  msgs(3).Timestamp = 2.5;
4  midisend(device,msgs);
```

Listato 10.2. Codice alternativo per l'invio di messaggi MIDI.

Nell'ipotesi di un generico vettore di oggetti `midimsg` inviati da un'unica chiamata di `midisend()`, l'effettiva simultaneità degli eventi MIDI dipende dalle informazioni di *timestamp* per ciascuno di essi, e – a parità di posizionamento temporale – subentra il comportamento tipico di un protocollo di comunicazione seriale.

Esiste la possibilità di adottare una sintassi ancora più compatta, come mostrato nel Listato 10.3.

```
1  device = mididevice(2);
2  msgs = [midimsg('ProgramChange',1,20); midimsg('Note',1,69,64,2.5)];
3  midisend(device,msgs);
```

Listato 10.3. Codice alternativo per l'invio di messaggi MIDI.

Al fine di approfondire la questione delle marcature temporali relative, descritta nel Paragrafo 10.2.2, si consideri il Listato 10.4. Nel codice vengono creati due eventi NOTE-ON e due eventi NOTE-OFF, ciascuno dotato di una propria collocazione temporale, ma vengono seguite strade differenti nel loro invio tramite `midisend()`.

```
1  device = mididevice(2);
2  onMsg1 = midimsg('NoteOn',1,60,90);
3  offMsg1 = midimsg('NoteOff',1,60,0,1);
4  onMsg2 = midimsg('NoteOn',1,61,90,1);
5  offMsg2 = midimsg('NoteOff',1,61,0,2);
6  midisend(device,onMsg1);
7  midisend(device,offMsg1);
8  midisend(device,onMsg2);
9  midisend(device,offMsg2);
10 midisend(device,[onMsg1,offMsg1,onMsg2,offMsg2]);
```

Listato 10.4. Approfondimento sull'invio di messaggi con *timestamp*.

Le righe 6–9 costituiscono quattro invii differenti. Di conseguenza, lo spegnimento della prima nota segue di 1 s la sua accensione, l'accensione della seconda nota è differita di 1 s rispetto all'evento precedente (generando così una pausa) e, infine, lo spegnimento della seconda nota ha luogo dopo 2 s (ottenendo un suono di durata doppia rispetto al primo).

A riga 10 ha luogo un unico invio "simultaneo" dei quattro messaggi, pertanto il riferimento per i *timestamp* è unico. In altri termini, la seconda nota viene accesa nell'istante di spegnimento della prima e i due eventi sonori presentano uguale durata.

10.5 Esempi

In questa sezione vengono presentati alcuni casi esemplari di applicabilità dell'*Audio Toolbox* alla comunicazione MIDI. La premessa è che l'ecosistema MATLAB offre principalmente strumenti per il calcolo numerico, l'analisi statistica e l'elaborazione del segnale, quindi l'uso di MIDI in MATLAB è normalmente giustificato da esigenze legate a tali ambiti.

La scrittura di codice MATLAB si riconduce a due contesti applicativi ben distinti: le cosiddette *applicazioni stand-alone*, che sono in grado di girare autonomamente in ambiente MATLAB, e i *plug-in*, che richiedono un *host* per poter funzionare.

Nel primo caso è compito del programmatore gestire i dispositivi e le operazioni di ricezione e invio dei messaggi, avendo cura di richiamare tali funzioni nel momento desiderato. Sarebbe possibile, ad esempio, adottare un approccio iterativo invocando le funzioni in un *loop* che esegua *polling* sul dispositivo di interesse a intervalli di tempo regolari.

Nel caso di sviluppo di *plug-in*, invece, è compito dell'*host* quello di gestire dispositivi, input e output, mentre il *plug-in* dovrebbe operare sui messaggi MIDI scambiati con l'host, tipicamente in forma di vettori. Questo approccio rimane, però, a livello puramente teorico, in quanto l'*Audio Toolbox*, al momento, limita la comunicazione tra *host* e *plug-in* al solo audio e allo scambio di parametri. Gli esempi sotto riportati rientreranno, dunque, nel primo scenario.

10.5.1 Estrazione delle pitch class

Il concetto di *pitch class*, o classe di altezze, è proprio dell'approccio insiemistico nel contesto dell'analisi musicale computazionale. Per definizione, uno stesso valore di *pitch class* caratterizza l'insieme di tutte le altezze poste a un numero intero di ottave di distanza, indipendentemente dalla notazione musicale che le origina. Ad esempio, la classe di altezze del *do* naturale consta di tutti i *do* naturali in tutte le ottave e raggruppa in sè anche le note derivanti dalla scrittura in partitura di *si*♯ e *re*𝄫. Nel sistema musicale occidentale, in cui è considerato atomico l'intervallo di semitono, l'ottava è divisa in 12 parti che corrispondono a 12 *pitch class*. Per ulteriori dettagli, si rimanda il lettore a testi dedicati quali [13] e [32].

Sebbene non strettamente necessario, è consuetudine associare il valore di *pitch class* 0 all'altezza del *do* naturale, 1 a *do*♯ e via dicendo. Ogni altezza viene dunque ricondotta a un valore nell'intervallo [0, 11].

Partendo dal pitch MIDI è particolarmente agevole estrarre il valore di *pitch class*: si tratta del resto della divisione intera tra il pitch stesso e 12, operazione che MATLAB implementa attraverso la funzione `rem()`.

Il codice riportato nel Listato 10.5 consente di contare le *pitch class* associate agli eventi che giungono al dispositivo di ingresso `device` tra l'avvio dello script e la pressione di un tasto sulla tastiera del computer. Al termine dell'esecuzione viene prodotto un grafico a barre.

```
1  device = mididevice(1);
2  disp('Receiving notes from input port. Press any key to stop and draw the diagram...')
3  pause()
4  receivedMessages = midireceive(device);
5  types = [receivedMessages.Type];
6  pitches = [receivedMessages(types==1).Note];
7  receivedPitchClasses = rem(pitches,12);
8  receivedPitchClasses = categorical(receivedPitchClasses);
```

```
allPitchClasses = categorical(0:11);
histogram(receivedPitchClasses,allPitchClasses);
set(gca,'XTickLabel',allPitchClasses);
title('Histogram of pitch classes');
xlabel('Pitch classes');
ylabel('Count');
grid on;
```

Listato 10.5. Generazione dell'istogramma dei valori di pitch class.

Si osservi, alle righe 5 e 6, il processo di filtraggio operato sui messaggi in ingresso per tenere traccia solo di quelli di tipo 1, ossia NOTE-ON. Ne deriva un vettore costituito dai soli pitch degli eventi dei NOTE-ON, sui cui elementi viene operato a riga 7 il calcolo della *pitch class*.

Le righe 10–15 si occupano di realizzare una rappresentazione grafica a istogramma dei valori ottenuti.

10.5.2 Sonificazione di dati meteorologici

Un'altra possibile applicazione che coinvolge MATLAB e MIDI è la sonificazione di dati numerici, ossia la trasformazione del loro valore e delle loro relazioni in segnali acustici ai fini di facilitarne la comunicazione o l'interpretazione [3, 17, 20]. Quando la sonificazione è rivolta alla performance musicale, si parla più propriamente di musificazione [8, 15, 36].

Il punto di partenza è dato da una o più serie di valori disponibili all'interno dell'ambiente in forma matriciale. A monte dell'intervento di sonificazione, MATLAB può analizzare i dati da vari punti di vista, manipolarli a seconda delle esigenze applicative o generarne di nuovi in modo agevole. Una volta ottenuta la serie di dati desiderata, la sonificazione dell'informazione avviene associando i valori numerici a opportuni parametri musicali.

L'esempio sotto riportato trae spunto da un documento in formato CSV costituito da 9 colonne e 31 righe, contenente le rilevazioni giornaliere della temperatura minima, della temperatura massima e della percentuale di umidità nell'aria acquisite da tre diverse stazioni (Milano Linate, Firenze Peretola e Palermo Punta Raisi) nel mese di dicembre 2020. Tali dati sono disponibili all'interno di un file chiamato "open_data.csv". La prima, la quarta e la settima colonna del CSV corrispondono a 3 serie di temperature minime (una per ciascuna città), la seconda, la quinta e l'ottava colonna a 3 serie di temperature massime e, infine, la terza, la sesta e la nona colonna alle percentuali di umidità. Le righe corrispondono ai 31 giorni delle rilevazioni.

Si sceglie in modo arbitrario di associare le 3 stazioni di rilevamento a 3 timbri facilmente riconoscibili: pianoforte acustico, violoncello e flauto, i cui *program number* sono, rispettivamente, 1, 43 e 74.

Le temperature massime e minime vengono rappresentate dalle altezze delle note (pitch), opportunamente scalate in modo da sfruttare al meglio l'intervallo [0, 127] ed evitare di uscirne. Trattandosi del periodo invernale, un'associazione diretta tra temperature e pitch collocherebbe la performance in un registro molto basso, spesso al di sotto del limite inferiore di udibilità, e introdurrebbe, talvolta, valori negativi non accettabili come pitch MIDI. Per risolvere il problema si sceglie di far corrispondere il valore 0°C al pitch 60. Per migliorare l'intelligibilità dei risultati, i pitch del violoncello vengono trasposti un'ottava sotto (0°C = 48), mentre quelli del flauto un'ottava sopra (0°C = 72).

La percentuale di umidità, per definizione compresa tra 0 e 100, può riflettersi senza ulteriori modifiche nella velocity delle note. La presenza di valori nulli andrebbe opportunamente gestita, ad esempio sommando 1 a tutti i valori. Si osservi che, per quanto estremamente improbabile nelle città italiane nel mese di dicembre, questo elemento di criticità potrebbe emergere in caso di dati non registrati.

Infine, la cadenza giornaliera della rilevazioni trova corrispondenza sull'asse temporale di invio dei messaggi NOTE-ON e dei corrispettivi NOTE-OFF.

In definitiva, il risultato atteso della sonificazione è una successione omoritmica di 31 accordi, uno per ciascun giorno del mese. Ogni accordo si compone come sovrapposizione di 3 bicordi, uno per ciascuna stazione di rilevamento. Ogni bicordo include 2 note, quella inferiore relativa alla

temperatura minima e quella superiore alla temperatura massima. L'intensità nella produzione dei singoli bicordi riflette la percentuale di umidità rilevata.

Una prima implementazione, di agevole lettura anche per l'utente non esperto di MATLAB, è mostrata nel Listato 10.6.

```
1  device = mididevice(2);
2  T = readmatrix('open_data.csv');
3  prChMsg1 = midimsg('ProgramChange',1,1);
4  prChMsg2 = midimsg('ProgramChange',2,43);
5  prChMsg3 = midimsg('ProgramChange',3,74);
6  midisend(device,[prChMsg1,prChMsg2,prChMsg3]);
7  for k = 1:size(T,1)
8    % Firenze
9    onMsg1Fi = midimsg('NoteOn',1,60+T(k,1),T(k,3));
10   onMsg2Fi = midimsg('NoteOn',1,60+T(k,2),T(k,3));
11   offMsg1Fi = midimsg('NoteOff',1,60+T(k,1),0,1);
12   offMsg2Fi = midimsg('NoteOff',1,60+T(k,2),0,1);
13   % Milano
14   onMsg1Mi = midimsg('NoteOn',2,48+T(k,4),T(k,6));
15   onMsg2Mi = midimsg('NoteOn',2,48+T(k,5),T(k,6));
16   offMsg1Mi = midimsg('NoteOff',2,48+T(k,4),0,1);
17   offMsg2Mi = midimsg('NoteOff',2,48+T(k,5),0,1);
18   % Palermo
19   onMsg1Pa = midimsg('NoteOn',3,72+T(k,7),T(k,9));
20   onMsg2Pa = midimsg('NoteOn',3,72+T(k,8),T(k,9));
21   offMsg1Pa = midimsg('NoteOff',3,72+T(k,7),0,1);
22   offMsg2Pa = midimsg('NoteOff',3,72+T(k,8),0,1);
23   % invio dei messaggi
24   midisend(device,[onMsg1Fi,onMsg2Fi,onMsg1Mi,onMsg2Mi,onMsg1Pa,onMsg2Pa]);
25   midisend(device,[offMsg1Fi,offMsg2Fi,offMsg1Mi,offMsg2Mi,offMsg1Pa,offMsg2Pa]);
26 end
```

Listato 10.6. Codice per la sonificazione di dati meteo.

Il dispositivo individuato a riga 1, identificato nell'esempio dal valore 2, deve risultare disponibile nel sistema e deve corrispondere a un'uscita. Le operazioni di lettura del file CSV e di creazione di una matrice `T` contenente i dati vengono effettuate a riga 2. La costruzione dei messaggi da inviare si basa sulla lettura ciclica dei valori in `T`. La variabile di controllo k del ciclo `for` contiene, a ogni iterazione, un nuovo indice di riga, mentre i valori di pitch e velocity vengono letti dall'elemento che si trova all'incrocio con una specifica colonna.

Si noti il meccanismo di *offset* temporale attuato sui messaggi di accensione e di spegnimento delle note. Esso comporta, secondo le regole enunciate nel Paragrafo 10.2.2, un primo allineamento tra messaggi NOTE-ON e un secondo allineamento tra messaggi NOTE-OFF a distanza di 1 s. Ogni iterazione comporta l'invio dei messaggi con un'unica invocazione di `midisend()`.

Il Listato 10.7 mostra una versione alternativa, più affine alla scrittura di codice MATLAB.

```
1  device = mididevice(2);
2  T = readmatrix('open_data.csv');
3  prChMsg1 = midimsg('ProgramChange',1,1);
4  prChMsg2 = midimsg('ProgramChange',2,43);
5  prChMsg3 = midimsg('ProgramChange',3,74);
6  midisend(device,[prChMsg1,prChMsg2,prChMsg3]);
7  T(:,[1,2]) = T(:,[1,2]) + 60;
8  T(:,[4,5]) = T(:,[4,5]) + 48;
9  T(:,[7,8]) = T(:,[7,8]) + 72;
10 for k = 1:size(T,1)
11   % Firenze
12   msg1Fi = midimsg('Note',1,T(k,1),T(k,3),1);
13   msg2Fi = midimsg('Note',1,T(k,2),T(k,3),1);
14   % Milano
15   msg1Mi = midimsg('Note',2,T(k,4),T(k,6),1);
16   msg2Mi = midimsg('Note',2,T(k,5),T(k,6),1);
17   % Palermo
18   msg1Pa = midimsg('Note',3,T(k,7),T(k,9),1);
19   msg2Pa = midimsg('Note',3,T(k,8),T(k,9),1);
20   % invio dei messaggi
21   midisend(device,[msg1Fi,msg2Fi,msg1Mi,msg2Mi,msg1Pa,msg2Pa]);
22 end
```

Listato 10.7. Codice alternativo per la sonificazione di dati meteo.

Rispetto al Listato 10.6, i dati sui pitch vengono preparati al di fuori del ciclo, agendo direttamente su `T` (righe 7–9). La dimensione del corpo del ciclo viene sensibilmente ridotta utilizzando eventi di tipo `Note`, che riassumono in sé le funzioni di NOTE-ON e NOTE-OFF, e richiamando una sola volta la funzione `midisend()` al suo interno.

Sarebbero possibili ulteriori approcci migliorativi, ad esempio delegando al ciclo la creazione di oggetti `midimsg` opportunamente temporizzati ed effettuando un'unica chiamata di `midisend()` al di fuori dal ciclo sul vettore o sulla matrice risultanti.

10.6 MIDI Toolbox

In conclusione al capitolo, si ritiene utile menzionare il *MIDI Toolbox*, un insieme di strumenti implementato da terze parti e scaricabile gratuitamente con licenza GNU General Public License. Questo software non costituisce un modulo dell'*Audio Toolbox* e richiede, per poter funzionare, l'installazione di MATLAB.

Gli autori del *MIDI Toolbox* sono Petri Toiviainen, attualmente professore del Dipartimento di Musica dell'Università di Jyväskylä, e Tuomas Eerola, attualmente professore presso la Durham University. I principali siti web di riferimento sono https://www.jyu.fi/hytk/fi/laitokset/mutku/en/research/materials/miditoolbox/ e https://github.com/miditoolbox/. Esistono inoltre pubblicazioni scientifiche incentrate sull'argomento, tra cui si segnalano [10], [11] e [21].

Il *MIDI Toolbox* è una raccolta di strumenti per analizzare e visualizzare i file MIDI in ambiente MATLAB. Oltre a semplici funzioni di manipolazione e filtraggio, esso implementa tecniche di analisi, quali il contorno melodico, la similarità, la ricerca della tonalità, l'individuazione del metro e la segmentazione dei brani. Nato nell'ambito del *data mining* musicale, della modellazione della percezione musicale e della gestione dei dati nella sperimentazione percettiva, lo scopo del software è fornire all'utente strumenti di facile uso, modulari e flessibili per l'analisi e la visualizzazione dell'informazione MIDI.

Il *MIDI Toolbox* permette di aprire Standard MIDI Files e caricarne gli eventi in una matrice, salvare opportune strutture dati di MATLAB come file MIDI, eseguire agevolmente operazioni dal significato musicale quali la trasposizione per un dato intervallo, l'estrazione della voce superiore, il filtraggio degli eventi sulla base delle caratteristiche musicali.

Un altro aspetto di particolare interesse è la visualizzazione dei dati MIDI, richiamabile attraverso specifiche funzioni: ad esempio, è possibile rappresentare gli eventi in formato *piano-roll*, ottenere la distribuzione delle *pitch class* di una performance, e via dicendo. Purtroppo la rappresentazione dell'informazione MIDI propria del *MIDI Toolbox* differisce da quella dell'*Audio Toolbox*, richiedendo quindi forme di conversione per rendere le strutture dati compatibili.

Una trattazione dettagliata dell'argomento esula dagli scopi del presente testo. Per maggiori informazioni si rimanda il lettore alla documentazione precedentemente citata.

Capitolo 11

Csound e MIDI

Csound è un linguaggio di programmazione specifico per il dominio dell'audio. Originariamente progettata da Barry Vercoe nel 1985 al MIT Media Lab [38] e sviluppata in linguaggio C, questa tecnologia ha visto via via crescere il numero di sviluppatori coinvolti ed è stata utilizzata in numerosi progetti artistici [7]. Al giorno d'oggi Csound può contare su un'ampia comunità di volontari che contribuisce attivamente rilasciando esempi, documentazione e articoli e prende parte in modo diretto alla sua crescita segnalando bug, chiedendo funzionalità aggiuntive e intavolando discussioni con il team di sviluppo principale.

Sebbene Csound vanti una forte tradizione nella composizione di brani elettroacustici, questo linguaggio viene attualmente utilizzato da compositori e musicisti per generare o modificare qualsiasi tipo di musica con l'aiuto del computer. A causa dei tempi di elaborazione non trascurabili, in passato Csound è stato principalmente impiegato in contesti non interattivi, ossia predisponendo una partitura prefissata e i relativi strumenti di sintesi, compilando il sorgente e attendendo il completamento del processo di generazione sonora; grazie all'evoluzione tecnologica, tale linguaggio è attualmente utilizzato anche in scenari in tempo reale, il che lo ricollega all'argomento principale di questo volume.

Csound viene spesso definito come un compilatore di suoni. In campo informatico un compilatore è un modulo software che accetta istruzioni testuali sotto forma di codice sorgente e le converte in codice oggetto, che viene a sua volta tradotto in un codice binario, eseguibile sotto forma di programma per computer. Csound funziona in modo simile, ma: 1) il codice oggetto è un flusso di valori numerici che rappresentano campioni audio, e 2) per poter "eseguire il codice", ossia ascoltare il risultato, tale flusso deve essere riprodotto da un convertitore audio digitale/analogico (DAC).

L'obiettivo di Csound è consentire la creazione di timbri (anche) a basso livello, partendo dalle fondamenta stesse della programmazione audio. Csound supporta i tipi di sintesi additiva, sottrattiva, granulare, per formanti, a modulazione di ampiezza e di frequenza, a modelli fisici, e molti altri che ne derivano in modo diretto o indiretto.

Esistono distribuzioni di Csound per i principali sistemi operativi, compresi Android e iOS. È ampiamente supportata anche l'integrazione con altri linguaggi di programmazione come Python, Lua, C/C++, Java e via dicendo.

11.1 Basi sintattiche

Il codice Csound si articola principalmente in due sezioni, dette *orchestra* e *score*, la cui sintassi è espressa come testo semplice (*plain text*).

All'interno dell'*orchestra* vengono definiti gli *instruments*, unità atte a svolgere una serie di funzioni organizzate. Per semplicità si può supporre che si tratti di strumenti musicali, ossia sorgenti sonore caratterizzate da una timbrica, attivate e disattivate nel tempo sulla base di una partitura. Pur essendo questa accezione del tutto comune, va precisato che un *instrument*

all'interno di Csound può svolgere anche altre funzioni, quali, ad esempio, applicare effetti a suoni emessi da altri strumenti o generare sequenze pseudo-casuali di valori numerici.

La definizione degli strumenti Csound è delegata ai cosiddetti *instrument blocks*, porzioni di codice che si aprono con la parola riservata `instr` e si chiudono con la parola riservata `endin`. A ogni strumento viene assegnato un numero (o un nome) al fine di poterlo identificare all'interno dell'*orchestra*. Un esempio di *instrument block* è dato da:

```
instr 1
     ... istruzioni ...
endin
```

Ogni blocco di definizione di uno strumento contiene funzioni più atomiche, dette *opcode* o *unit generator*, che rappresentano i mattoni costitutivi del suo funzionamento. Gli *opcode* svolgono i compiti più disparati, dalla generazione di forme d'onda elementari tramite oscillatori digitali al filtraggio del segnale, passando per il calcolo di funzioni matematiche, la lettura di valori da tabella, la gestione di suoni campionati, il debugging, e via dicendo. A seconda della propria funzione, gli *opcode* presentano uno o più input e output, detti argomenti. La sintassi originaria di Csound prevede che gli argomenti in ingresso si trovino sul lato destro dell'*opcode* e quelli in uscita sul lato sinistro, come nell'esempio seguente:

```
output OPCODE input1, input2, input3, .., inputN
```

Un semplice esempio è dato dall'*opcode* chiamato `oscil`, che, nella sua forma più semplice, realizza un oscillatore sinusoidale controllabile in ampiezza e frequenza e origina una sequenza di valori calcolati alla frequenza di campionamento. Nella riga di codice

```
aSignal oscil 0dbfs/4, 220
```

il primo argomento in ingresso determina un'ampiezza pari a un quarto del valore *0 dB full scale* e il secondo argomento fissa la frequenza a 220 Hz; l'output viene memorizzato in una variabile chiamata `aSignal`, atta a contenere singoli valori ricalcolati alla frequenza di campionamento.

A partire da Csound 6, il codice può essere scritto in modo più simile alla maggior parte dei linguaggi di programmazione:

```
aSignal = oscil(0dbfs/4,220)
```

La sezione di partitura (*score*) contiene un elenco di eventi che descrivono quando accendere, con quali parametri e per quanto tempo mantenere attivi determinati strumenti. Gli eventi di partitura sono singole righe di testo, aperte da una lettera che determina il tipo di istruzione seguita da più argomenti (generalmente) numerici separati da spazi o da tabulazioni.

Accanto a istruzioni per definire tabelle di valori numerici, modificare il tempo metronomico, chiudere sezioni della partitura e via dicendo, assume particolare interesse l'istruzione `i` (*i statement*), che ha lo scopo di far generare un evento sonoro a un dato strumento. La sintassi prevede, dopo la lettera `i`, tre argomenti obbligatori che indicano l'identificativo dello strumento, l'istante di accensione e la durata dell'evento sonoro misurati in pulsazioni. Essendo il metronomo fissato inizialmente a 60 bpm, si può ipotizzare che il secondo e il terzo argomento siano espressi in secondi. Pertanto la sintassi

```
i 1 0 2
```

implica l'accensione dello strumento 1 all'istante 0 con una durata di 2 pulsazioni, ossia 2 s al metronomo predefinito.

Per approfondire la sintassi di Csound si rimanda il lettore alla documentazione ufficiale in rete [1] e a numerosi testi dedicati all'argomento, tra cui [2], [22] e [26].

[1] https://csound.com/docs/manual/

11.2 Struttura del documento

Tradizionalmente, le parti di codice chiamate *orchestra* e *score* venivano posizionate in due file distinti, con estensione .orc e .sco. Eventuali opzioni per indirizzare il comportamento da parte del compilatore venivano specificate da riga di comando attraverso appositi *flag*, argomento che verrà ripreso nel seguito.

Anche se questo metodo è ancora attuabile, è sempre più comune fondere le parti di codice e i *flag* all'interno di un unico documento con estensione .csd strutturato in tre sezioni:

1. *CsOptions*, che contiene le opzioni di configurazione. Ad esempio, la stringa `-o dac`, che indica la ridirezione dell'output su convertitore analogico/digitale, istruisce Csound nell'emettere audio in tempo reale anziché produrre un file su disco;

2. *CsInstruments*, che racchiude la parte di *orchestra*. In questa sezione trovano posto le definizioni degli strumenti, di alcuni parametri opzionali (quali la frequenza di campionamento, il numero di canali, ecc.) e di eventuali funzioni aggiuntive definite dall'utente (*user-defined ocpodes*);

3. *CsScore*, che contiene le istruzioni di partitura riferibili allo *score*.

All'interno delle tre sezioni, la sintassi è basata su testo semplice e ricalca i blocchi di codice originariamente scritti nei file .orc e .sco.

Per mantenere organizzate le sezioni all'interno del documento si fa uso di una sintassi XML che prevede tag di apertura e di chiusura, come mostrato nel Listato 11.1.

```
1  <CsoundSynthesizer>
2
3  <CsOptions>
4  -odac
5  </CsOptions>
6
7  <CsInstruments>
8
9  sr = 44100
10
11 instr 1
12 aSin      oscil    0dbfs/4, 440
13 out       aSin
14 endin
15
16 </CsInstruments>
17
18 <CsScore>
19 i 1 0 1
20 </CsScore>
21
22 </CsoundSynthesizer>
```

Listato 11.1. Scheletro di un documento Csound in formato .csd.

11.3 Csound in tempo reale

L'uso di Csound più tradizionale consiste nella preparazione di uno *score* che richiama strumenti dell'*orchestra*. La compilazione del codice, i cui tempi di calcolo potrebbero essere non trascurabili, porta alla generazione di un file audio su disco, da riprodurre in un secondo momento.

In tempi recenti, grazie al notevole incremento nelle prestazioni dei sistemi di elaborazione, si è sempre più affermato l'uso in tempo reale di Csound, in cui lo *score* può rimanere vuoto o essere solo parzialmente definito, in quanto gli eventi giungono in modo estemporaneo da sorgenti esterne. MIDI è uno dei protocolli di comunicazione supportati: Csound è in grado di leggere da ingressi MIDI, utilizzare file in formato SMF come input e inviare messaggi MIDI.

11.3.1 I flag

I *flag* sono letteralmente indicatori che segnalano una variazione delle condizioni iniziali.

Nella sintassi a riga di comando, i *flag* devono essere inseriti immediatamente dopo il comando `csound`; nella sintassi dei file .csd, vanno collocati nella sezione `<CsOptions>...</CsOptions>`, in una riga intermedia rispetto ai *tag* di apertura e di chiusura.

I nomi dei *flag* sono introdotti dal trattino singolo "–" o doppio "––", e il testo che segue ne costituisce il valore. Talvolta il valore è separato da spazi (o non è in alcun modo separato) rispetto al nome del *flag*, talvolta la sintassi richiede l'uso del carattere "=" per effettuare l'assegnamento del valore al *flag*. Un metodo agevole per ottenere dettagli su tutti i *flag* supportati e sulle rispettive sintassi è digitare da riga di comando

```
csound -help
```

ove `--help` costituisce, a sua volta, un *flag*. L'ordine reciproco dei *flag* non è significativo. La sintassi opera distinzione tra caratteri maiuscoli e minuscoli: ad esempio, i *flag* `-m` e `-M` svolgono funzioni diverse.

11.3.2 Flag per input MIDI in tempo reale

La modalità di ascolto in tempo reale di messaggi MIDI in ingresso viene attivata usando uno dei seguenti *flag*, alternativi tra loro:

```
-M DEVICE_ID                    --midi-device=DEVICE_ID
```

Se l'input MIDI giunge su una sola porta, è necessario specificare, dopo il *flag* `-M` (o dopo il carattere "=" della sintassi alternativa), il numero o il nome del dispositivo cui connettersi. Viene inoltre supportata la sintassi `-Ma` (ove `a` sta per *all*) che implica l'ascolto simultaneo di tutti i dispositivi rilevati in ingresso. Questa impostazione, che funziona anche quando non viene trovato alcun dispositivo MIDI, è utile per ricevere tutti i messaggi prodotti dal sistema MIDI a monte.

A partire dalla versione 6.14, Csound è in grado di gestire porte multiple, che vengono mappate su canali di ordine più alto. In particolare, detto n il numero del dispositivo e c il numero di canale in uso, gli ingressi vengono rimappati sul canale $(n+1) \times c$. Questo non avviene nel caso di utilizzo del *flag* `-Ma`, che fonde canale per canale i messaggi in ingresso su più porte.

Per ottenere l'elenco dei dispositivi MIDI disponibili, e quindi i valori consentiti per `DEVICE_ID`, Csound mette a disposizione il *flag*:

```
--midi-devices
```

Oltre a elencare identificativo e nome dei dispositivi di ingresso e di uscita riconosciuti nel sistema, vengono mostrate informazioni aggiuntive, quali la risoluzione temporale supportata, il valore numerico assoluto associato a *0 dB full scale* (0dBFS) e i moduli MIDI disponibili, argomento cui è dedicato il prossimo paragrafo.

Ad esempio, per leggere in tempo reale i messaggi MIDI provenienti dal dispositivo 2 e usare il file "mymidi.csd" come *orchestra* (e potenzialmente anche come complemento dello *score*), il comando completo è:

```
csound -M2 mymidi.csd
```

11.3.3 Flag per input MIDI da file

I seguenti *flag*, alternativi tra loro, consentono di specificare da riga di comando che gli eventi MIDI devono essere letti da file:

```
-F FILE                         --midifile=FILE
```

ove `FILE` va sostituito dal percorso completo del file da caricare.

Il file MIDI viene letto in tempo reale e si comporta come se i messaggi venissero inviati da una qualsiasi sorgente MIDI. Il codice Csound non ha modo di sapere se l'input provenga da controller o da sequencer.

Quando Csound opera in tempo reale, è prassi non dimensionare temporalmente lo *score*, in modo che il sistema si ponga indefinitamente in attesa di eventi in ingresso. Nel caso dei file MIDI, però, la durata della performance è predeterminata. In questo scenario è possibile specificare nella riga di comando uno dei *flag*

```
-T                          --terminate-on-midi
```

per terminare l'esecuzione al raggiungimento della fine del file MIDI.

Fino alla versione 4 di Csound, il file poteva contenere un'unica traccia, mentre le versioni da Csound 5 in avanti hanno superato questa limitazione.

11.3.4 Flag per output MIDI

Il *flag* `-Q`, seguìto da un identificativo valido, permette di specificare il dispositivo cui inviare messaggi MIDI in uscita da Csound.

Il suo utilizzo consente di gestire parallelamente l'invio di messaggi sul connettore MIDI OUT e la riproduzione di suoni di sintesi attraverso il convertitore digitale/analogico (DAC).

Si osservi che l'intera temporizzazione in tempo reale di Csound si basa sul flusso di campioni nel buffer DAC, coerentemente con la finalità di sintesi del suono. Questo potrebbe però comportare irregolarità nella scansione temporale dei messaggi MIDI in uscita. Un modo per mitigare il problema è dimensionare il buffer di I/O a un numero più piccolo di campioni, operazione che si realizza attraverso il *flag* `-b`.

11.3.5 Moduli MIDI

Pur avendo un'impostazione predefinita, Csound consente di scegliere il modulo per la gestione in tempo reale dei messaggi MIDI. Di seguito l'elenco dei nomi (e, tra parentesi, dei corrispettivi identificativi) dei moduli presenti nei diversi sistemi operativi:

- PortMIDI (`portmidi`), presente su tutte le piattaforme e in grado di supportare ingressi su più porte simultaneamente;
- ALSA (`alsa`), disponibile solo su sistemi Linux;
- JACK (`jack`), server audio che opera come demone e fornisce connessioni a bassa latenza per dati audio e MIDI, disponibile per Linux, macOS, Microsoft Windows, FreeBSD e Solaris/OpenSolaris;
- Windows MME (`winmme`), disponibile solo su sistemi Windows;
- tastiera virtuale grafica (`virtual`), da utilizzare come sorgente di messaggi MIDI, disponibile su tutte le piattaforme.

L'opzione predefinita è PortMIDI, modulo universale, multi-porta e in grado di gestire la comunicazione MIDI in tempo reale, tanto in ingresso, quanto in uscita. Tuttavia, all'utente viene lasciata la possibilità di scelta per migliorare gli aspetti di performance e di affidabilità in specifici contesti. La sintassi a riga di comando è la seguente:

```
-+rtmidi=MODULE_NAME
```

ove `MODULE_NAME` va sostituito con l'identificativo indicato tra parentesi nel precedente elenco.

Nell'esempio seguente si sceglie esplicitamente di usare il modulo PortMIDI, leggendo gli eventi MIDI in ingresso dal dispositivo 2 e inviando i messaggi MIDI in uscita al dispositivo 1:

```
-+rtmidi=portmidi -M2 -Q1
```

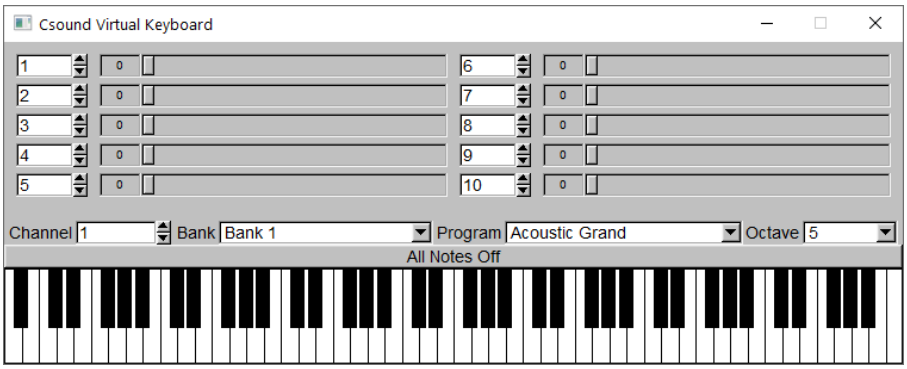


Figura 11.1. Interfaccia grafica della tastiera virtuale Csound in ambiente Windows.

Csound mette a disposizione anche una tastiera virtuale (Figura 11.1). Attivata dal *flag* `-+rtmidi=virtual`, essa può inviare messaggi NOTE-ON, NOTE-OFF e PROGRAM CHANGE, selezionare il banco di suoni e generare altri messaggi CONTROL CHANGE; a quest'ultimo scopo sono presenti 10 slider collegati di default ai controller MIDI da 1 a 10, riconfigurabili canale per canale.

Anche se il modulo MIDI è virtuale, si rende comunque necessario fornire un valore al *flag* `-M`, normalmente posto a 0. L'invocazione completa da riga di comando diventa dunque:

```
csound -+rtmidi=virtual -M0 prova.csd
```

Valori diversi da 0 per il *flag* `-M` consentirebbero di caricare appositi file per rimappare la tastiera, consentendo all'utente di personalizzare 1) nome e numero dei banchi e 2) nome e numero dei timbri all'interno di ciascun banco. Un esempio di *keyboard mapping file* è disponibile all'URL https://csound.com/docs/manual/MidiTop.html.

11.4 Instradamento di messaggi MIDI a strumenti Csound

Una volta attivati ingressi (*flag* `-M` e `-F`) e/o uscite (*flag* `-Q`) in tempo reale, è possibile gestire i messaggi MIDI all'interno di Csound. In questa sezione del testo ci si occuperà in particolare dei messaggi in ingresso al sistema.

Riguardo agli ingressi, ogni messaggio di NOTE-ON genera un evento sonoro destinato allo strumento Csound identificato dal numero del canale MIDI in uso. In questo modo si crea un'associazione biunivoca tra i canali $1, 2, \ldots, 16$ e gli strumenti $1, 2, \ldots, 16$. Fa eccezione il caso in cui sia presente un solo *instrument* nella sezione di *orchestra*: in questo scenario Csound opera in una sorta di *omni mode*, indirizzando tutti gli eventi in ingresso a quell'unico strumento, indipendentemente dall'informazione di canale. Lo strumento 1 è quello normalmente utilizzato anche per gestire i messaggi in arrivo su canali cui non corrisponde uno strumento Csound.

Si osservi che, all'interno di Csound, ogni strumento MIDI diventa potenzialmente polifonico, in quanto tutti i messaggi indirizzati a un certo canale provocano accensioni indipendenti dello strumento cui sono associati.

La corrispondenza predefinita tra canali e strumenti può essere modificata attraverso opportuni *opcode* nell'*orchestra*, normalmente (ma non necessariamente) localizzati nella sezione di intestazione, prima degli *instrument block*.

In questa sezione, e nel seguito, la sintassi degli *opcode* fa riferimento alla documentazione ufficiale, anche per quanto riguarda i nomi attribuiti agli argomenti. Le parentesi quadre indicano una parte opzionale dell'istruzione.

11.4.1 Opcode `massign`

Un primo *opcode* che determina l'instradamento di messaggi MIDI in ingresso a specifici strumenti Csound è `massign`, che opera sulla base dell'informazione di canale. Le due varianti sintattiche

```
massign ichnl, insnum[, ireset]
massign ichnl, "insname"[, ireset]
```

assegnano a un numero di canale MIDI (argomento `ichnl`) uno specifico strumento Csound, identificato da un numero (argomento `insnum`) o da un nome (argomento `"insname"`, espresso tra doppi apici in quanto stringa). Un esempio di modifica nella corrispondenza tra canale e strumento Csound è

```
massign 3,20
```

che assegna lo strumento 20 al Canale MIDI 3.

È possibile eseguire il *muting* di uno specifico canale ponendo l'argomento `insnum` a un valore nullo o negativo. È inoltre possibile porre `ichnl` a 0, al fine di utilizzare lo stesso valore di `insnum` per tutti i canali. Ne consegue che la sintassi sotto riportata disabiliti la gestione da parte di Csound delle note MIDI in arrivo su qualsiasi canale:

```
massign 0,0
```

ed eventuali ulteriori chiamate di `massign` andrebbero a modificare puntualmente questa impostazione generale.

Specificando un valore non nullo per l'argomento opzionale `ireset`, i controller associati vengono riportati ai valori predefiniti; si tratta anche dell'impostazione di default ottenuta richiamando l'*opcode* senza esprimere un valore per `ireset`.

11.4.2 Opcode `pgmassign`

Sempre riguardo all'instradamento dei messaggi MIDI in ingresso, esiste la possibilità di assegnare uno strumento Csound sulla base del *program number*, ossia della selezione di un timbro o una patch attraverso il messaggio di canale PROGRAM CHANGE. L'*opcode* che realizza tale funzione è `pgmassign`, che supporta le seguenti varianti:

```
pgmassign ipgm, inst[, ichn]
pgmassign ipgm, "insname"[, ichn]
```

Riguardo agli argomenti, `ipgm` rappresenta il numero di programma nell'intervallo [1, 128]; il valore 0 seleziona tutti i numeri di programma.

L'argomento `inst` identifica il numero di strumento; se posto a un valore nullo o negativo, i messaggi PROGRAM CHANGE vengono ignorati. Identico effetto si verifica quando il numero di programma non trova corrispondenza in Csound, ma è previsto che future versioni mostrino un messaggio di errore. Pertanto la sintassi

```
pgmassign 0, 0
```

implica che tutti i PROGRAM CHANGE vengano ignorati, indipendentemente dal canale.

L'argomento `"insname"`, tra doppi apici in quanto stringa, permette di specificare uno strumento Csound per nome anziché per identificativo numerico.

Infine, l'argomento opzionale `ichn` riguarda il numero di canale cui assegnare i PROGRAM CHANGE; quando posto a 0, che rappresenta anche il valore di default, i PROGRAM CHANGE vengono assegnati a tutti i canali.

Poiché ogni canale può essere associato a un timbro, peraltro variabile nel tempo per via di PROGRAM CHANGE, le mappature determinate da `massign` e `pgmassign` agiscono in modo potenzialmente concorrente. Il loro effetto verrà chiarito attraverso l'esempio riportato nel Paragrafo 11.8.2.

11.5 Lettura di pitch e velocity attraverso flag

Nel precedente paragrafo è stata discussa la logica di assegnamento dei canali e dei timbri MIDI agli strumenti Csound. Va ora specificato in che modo l'informazione trasportata dai byte di dati, in particolare quelli dei messaggi di performance quali NOTE-ON e NOTE-OFF, possa essere recuperata all'interno di un *instrument block* al fine di pilotarne il funzionamento.

Un primo modo per determinare come i messaggi di NOTE-ON in arrivo controllino ampiezza e frequenza degli oscillatori (o altri parametri) consiste nell'uso di *flag* appositi. Metodi alternativi basati sugli *opcode* verranno mostrati nel Paragrafo 11.6.

Riguardo all'altezza (o pitch) della nota:

- `--midi-key-cps=N` reindirizza al *p-field* `N` nell'*instrument block* il pitch espresso come intero nell'intervallo [0, 127]. A titolo di esempio, con `N` = 4, il *p-field* che contiene tale informazione è `p4`. Se questo valore fosse destinato a controllare la frequenza di un oscillatore digitale si renderebbe necessaria una conversione esplicita in hertz, come mostrato nella Formula 2.5;
- `--midi-key-cps=N` reindirizza al *p-field* `N` il pitch convertito in numero di cicli al secondo, quindi espresso in hertz;
- `--midi-key-oct=N` reindirizza al *p-field* `N` il pitch convertito in un valore lineare di ottava attraverso la notazione detta *octave-point-decimal*. In questa particolare rappresentazione, il *do* centrale corrisponde al valore 8.00, il *do* all'ottava superiore al valore 9.00 e il *fa*♯, trovandosi in posizione esattamente intermedia, al valore 8.50;
- `--midi-key-pch=N` reindirizza al *p-field* `N` il pitch in notazione *octave-point-pitch-class*, ove la parte intera del valore numerico aggregato corrisponde al numero di ottava e la parte decimale allo scostamento in numero di semitoni rispetto al *do* naturale dell'ottava stessa. In questa scala il *do* centrale corrisponde al valore 8.00, il *do* all'ottava superiore al valore 9.00, il *do*♯ dell'ottava centrale al valore 8.01 e il *fa*♯ dell'ottava centrale al valore 8.07.

Riguardo all'intensità (o velocity) della nota:

- `--midi-velocity=N` deposita nel *p-field* `N` nell'*instrument block* il valore di intensità espresso nell'intervallo [0, 127];
- `--midi-velocity-amp=N` pone all'interno del *p-field* `N` il valore di intensità come ampiezza linearizzata nell'intervallo [0, m], ove m è il valore associato a *0 db full scale* (di default 32.767).

Si rammenta al lettore che i *flag* devono essere posizionati o nella riga del comando `csound` o nella sezione `<CsOption>...</CsOptions>` del file .csd.

11.6 Altri opcode per il MIDI

In questo paragrafo vengono elencati i principali *opcode* per il MIDI in aggiunta a quelli già mostrati nel Paragrafo 11.4: `massign` e `pgmassign`.

Per una trattazione più articolata e corredata da esempi si rimanda il lettore alla documentazione ufficiale di Csound [2].

11.6.1 Lettura e conversione del pitch

Il principale *opcode* per l'estrazione dell'informazione di pitch da un messaggio MIDI in ingresso è `notnum`, la cui sintassi è:

```
ival notnum
```

[2] Gli *opcode* trattati nel seguito si collocano nella sezione "Orchestra Opcodes and Operators" del manuale online, http://www.csounds.com/manual/html/PartReference.html.

Csound, in quanto linguaggio per la programmazione timbrica, in generale predilige informazione di altezza espressa in hertz anziché nell'intervallo [0, 127] tipico di MIDI. Una riga di codice che effettua in modo esplicito tale conversione è

```
iHz = (440.0 * exp(log(2.0) * ((ival) - 69.0) / 12.0))
```

Come mostrato nel seguito, esistono *opcode* che effettuano la conversione da pitch MIDI in modo diretto, restituendo valori in hertz o in altri formati di rappresentazione meglio supportati da Csound.

Conversione da pitch MIDI a cicli al secondo

Gli *opcode* presentati in questo paragrafo hanno a che vedere con il numero di nota MIDI, ossia con il pitch, e la sua conversione in un'altezza espressa in cicli al secondo, ossia in hertz.

Il primo di essi, `cpsmidi`, recupera il numero di nota dell'evento MIDI corrente e lo converte in cicli al secondo. Il valore in hertz così ottenuto è quanto richiesto – in generale – dalle unità di generazione del suono Csound, quali gli *opcode* della famiglia degli oscillatori digitali. La sintassi non prevede argomenti in ingresso, ma solo una variabile calcolata a *i-rate* e atta a contenere il valore in uscita dall'*opcode*:

```
icps cpsmidi
```

L'istante temporale sopra definito *i-rate* corrisponde all'inizializzazione dell'istanza dello strumento. Nell'ipotesi – del tutto plausibile – che lo strumento in questione venga acceso da un evento MIDI in ingresso, il pitch verrebbe recuperato dal corrispondente NOTE-ON e non risulterebbe più modificabile per tutto il tempo di vita dell'istanza; peraltro un nuovo NOTE-ON sullo stesso canale, e quindi destinato allo stesso strumento Csound, avvierebbe una nuova istanza dello strumento, senza interferire con quella in esame.

L'*opcode* `cpsmidinn` apparentemente svolge la stessa funzione di `cpsmidi`, ma con una fondamentale differenza: quest'ultimo produce risultati significativi solo quando l'accensione dello strumento è provocata da eventi MIDI generati in tempo reale (*flag* `-M`) o letti da file MIDI (*flag* `-F`), leggendo il valore del pitch dall'evento MIDI associato all'istanza dello strumento corrente; l'*opcode* `cpsmidinn`, invece, converte un valore passato in ingresso e può essere impiegato in qualsiasi strumento Csound, indipendentemente dalla sua attivazione da parte di un evento MIDI, di un evento sonoro specificato nello *score* o altro ancora. La sintassi diventa

```
cpsmidinn (MidiNoteNumber)
```

ove `MidiNoteNumber` è un valore o un'espressione che contiene argomenti calcolati al passo di inizializzazione dello strumento (a *i-rate*) o alla frequenza di controllo (a *k-time*). Il valore di ingresso per `cpsmidinn` potrebbe derivare da un *p-field* dello *score* o da un evento MIDI in tempo reale che provoca l'accensione dello strumento passando per l'*opcode* `notnum` precedentemente descritto.

L'*opcode* `cpsmidib` è una variante di `cpsmidi` che recupera il numero di nota MIDI, modifica l'informazione di altezza prendendo in considerazione il valore corrente di *pitch bend* e, infine, restituisce un risultato in hertz. Le due versioni della sintassi, differenziate dalla sola frequenza di calcolo dell'output, sono

```
icps cpsmidib [irange]
kcps cpsmidib [irange]
```

L'argomento opzionale `irange` consente di determinare l'intervallo in semitoni ammesso per il *pitch bend*.

L'*opcode* `cpstmid` è un'ulteriore variante di `cpsmidi` che ammette la costruzione di scale musicali con micro-intonazione sotto il controllo dell'utente. Nonostante il funzionamento richieda quattro

parametri, chiamati `numgrades`, `interval`, `basefreq` e `basekeymidi`, la sintassi dell'opcode è molto semplice:

```
icps cpstmid ifn
```

Infatti, la conversione da pitch a hertz viene demandata ai contenuti della tabella funzione `ifn` appositamente costruita. Usando una funzione generatrice `GEN02` è possibile elencare esplicitamente la serie di argomenti richiesta, che include innanzi tutto i seguenti quattro valori:

1. `numgrades`, numero di gradi della scala;
2. `interval`, indicatore della banda di frequenze coperta prima di ripetere i rapporti di micro-intonazione della scala. Ad esempio, 2 implica un intervallo di ottava, 1,5 una quinta giusta e via dicendo;
3. `basefreq`, frequenza base della scala espressa in hertz;
4. `basekeymidi`, numero di nota MIDI cui viene associata (senza modifiche) la frequenza `basefreq`.

A seguire, in tabella per ciascuno dei `numgrades` gradi della scala viene riportato il fattore moltiplicativo per la micro-intonazione.

Il segno negativo prima del valore 2 che identifica la funzione generatrice `GEN02` serve a evitare che i contenuti numerici della tabella vengano normalizzati nell'intervallo [-1, +1]; si tratta di uno strumento sintattico di uso comune nella programmazione Csound.

Un primo esempio per comprendere il funzionamento di `cpstmid` si basa su una seguente tabella funzione che definisce una scala cromatica standard composta da 12 semitoni, in cui la frequenza base 261,63 Hz viene posta in corrispondenze con il pitch 60 e la sequenza si ripete a intervalli di ottava:

```
;             arg1   arg2   arg3   arg4   tuning ratios
f1 0 64 -2    12     2      261.63 60     1  1.059  1.122  ...
```

Per motivi di spazio nella riga di commento sono state riportate le etichette `arg1`, `arg2`, `arg3` e `arg4` in luogo di `numgrades`, `interval`, `basefreq` e `basekeymidi`. Si osservi che i fattori moltiplicativi per le 12 altezze sono potenze di $\sqrt[12]{2}$, coerentemente con il cosiddetto temperamento equabile:

$$(\sqrt[12]{2})^0 = 1$$

$$(\sqrt[12]{2})^1 = 2^{1/12} \approx 1,059$$

$$(\sqrt[12]{2})^2 = 2^{1/6} \approx 1,122$$

$$\ldots$$

Un secondo esempio, decisamente più originale, riguarda la definizione di una scala di 24 note a partire dalla frequenza base di 220 Hz associata al pitch 48, la cui sequenza si ripete a intervalli di quinta giusta:

```
;             arg1   arg2   arg3   arg4   tuning ratios
f1 0 64 -2    24     1.5    220    48     1  1.01  1.02  ...
```

In generale è plausibile – anche se non strettamente richiesto – che il primo fattore moltiplicativo nella sequenza sia 1, in quanto un valore diverso implicherebbe una scelta poco accorta di `basefreq`.

Conversione da pitch MIDI a octave-point-pitch-class

Un insieme di *opcode* è volto a convertire i pitch MIDI in altre forme di rappresentazione dell'altezza dei suoni supportate da Csound. Un primo metodo è detto *pitch class*, in alcuni testi chiamato più estensivamente *octave-point-pitch-class*. Secondo questa forma di rappresentazione l'ottava costituisce la parte intera del valore aggregato, mentre lo scostamento in semitoni – non necessariamente intero – è delegato alla parte decimale e posto in corrispondenza con i centesimi. Ad esempio, il *do* centrale corrisponde al valore 8.00, il *do*♯ a 8.01, il *la* centrale a 8.09 e il *do* all'ottava superiore a 9.00.

Per completezza si riportano alcuni scenari particolari. Innanzi tutto, volendo rappresentare micro-intervalli, è possibile usare ulteriori cifre decimali; ad esempio, un *do* centrale alterato di quarto di tono temperato in senso ascendente corrisponde al valore 8.005. Un altro esempio riguarda la (mancata) circolarità del vettore delle cifre decimali: nonostante fuoriesca dall'intervallo [0, 11] accettato per i semitoni in un'ottava, la notazione 8.12 non genera errore, costituendo piuttosto un alias di 9.00; analogamente, 8.13 diventa un alias di 9.01; e via dicendo.

Un primo *opcode* rivolto alla rappresentazione *octave-point-pitch-class* è `pchmidi`, la cui sintassi

```
ipch pchmidi
```

restituirebbe 8.00 per il pitch 60, 8.01 per il pitch 61, ..., 9.00 per il pitch 72 e via dicendo.

Analoga operazione viene svolta da `pchmidinn`, con la stessa differenza di comportamento evidenziata tra `cpsmidi` e `cpsmidinn` nel paragrafo precedente. Anche in questo caso, la sintassi si arricchisce di un valore numerico o un'espressione in ingresso per specificare o calcolare il numero di nota MIDI:

```
pchmidinn (MidiNoteNumber)
```

Infine, le due varianti dell'*opcode* `pchmidib` tengono conto anche del valore corrente di *pitch bend*:

```
ipch pchmidib [irange]
kpch pchmidib [irange]
```

Conversione da pitch MIDI a octave-point-decimal

Un'ulteriore forma di rappresentazione dell'altezza supportata da Csound è la cosiddetta notazione *octave-point-decimal*. Ancora una volta un singolo valore numerico con separatore decimale consente di aggregare, mantenendole distinte, le informazioni relative a ottava e scostamento interno all'ottava. In questo caso, però, la parte decimale non rappresenta il numero di semitoni di distanza dalla prima nota dell'ottava, bensì la frazione dell'intervallo di ottava. Ad esempio, il valore 8.50 sta a indicare una frequenza posta esattamente a metà strada tra 8.00 (*do* naturale dell'ottava centrale) e 9.00 (*do* naturale dell'ottava superiore): nel sistema temperato si tratta della frequenza di *fa*♯ o *sol*♭, posta a 6 semitoni di distanza rispetto a entrambi gli estremi dell'intervallo.

Anche la notazione *octave-point-decimal* permette di approssimare qualsiasi distanza intervallare rispetto alla frequenza base dell'ottava, prestandosi a interventi di micro-intonazione.

Le controparti degli *opcode* sopra descritti la notazione *octave-point-pitch-class* sono, rispettivamente, `octmidi`, `octmidinn` e `octmidib`.

11.6.2 Lettura della velocity

In Csound esistono principalmente due *opcode* per la gestione dell'intensità degli eventi MIDI in ingresso: `ampmidi` e `ampmidid`.

Il primo di questi recupera la velocity dell'evento MIDI corrente, in pratica il valore numerico del secondo byte di dati di NOTE-ON. La sua sintassi è

```
iamp ampmidi iscal [, ifn]
```

ove `iscal` rappresenta il fattore di scala che porta l'originario intervallo [0, 127] a diventare [0, `iscal`]. Una scelta tipica dell'audio digitale è impostare `iscal` a 1. Il secondo argomento, opzionale e posto a 0 di default, permette di definire una mappatura tra informazione di ampiezza e valori numerici arbitrari, grazie all'uso di una tabella funzione specificata da `ifn`.

Altro *opcode* il cui funzionamento è molto simile a `ampmidi` è `veloc`, che recupera la velocity di un evento MIDI consentendo di ridistribuirne il valore, se richiesto, nell'intervallo [ilow, ihigh]:

```
ival veloc [ilow] [, ihigh]
```

L'*opcode* `ampmidid` è, invece, un'evoluzione di `ampmidi` che effettua una conversione in termini musicali tra la velocity dell'evento MIDI e l'ampiezza di picco di una gamma dinamica espressa in decibel. Tale mappatura si basa sulla seguente formula, presentata in [9]:

$$a = (m \cdot v + b)^2$$

in cui a indica l'ampiezza espressa come valore lineare, v è la velocity MIDI, $m = (1 - b)/127$ e, infine,

$$b = 127/(126 \cdot \sqrt{r}) - 1/126$$

ove $r = 10^{(R/20)}$ ed R rappresenta il valore massimo della gamma dinamica espresso in decibel.

La due sintassi alternative dell'*opcode* sono

```
iamplitude ampmidid ivelocity, idecibels
iamplitude ampmidid kvelocity, idecibels
```

e richiedono entrambe un valore di velocity nell'intervallo [0, 127] come primo argomento e l'estremo destro della gamma dinamica [0, `idecibels`] come secondo argomento. Le due sintassi differiscono per la frequenza di ricalcolo del primo argomento, il cui valore può essere determinato al solo passo di inizializzazione (prima riga) e tenuto fisso per l'intera accensione dello strumento, oppure variare alla frequenza di calcolo delle variabili di controllo (seconda riga).

11.6.3 Lettura di dati MIDI

Csound include un gran numero di *opcode* pensati per recuperare informazione specifica dai messaggi MIDI in ingresso.

Numero di canale, byte di stato e di dati

L'*opcode* `midichn`, la cui sintassi è

```
ichn midichn
```

restituisce il numero di canale MIDI della nota che ha attivato lo strumento.

L'*opcode* `midiin`, la cui sintassi è

```
kstatus, kchan, kdata1, kdata2 midiin
```

parcellizza l'informazione in arrivo da un generico messaggio MIDI in ingresso, inviando sui quattro argomenti in uscita: 1) il valore del byte di dati al netto dell'informazione di canale (`kstatus`), 2) l'informazione di canale (`kchan`), 3) il valore del primo byte di dati ((`kdata1`) e 4) il valore del secondo byte di dati (`kdata2`). Si tratta di valori ricalcolati a *k-rate*, quindi potenzialmente variabili durante l'accensione dello strumento.

Esistono *opcode* specializzati per la ricezione di messaggi specifici, tra cui:

- `midinoteoff` per NOTE-OFF;
- `midichannelaftertouch` per CHANNEL PRESSURE;
- `midicontrolchange` per CONTROL CHANGE. Per leggere il solo valore di uno specifico controller MIDI è invece disponibile `midictrl`;

- `midipitchbend` per PITCH BEND CHANGE;
- `midipolyaftertouch` per POLYPHONIC KEY PRESSURE;
- `midiprogramchange` per PROGRAM CHANGE.

Opcode per i file MIDI

L'*opcode* `miditempo` restituisce il tempo corrente del file MIDI (se disponibile) o dello score, aggiornando tale informazione a *k-rate*. La sintassi è particolarmente semplice:

```
ksig miditempo
```

e il valore del tempo può essere visualizzato attraverso un'istruzione quale

```
prints "miditempo = %d\n", ksig
```

Va comunque sottolineato che il risultato è significativo solo nel caso di file MIDI, il cui metronomo viene restituito come un usuale bpm (ad esempio, 96 o 120); nel caso di eventi provenienti dallo *score*, invece, il risultato è pari alla frequenza di campionamento moltiplicata per il metronomo predefinito, ossia 60 bpm, e non varia nemmeno se viene specificata nuova informazione di metronomo attraverso un *t statement*.

Altro *opcode* interessante è `midifilestatus`, che restituisce lo stato di esecuzione (*playback*) del file MIDI in ingresso aggiornandolo a *k-rate*:

```
ksig midifilestatus
```

L'argomento `ksig` assume valore 1 se il file è in esecuzione, 0 se è stata raggiunta la sua conclusione.

Approfondimento: opcode `chanctrl`

In questo paragrafo si approfondisce, a titolo esemplificativo, l'*opcode* `chanctrl`. Esso legge il valore dei messaggi CONTROL CHANGE su un determinato canale e aventi uno specifico *CC number*, consentendo, inoltre, di ridistribuire tale valore su uno specifico intervallo.

La due sintassi alternative dell'*opcode* sono

```
ival chanctrl ichnl, ictlno [, ilow [, ihigh]]
kval chanctrl ichnl, ictlno [, ilow [, ihigh]]
```

e si differenziano solo per il tipo di argomento in uscita, rispettivamente calcolato a *i-rate* o a *k-rate*.

Riguardo agli argomenti, `ichnl` rappresenta il numero di canale, `ictlno` il numero di controller e i parametri opzionali `ilow` e `ihigh` gli estremi sinistro e destro dell'intervallo al cui interno riposizionare il valore originariamente nel *range* [0, 127]. Ad esempio, la riga

```
ival chanctrl 1, 7
```

legge il valore dei messaggi CONTROL CHANGE con primo byte di dati posto a 7 ("Channel Volume") in arrivo sul Canale 1.

Ad esempio, avendo posto il valore di *0 dB full scale* a 1 nell'intestazione, è possibile pilotare l'ampiezza di un oscillatore attraverso la riga

```
asig oscil ival*(1/127), 220, 1
```

oppure optare per un ridimensionamento dei valori di `ival` nell'intervallo [0, 1] attraverso la riga

```
ival chanctrl 1, 7, 0, 1
```

11.6.4 Interoperabilità tra MIDI e *score*

In alcuni contesti è comodo gestire i valori in ingresso a un dato strumento Csound prescindendo dalla sua attivazione da parte di MIDI o di eventi sonori nello *score*. In questo scenario si rivelano particolarmente utili *opcode* quali `midinoteoncps`, `midinoteonkey`, `midinoteonoct` e `midinoteonpch`, i quali salvano, all'interno di appositi argomenti, i valori di pitch e velocity letti da eventi MIDI, potenzialmente in concorrenza con quelli provenienti dallo *score*.

La differenza tra i quattro *opcode* consiste nella codifica dell'altezza, rispettivamente una frequenza in cicli al secondo, un numero di nota MIDI, un valore *octave-point-decimal* e un valore *octave-point-pitch-class*.

La sintassi di questi *opcode* è particolare: gli argomenti al cui interno depositare i valori compaiono sul lato destro, nella posizione normalmente riservata ai parametri in ingresso.

A titolo di esempio, la riga di codice

```
midinoteoncps p4, p5
```

pone nei *p-field* chiamati `p4` e `p5` il valori di pitch convertito in hertz e il valore di velocity provenienti dall'attivazione dello strumento da parte di un evento MIDI. Se lo strumento venisse istanziato dallo score, i valori di `p4` e `p5` giungerebbero invece dall'*i statement*. In questo modo un unico strumento funziona con entrambe le sorgenti, senza necessità di gestire la differente provenienza dei dati tramite costrutti condizionali.

Un *opcode* che, al contrario, rende esplicita tale informazione è `mididefault`, la cui sintassi è

```
mididefault xdefault, xvalue
```

Esso specifica un valore di default, chiamato `xdefault` e variabile nel tempo, che verrà usato durante l'attivazione dello strumento da parte di MIDI, sovrascrivendo, in tal caso, l'argomento `xvalue`; l'argomento `xvalue`, anch'esso potenzialmente variabile nel tempo, rimarrà invece inalterato in caso di attivazione dello strumento da parte dello *score*.

Al lettore interessato si segnala un'intera sezione della documentazione Csound dedicata all'interoperabilità tra MIDI e *score*, disponibile all'URL http://www.csounds.com/manual/html/MidiInterop.html.

11.6.5 Invio di messaggi MIDI

Csound, oltre a leggere dati in ingresso da MIDI IN, può inviare messaggi su MIDI OUT. La necessaria premessa è aver specificato un dispositivo di uscita valido attraverso il *flag* `-Q`.

A seconda dell'*opcode*, il corrispettivo messaggio MIDI può essere inviato a *i-rate*, ossia solo quando lo strumento viene istanziato, o a *k-rate*, ossia alla frequenza di calcolo delle variabili di controllo. Un valore standard per *k-rate* è 1/10 della frequenza di campionamento, ad esempio 4.410 Hz quando la frequenza di campionamento è 44.100 Hz.

Opcode `midiout`

La funzione più generale di invio di messaggi di performance è `midiout`, la cui sintassi è

```
midiout kstatus, kchan, kdata1, kdata2
```

La variabile di controllo `kstatus` determina il tipo di messaggio MIDI, e può assumere i valori 128 (NOTE-OFF), 144 (NOTE-ON), 160 (POLYPHONIC KEY PRESSURE), 176 (CONTROL CHANGE), 192 (PROGRAM CHANGE), 208 (CHANNEL PRESSURE), 224 (PITCH BEND CHANGE) e 0 (nessun messaggio). Si nota come questi valori siano la rappresentazione in base 10 del valore del byte di stato dei rispettivi messaggi, al netto dell'informazione di canale[3]}. La sintassi permette poi di inviare

[3] Si tratta, parimenti, del valore del byte di stato per i messaggi di canale inviati sul Canale 1: presentando *nibble* di ordine basso pari a 0, l'informazione di canale in questo caso non incide sul valore numerico complessivo.

separatamente l'informazione di canale che completa il byte di stato (`kchan`) e il valore dei due byte di dati (`kdata1` e `kdata2`).

Si può notare come `midiout` non presenti argomenti in uscita, in quanto la sua funzione si esaurisce con l'invio di messaggi sulla connessione MIDI OUT.

L'*opcode* opera a *k-rate*, ossia alla frequenza di calcolo delle variabili di controllo. Questo significa che, mantenendo `kstatus` a uno dei valori ammessi e non nulli, Csound genererebbe potenzialmente migliaia di messaggi al secondo. Per questo motivo il valore di default per l'argomento `kstatus` è 0: in un normale contesto di funzionamento, esso deve essere modificato in modo puntuale solo quando si intende inviare un messaggio MIDI, per poi essere riportato immediatamente a 0. Anche il valore degli altri argomenti dell'*opcode* può variare a *k-rate*.

Opcode `noteon`, `noteoff` e `noteondur`

Gli *opcode* `noteon` e `noteoff` inviano su una connessione MIDI OUT i rispettivi messaggi MIDI, secondo una sintassi che non merita particolari commenti aggiuntivi:

```
noteon ichn, inum, ivel
noteoff ichn, inum, ivel
```

Poiché gli *opcode* vanno richiamati entrambi – a una certa distanza temporale – al fine di non mantenere indefinitamente accesi i suoni, essi sono spesso associati all'uso dell'istruzione `timout`.

Nella maggior parte dei casi questa coppia di *opcode* può essere validamente sostituita da `noteondur` e `noteondur2`, *opcode* che inviano tanto un NOTE-ON quanto un NOTE-OFF con uguali valori di canale, pitch e velocity a una distanza temporale determinata in secondi da `idur`:

```
noteondur ichn, inum, ivel, idur
noteondur2 ichn, inum, ivel, idur
```

Il primo *opcode* tronca la nota quando l'istanza dello strumento viene disattivata, mentre `noteondur2` prolunga la sua esecuzione fino al naturale esaurimento del tempo `idur`.

Opcode `midion` e `midion2`

Gli *opcode* `midion` e `midion2` inviano a *k-rate* messaggi di NOTE-ON (e NOTE-OFF) alla connessione MIDI OUT. Le loro sintassi sono

```
midion kchn, knum, kvel
midion2 kchn, knum, kvel, ktrig
```

e si differenziano per la presenza di un trigger: `midion2` funziona solo quando il valore di `ktrig` è non nullo.

Normalmente, l'invio riguarda messaggi di tipo NOTE-ON, in quanto i corrispettivi NOTE-OFF vengono gestiti dallo spegnimento dello strumento Csound. Come è noto al lettore, però, l'adozione di `kvel = 0` permette di forzare lo spegnimento di una nota attraverso l'invio di un messaggio NOTE-ON.

Un esempio basato su `midion` verrà presentato nel Paragrafo 11.8.4.

Opcode specializzati a *i-rate* e *k-rate*

Esistono *opcode* specializzati sulla base del messaggio MIDI da inviare e dotati di una doppia versione a seconda del funzionamento a *i-rate* o a *k-rate*. In particolare, si tratta di:

- `outipc` e `outkpc` per i messaggi di PROGRAM CHANGE;
- `outipb` e `outkpb` per i messaggi di PITCH BEND CHANGE;
- `outiat` e `outkat` per i messaggi di CHANNEL PRESSURE, definiti "aftertouch" nelle specifiche Csound (donde il nome degli *opcode*);

- `outipat` e `outkpat` per i messaggi di POLYPHONIC KEY PRESSURE, definiti "polyphonic aftertouch" (donde il nome degli *opcode*);
- `outic` e `outkc` e la loro controparte a 14 bit `outic14` e `outkc14` per i messaggi di CONTROL CHANGE. La versione a 14 bit consente di inviare in un unico messaggio tanto la parte MSB quanto LSB dei controller a granularità fine.

A titolo di esempio, la sintassi del primo degli *opcode* elencati è

```
outipc ichn, iprog, imin, imax
```

ove `ichn` indica il numero di canale MIDI nell'intervallo [1, 16], `iprog` è il numero di programma da selezionare e `imin` e `imax` sono i limiti dell'intervallo in ingresso cui far corrispondere il classico intervallo in uscita [0, 127], rispetto al quale `iprog` va ridimensionato. Questa apparente complicazione, oltre a fornire un maggior grado di flessibilità, tiene conto del fatto che i valori numerici in Csound siano in virgola mobile, non esistendo nel linguaggio il concetto di numero intero. Risulta comunque agevole realizzare l'invio di un messaggio PROGRAM CHANGE con gli usuali valori ponendo `imin` = 0 e `imax` = 127, come nell'esempio sottostante:

```
outipc 1, 80, 0, 127
```

Opcode `mclock` e `mrtmsg`

Oltre a inviare messaggi di performance, Csound può anche generare ed emettere messaggi della famiglia *System Real Time*.

L'*opcode* `mclock` si concentra su messaggi TIMING CLOCK. La sintassi

```
mclock ifreq
```

provoca la generazione di un messaggio TIMING CLOCK ($F8_{16}$) alla frequenza `ifreq`.

Per altri messaggi della famiglia *System Real Time* esiste l'apposito *opcode* `mrtmsg`:

```
mrtmsg imsgtype
```

Anche in questo caso, coerentemente con le specifiche MIDI, non è necessario specificare alcuna informazione ulteriore rispetto al tipo di messaggio. I valori supportati per l'argomento `imsgtype` sono 1 per START (FA_{16}), 2 per CONTINUE (FB_{16}), 0 per STOP (FC_{16}), -1 per SYSTEM RESET (FF_{16}) e -2 per ACTIVE SENSING (FE_{16}).

11.7 Durata della performance

Potendo prescindere dal funzionamento interattivo in tempo reale, Csound effettuerebbe un'elaborazione dell'elenco completo degli eventi in partitura, per poi interrompere la propria esecuzione. La durata della performance sarebbe dettata dallo spegnimento dell'ultimo *i statement* [4] nello *score*.

Nel caso in cui non siano presenti righe di codice nella sezione di *score*, invece, Csound rimane attivo per un tempo indefinito, in attesa dell'arrivo di messaggi inviati da sorgenti esterne. Questo potrebbe essere lo scenario tipico di una performance in tempo reale basata sull'interazione tra Csound e un sistema MIDI.

Conoscendo a priori l'intervallo di tempo in cui Csound deve rimanere in funzione e in attesa di messaggi, è possibile usare, all'interno dello *score*, un escamotage sintattico detto *f 0 statement*, che consiste nella creazione di una tabella-funzione vuota a un dato istante di tempo specificato in secondi. Ad esempio, la riga di codice

```
f 0 36000
```

[4] Si parla di ultimo *i statement* in riferimento al tempo, non alla posizione della riga di codice, in quanto una routine di Csound provvede a riordinare temporalmente gli eventi in partitura prima di compilare il codice.

mantiene attivo Csound per 36.000 s, ossia per 10 ore.

Spesso è necessario introdurre l'*f 0 statement* anche quando la durata della performance risulti indefinita o non sia nota a priori. Infatti non è sufficiente l'assenza di *i statements* nello *score*; tale sezione deve risultare completamente vuota, senza contenere neppure *f statements*, ossia definizioni di tabelle-funzione che spesso sono necessarie per il funzionamento degli oscillatori digitali.

11.8 Esempi

In questo paragrafo vengono presentati alcuni esempi chiarificatori sull'integrazione tra MIDI e Csound. Innanzi tutto si esplorano le possibilità più basilari per consentire a una catena MIDI di controllare in tempo reale gli strumenti Csound, partendo dall'approccio basato su *flag* (Paragrafi 11.8.1 e 11.8.2). Si introduce poi un semplice esempio di sintesi a partire dai dati contenuti in un file MIDI (Paragrafo 11.8.3). Si mostra, infine, un caso di lettura di messaggi MIDI, loro rielaborazione e invio a una catena posta a valle (Paragrafo 11.8.4).

I listati introdotti e commentati in questa sezione sono disponibili per lo scaricamento all'URL https://ludovico.lim.di.unimi.it/download/libri/midi/csound.zip.

11.8.1 Associazione di messaggi MIDI a un unico strumento

Nel Listato 11.2 viene mostrato un esempio di instradamento di tutti i messaggi MIDI in ingresso all'unico strumento presente, chiamato `instr 1`.

```
1  <CsoundSynthesizer>
2  <CsOptions>
3  -odac -+rtmidi=virtual -M0 --midi-key=4 --midi-velocity-amp=5
4  </CsOptions>
5  <CsInstruments>
6
7  sr = 44100
8  ksmps = 32
9  nchnls = 2
10 0dbfs = 1
11
12 massign 0, 1
13
14 instr 1
15 inote = p4
16 icps  = cpsmidinn(inote)
17 print icps
18 asig  oscil 0.5, icps, 1
19 outs  asig, asig
20 endin
21
22 </CsInstruments>
23 <CsScore>
24
25 f 0 60
26 f 1 0 16384 10 1
27
28 i1 0 1 60
29 e
30
31 </CsScore>
32 </CsoundSynthesizer>
```

Listato 11.2. Associazione di messaggi MIDI in ingresso a un unico strumento.

Alla riga 3 sono elencati i *flag*: `-odac` implica sintesi in tempo reale, `-+rtmidi=virtual` fa apparire e adotta come modulo di ingresso MIDI la tastiera virtuale di Csound, `-M0` riguarda la scelta di uno specifico ingresso (richiesta anche nel caso di tastiera virtuale), `--midi-key=4` fa sì che il valore del pitch sia disponibile all'interno dello strumento come valore nell'intervallo [0, 127] del *p-field* `p4` e `--midi-velocity-amp=5` rende disponibile il valore linearizzato della velocity in `p5` (argomento che non viene mai utilizzato nell'esempio).

La presenza di un unico *instrument block* porta Csound a funzionare automaticamente in *omni mode*, inoltrando a esso tutti i messaggi in ingresso. Questo renderebbe superflua la riga 12.

Alla riga 16 viene invocato l'*opcode* `cpsmidinn`, che provvede a convertire il valore del pitch da intero espresso nell'intervallo [0, 127] a frequenza in hertz, senza richiedere l'esplicito ricorso a formule. Il valore viene mostrato a console dall'*opcode* `print` di riga 17.

La riga 18 assegna alla variabile audio `asig` i valori calcolati dall'oscillatore digitale realizzato attraverso l'*opcode* `oscil`. Gli argomenti in ingresso sono, rispettivamente, l'ampiezza, il cui valore 0,5 va rapportato al valore 1 che corrisponde a 0dBfs (si veda la riga 10), la frequenza, che deriva dalla conversione del pitch MIDI, e la forma d'onda base, che rimanda alla tabella funzione 1 (riga 26) e – nella fattispecie – è la discretizzazione in 16.384 valori di un ciclo di sinusoide pura. Nelle ultime versioni di Csound è possibile omettere il terzo argomento di `oscil`, che, di default, è proprio un ciclo di sinusoide; nell'esempio in questione, tale semplificazione avrebbe permesso di non definire la tabella funzione 1.

Il cosiddetto *f 0 statement* alla riga 25, ossia la definizione di una tabella-funzione fantoccio, ha il solo scopo di fissare la durata complessiva della performance a 60 s. Si potrebbe erroneamente pensare che l'eliminazione delle righe 28-29 svuoterebbe lo *score*, consentendo quindi di evitare una definizione esplicita della durata della performance a riga 25. Purtroppo, quanto affermato nel Paragrafo 11.7 richiede che l'intera sezione `<CsScore>...</CsScore>` sia vuota, quindi nemmeno riga 26 potrebbe permanervi. Questa forte limitazione rende il più delle volte necessario il ricorso all'*f 0 statement*, specificando un intervallo temporale ragionevole.

Un'ultima considerazione riguarda la riga 28, in cui è presente un evento di partitura che va a sovrapporsi con l'esecuzione estemporanea dei messaggi MIDI in ingresso. L'utilizzo del *p-field* p4 all'interno dello strumento consente l'interoperabilità tanto con i messaggi MIDI in tempo reale quanto con gli eventi provenienti dallo *score*.

11.8.2 Associazione di canali e numeri di programma a strumenti differenti

Il Listato 11.3 esemplifica la possibilità di modificare l'associazione predefinita tra numeri di canale e/o *program number* e strumenti Csound.

Si ritiene opportuno richiamare il comportamento del linguaggio di programmazione quando rapportato agli ingressi MIDI: se esistono più strumenti nell'*orchestra* e non vengono modificate le impostazioni di default, il Canale 1 è associato allo strumento 1, il Canale 2 allo strumento 2 e via dicendo; ma, contemporaneamente, il *program number* 1 è associato allo strumento 1, il *program number* 2 allo strumento 2 e via dicendo.

In MIDI non esiste una pre-impostazione che associ di default il Canale 1 al numero di programma 1, il Canale 2 al numero di programma 2 e via dicendo; al contrario, se l'interfaccia del controller MIDI supporta l'invio di PROGRAM CHANGE, normalmente tutti i canali risultano inizialmente associati al primo timbro disponibile nell'elenco GM, ossia "Acoustic Grand Piano".

A un'analisi superficiale, sembrerebbero giungere messaggi MIDI recanti informazione ambigua. Ad esempio, un NOTE-ON sul Canale 5, cui è stato assegnato nel sistema MIDI il timbro di celesta (*program number* 9), verrebbe gestito dallo strumento 5 o dallo strumento 9? In realtà, tale ambiguità non sussiste se si prende in considerazione la comunicazione MIDI, in cui ogni messaggio reca solo una delle informazioni menzionate. I messaggi di NOTE-ON e NOTE-OFF sono di canale, per cui eventi-nota su canali distinti vengono gestiti dai rispettivi strumenti Csound sulla base del canale. Se poi giunge su un determinato canale un messaggio di PROGRAM CHANGE, a quel canale viene associato lo strumento determinato dalla corrispondenza tra *program number* e strumento. Si porteranno esempi di questi diversi scenari dopo aver commentato il listato.

```
1  <CsoundSynthesizer>
2  <CsOptions>
3  -odac -+rtmidi=portmidi -M0 --midi-key-cps=4
4  </CsOptions>
5  <CsInstruments>
6
7  sr = 44100
8  ksmps = 32
```

```
 9  nchnls = 2
10  0dbfs = 1
11
12  massign 0, 0
13  massign 1, 1
14  massign 2, 2
15  massign 3, 3
16  massign 4, 5
17  massign 10, 1
18
19  pgmassign 1, 1
20  pgmassign 2, 2
21  pgmassign 3, 3
22  pgmassign 4, 5
23  pgmassign 10, 2
24
25  instr 1
26  asig  oscil 0.5, p4, 1
27  print p4
28  outs  asig, asig
29  endin
30
31  instr 2
32  asig  oscil 0.5, p4, 2
33  outs  asig, asig
34  endin
35
36  instr 3
37  asig  oscil 0.5, p4, 3
38  outs  asig, asig
39  endin
40
41  instr 5
42  asig  oscil 0.5, p4, 4
43  outs  asig, asig
44  endin
45
46  </CsInstruments>
47  <CsScore>
48
49  f0 120
50  f1 0 16384 10 1                                          ; Sine
51  f2 0 16384 10 1 0.5 0.3 0.25 0.2 0.167 0.14 0.125 .111   ; Sawtooth
52  f3 0 16384 10 1 0   0.3 0    0.2 0     0.14 0     .111   ; Square
53  f4 0 16384 10 1 1   1   1    0.7 0.5   0.3  0.1          ; Pulse
54
55  </CsScore>
56  </CsoundSynthesizer>
```

Listato 11.3. Associazione di canali e numeri di programma a strumenti differenti.

Innanzi tutto si presti attenzione ai *flag* a riga 3. In questo caso si è scelto il modulo PortMIDI e l'ingresso 0, supponendo che si tratti di una porta MIDI di ingresso (fisica o virtuale) cui risulta collegata almeno una sorgente di messaggi MIDI, ad esempio un controller o un sequencer.

L'*orchestra* mette a disposizione quattro strumenti, volutamente numerati 1, 2, 3 e 5 (*sic!*). Per rendere evidenti le differenze a livello timbrico, le forme d'onda generate sono, rispettivamente, un'oscillazione sinusoidale pura, un dente di sega, un'onda quadra e un treno di impulsi. Questo dipende dalla tabella-funzione invocata da ciascuno strumento attraverso il terzo parametro di `oscil` in riferimento alle righe 49–52 dello *score*.

Si noti la semplificazione del corpo degli *instrument block* rispetto al Listato 11.2, dovuta al *flag* `-midi-key-cps=4`, che permette di riferirsi direttamente alla frequenza in hertz della nota attraverso il *p-field* `p4`.

Alle righe 12–17 vengono esplicitate le associazioni tra canali e strumenti (*opcode* `massign`), mentre, alle righe 19–23, quelle tra numeri di programma e strumenti (*opcode* `pgmassign`). Si osservi che riga 12 disabilita qualsiasi associazione tra canale e strumento che non venga esplicitata alle righe successive.

Si voglia ora analizzare il comportamento in tempo reale di Csound, ipotizzando di utilizzare una tastiera MIDI come unica sorgente. Il controller inizialmente associa tutti i canali al primo

timbro disponibile, ossia "Acoustic Grand Piano". In realtà, si tratta solo di un aspetto di interfaccia: poiché non vengono inviati messaggi espliciti di PROGRAM CHANGE, è evidente che tutti i canali siano associati al timbro GM di default; solo un'azione esplicita dell'utente sul selettore del timbro comporta l'invio di un PROGRAM CHANGE sul canale corrente. Questa considerazione è importante per comprendere la risposta di Csound ai diversi messaggi in arrivo.

Le prime note inviate sul Canale 1 vengono gestite dallo strumento 1 (sinusoidi pure). Il passaggio al Canale 2 porta le nuove note a essere eseguite dallo strumento 2 (dente di sega), la selezione del Canale 4 comporta l'invocazione dello strumento 5 (treno di impulsi), la scelta del Canale 10 provoca anch'essa l'uso dello strumento 1. La scelta di qualsiasi altro canale inibisce la sintesi.

Tornando ora al Canale 1 e scegliendo il timbro "Bright Acoustic Piano" (numero di programma 2), tutti i messaggi su tale canale vengono inoltrati allo strumento 2, come determinato da riga 20. In questo stato l'invio di messaggi sul Canale 1 o sul Canale 2 risulta indifferente dal punto di vista timbrico. L'eventuale arrivo di messaggi su altri canali non influenza il fatto che tale associazione canale-timbro-strumento rimanga in vigore fino a diversa indicazione. Ad esempio, la selezione del Canale 5, l'invio di nuovi NOTE-ON e NOTE-OFF e il ritorno al Canale 1 non cambierebbe lo stato delle cose.

Alla luce di quanto detto si può ipotizzare che l'associazione tra canali e strumenti, anche quando esplicitata nel codice dell'*orchestra*, sia una sorta di impostazione iniziale che viene sovrascritta da eventuali cambi di programma successivi: una volta effettuato il primo PROGRAM CHANGE sul canale, sarà sempre il numero di programma a determinare l'associazione tra gli eventi sul canale e lo strumento Csound. Peraltro, l'associazione canale-timbro-strumento viene determinata parzialmente in MIDI (canale-timbro) e parzialmente in Csound(dapprima canale-strumento, poi timbro-strumento). Questo comportamento è del tutto intuitivo se si considera la sequenza di messaggi in transito sul canale trasmissivo.

Tornando all'esempio in oggetto, solo l'utente può sapere di aver modificato il canale di trasmissione sul controller; dal punto di vista del destinatario, sta semplicemente giungendo una nuova sequenza di messaggi che, incidentalmente, si collocano tutti su un diverso canale. Questo fenomeno non si verificherebbe se la catena a monte contenesse un controller con aree di split, più controller o un sequencer.

11.8.3 Sintesi di un file MIDI

Il Listato 11.4 mostra una forma alquanto semplice di sintesi a partire dai dati provenienti da un file MIDI.

```
1   <CsoundSynthesizer>
2   <CsOptions>
3   -odac -+rtmidi=portmidi -F "abba_super_trouper.mid" --midi-key-cps=4
4   </CsOptions>
5   <CsInstruments>
6
7   sr = 44100
8   ksmps = 32
9   nchnls = 2
10  0dbfs = 1
11
12  massign 0, 1
13
14  instr 1
15  aout  oscil 0.05, p4
16  outs  aout, aout
17  endin
18
19  </CsInstruments>
20  <CsScore>
21
22  </CsScore>
23  </CsoundSynthesizer>
```

Listato 11.4. Sintesi di un file MIDI.

Un primo aspetto notevole del codice è l'adozione del *flag* `-F` seguito dalla stringa recante il nome del file MIDI, a riga 3.

Si osservi che tutti i canali vengono reindirizzati allo strumento 1 attraverso l'*opcode* `massign`, a riga 12.

Il timbro è un'oscillazione sinusoidale pura, la cui ampiezza viene arbitrariamente posta a 0,05 per tutte le note, in modo da minimizzare gli effetti di *clipping* dovuti alla sovrapposizione di più eventi sonori. Complicando il codice si sarebbe potuto intervenire con compressori e limitatori di dinamica, come, ad esempio, l'*opcode* `compress`. Altro aspetto certamente migliorabile riguarda il decadimento del suono: in questo listato la forma d'onda viene tagliata bruscamente nel momento in cui si verifica il `Note-Off`, senza introdurre un effetto di *fade out*. Tali aspetti, fondamentali per l'ottenimento di un buon risultato a livello di sintesi, sono stati volutamente tralasciati per mantenere l'attenzione del lettore sugli aspetti connessi al MIDI.

Lo *score* risulta completamente vuoto, il che mantiene Csound attivo indefinitamente.

11.8.4 Ricezione e invio di messaggi MIDI

In questo esempio si mostra un utilizzo di Csound che prescinde dalla sintesi del suono, sfruttandone unicamente le potenzialità di acquisizione e rielaborazione dei messaggi MIDI al fine di creare un'uscita MIDI opportuna.

Il Listato 11.5 mostra un semplice armonizzatore in cui le note in arrivo sul Canale 1 (affidate allo strumento 1) rappresentano la fondamentale delle triadi maggiori in uscita, mentre le note sul Canale 2 (assegnate allo strumento 2) originano triadi minori.

Le uscite non riguardano suoni di sintesi ma unicamente messaggi MIDI. Gli *opcode* coinvolti sono `notnum` per la lettura del pitch in ingresso, `veloc` per la lettura della velocity in ingresso e `midion` per inviare i messaggi MIDI a una catena a valle.

```
1  <CsoundSynthesizer>
2  <CsOptions>
3  -odac -+rtmidi=portmidi -M0 -Q1
4  </CsOptions>
5  <CsInstruments>
6
7  sr = 44100
8  kr = 4410
9  ksmps = 10
10 nchnls = 2
11
12 instr 1
13 ifund notnum
14 ivel  veloc
15 knote1 init ifund
16 knote2 init ifund + 4
17 knote3 init ifund + 7
18 midion 1, knote1, ivel
19 midion 1, knote2, ivel
20 midion 1, knote3, ivel
21 endin
22
23 instr 2
24 ifund notnum
25 ivel  veloc
26 knote1 init ifund
27 knote2 init ifund + 3
28 knote3 init ifund + 7
29 midion 1, knote1, ivel
30 midion 1, knote2, ivel
31 midion 1, knote3, ivel
32 endin
33
34 </CsInstruments>
35 <CsScore>
36 f0 3600
37 </CsScore>
38 </CsoundSynthesizer>
```

Listato 11.5. Ricezione e invio di messaggi MIDI.

Capitolo 12

Pure Data e MIDI

Pure Data, spesso abbreviato in Pd, è un ambiente di programmazione grafica in tempo reale (*real-time graphical programming environment*) per l'elaborazione di audio, video e grafica. Le attività di programmazione avvengono principalmente in modo visuale, ossia tramite la manipolazione visiva di elementi; all'evenienza è comunque possibile inserire porzioni più o meno estese di codice.

Pure Data, gratuito e open source, è uno standard *de facto* ampiamente utilizzato per performance di musica dal vivo, creazione di effetti sonori, composizione assistita dall'elaboratore, analisi audio, interfacciamento con sensori e via dicendo. I vari media sono gestiti all'interno dell'ambiente attraverso flussi di dati digitali.

Il nucleo originario di Pure Data è stato scritto da Miller S. Puckette, ora direttore associato del Center for Research in Computing and the Arts e professore di musica presso l'Università della California, San Diego. L'ambiente di programmazione, attualmente, si avvale del lavoro di molti sviluppatori. La comunità di utenti e programmatori partecipa attivamente allo sforzo collettivo creando i cosiddetti *externals* o *external libraries*, che ampliano in modo significativo le funzionalità dell'ambiente originario, rappresentando uno dei fattori di successo della piattaforma.

Pure Data è disponibile per diversi sistemi operativi, tra cui GNU/Linux, macOS, Windows, iOS e Android. Il sito web di riferimento è http://www.puredata.org/. Esistono due distribuzioni principali di Pure Data: la versione "vanilla" (standard) e la versione "extended" (arricchita di librerie e oggetti extra). In questo capitolo si farà riferimento esclusivamente alla versione "vanilla".

A Pure Data sono state dedicate numerose pubblicazioni tecniche e scientifiche. Tra queste si segnalano per autorevolezza, completezza e disponibilità in rete, i manuali [1], [5] e [31]. Altri testi di riferimento sono elencati alla pagina http://puredata.info/docs/BooksAboutPd/.

Le patch introdotte e commentate nel seguito del capitolo sono a disposizione del lettore all'URL https://ludovico.lim.di.unimi.it/download/libri/midi/puredata.zip.

12.1 Basi sintattiche

Il paradigma di programmazione visuale si basa, in generale, su blocchi funzionali (*boxes*) ed elementi connettivi (*arrows*).

In Pure Data questi elementi prendono il nome di oggetti (*objects*) e connessioni (*cords* o *wires*). Gli **oggetti** rappresentano processi e procedure che agiscono sui dati; essi possono manipolare messaggi, quali, ad esempio, numeri, simboli e liste, o segnali. Gli oggetti si interfacciano con il mondo esterno attraverso **connettori in ingresso** (*inlets*) e **in uscita** (*outlets*). La rappresentazione grafica degli oggetti in Pure Data fa uso di forme rettangolari.

Le **connessioni** permettono il flusso dei dati tra oggetti; allo scopo, esse uniscono gli *outlets* agli *inlets*. In Pure Data le connessioni si differenziano, logicamente e graficamente, a seconda del fatto che trasportino messaggi (linee rette sottili) o segnali (linee rette spesse). Nel caso di comunicazione MIDI, si ricadrà sempre nel primo scenario, ossia lo scambio di messaggi.

Per quanto concerne il flusso dei dati (*data flow*), l'interprete Pure Data esamina l'albero dei messaggi in verticale, seguendo una logica detta *depth-first*.

A livello di esecuzione, la strategia dipende dalla natura del flusso dei dati: per i messaggi il sistema è data-driven, mentre sui segnali viene effettuata una continua elaborazione su blocchi di campioni. Non c'è distinzione tra il momento della scrittura di codice e il momento della sua esecuzione: un programma in Pure Data è sempre in esecuzione, e qualsiasi azione ha effetto nell'istante stesso del suo completamento.

La descrizione grafica di un programma prende il nome di *patch*, e determina completamente le sue funzionalità.

12.2 Setup del sistema

Il primo passaggio per operare con MIDI all'interno di Pure Data è impostare correttamente le connessioni MIDI in ingresso e uscita dal menù "File > Preferenze > MIDI...".

Pure Data supporta simultaneamente più dispositivi MIDI, che possono essere aggiunti nella finestra di impostazioni attraverso il pulsante "Use Multiple Devices" (attualmente non localizzato in Italiano) e assegnandoli a qualsiasi porta risulti libera.

12.3 Ingressi MIDI

In questo paragrafo vengono trattati gli oggetti per la lettura di messaggi MIDI in ingresso messi a disposizione da Pure Data nella versione "vanilla". Essi lavorano di default in modalità *omni*, rispondendo a messaggi in arrivo su qualsiasi canale (e su qualsiasi porta, dato che Pure Data ha la possibilità di ascoltare più porte simultaneamente).

Nessuno degli oggetti con funzioni di lettura del MIDI in ingresso presenta *inlets*: l'unica forma di input ammessa sono i messaggi provenienti da un sistema MIDI posto a monte. Riguardo agli *outlets*, ogni oggetto ha delle proprie specificità, che verranno trattate nel seguito. In generale, l'ultimo degli *outlets* è riservato all'informazione di canale. Al fine di evitare ambiguità tra canali appartenenti a porte diverse, i canali sono numerati in modo consecutivo di porta in porta: l'intervallo [1, 16] si riferisce ai Canali da 1 a 16 della prima porta, l'intervallo [17, 32] ai Canali da 1 a 16 della seconda e via dicendo. In questo modo un unico valore numerico "aggregato" consente di rappresentare sia il canale sia la porta. Il problema non si pone per gli oggetti dedicati a messaggi di sistema, per i quali l'informazione di canale non è rilevante, quindi l'*outlet* dedicato emette solo il numero di porta: 1 per la prima porta, 2 per la seconda porta e via dicendo. Gli oggetti per i messaggi di canale supportano una sintassi con argomento opzionale che permette di selezionare lo specifico canale (e porta) di ascolto, disabilitando quindi l'*outlet* che emette questa informazione.

La Figura 12.1 è una rielaborazione della guida in linea di Pure Data per quanto riguarda gli oggetti trattati in dettaglio nel seguito.

12.3.1 Oggetto `midiin`

L'oggetto più basilare che gestisce messaggi MIDI in ingresso è `midiin`. Esso non presenta *inlets*, ma dispone di un primo *outlet* su cui viene emessa, byte per byte, la sequenza che compone il messaggio, e di un secondo *outlet* per il numero porta. L'informazione di canale per i *channel messages* può essere inferita dal *nibble* di ordine basso del primo byte sull'*outlet* di sinistra.

Nell'ipotesi di collegare il primo *outlet* a un oggetto di tipo numerico, un messaggio in ingresso quale NOTE-ON depositerebbe al suo interno i valori numerici dei byte via via ricevuti, quindi dapprima il byte di stato proseguendo in rapida successione con il primo e il secondo byte di dati, e sarebbe quest'ultimo a permanere al suo interno fino alla ricezione di un ulteriore messaggio

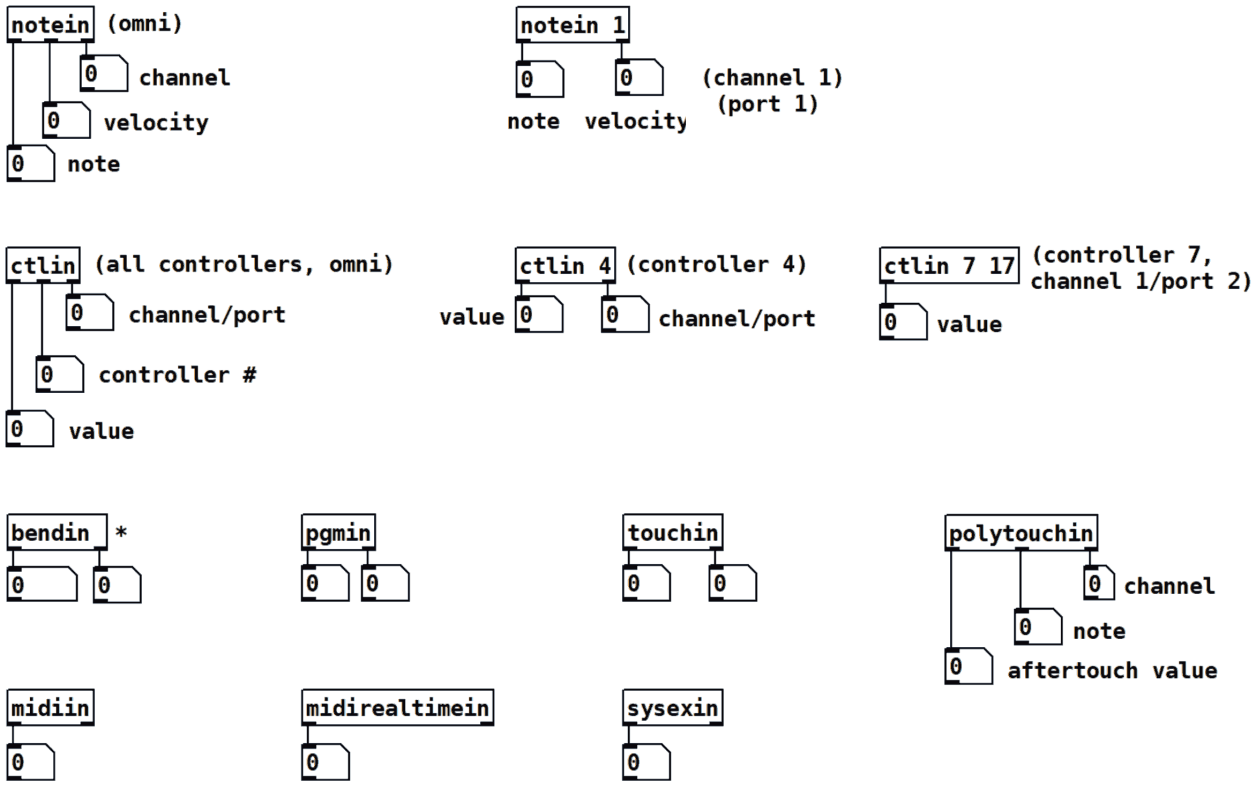


Figura 12.1. Rappresentazione grafica degli oggetti Pure Data per i messaggi MIDI in ingresso (rielaborazione della guida in linea di Pure Data).

MIDI. Questo comportamento può essere facilmente verificato collegando al primo *outlet* (anche) un oggetto `print`, in modo da mostrare su console i valori numerici via via ricevuti.

L'oggetto `midiin` non intercetta messaggi della famiglia *System Real Time*, cui è dedicato l'apposito oggetto `midirealtimein` trattato nel seguito.

12.3.2 Oggetto `notein`

L'oggetto `notein` è specializzato nella ricezione di messaggi NOTE-ON (e NOTE-OFF). I suoi *oulets* rendono disponibili, nell'ordine, il pitch, la velocity e il numero di canale.

Anche se di default l'oggetto opera in *omni mode*, specificando un argomento è possibile ascoltare selettivamente un dato canale. Ad esempio, scrivendo all'interno dell'oggetto

```
midiin 2
```

ci si pone in ascolto del Canale 2 della porta 1, scrivendo

```
midiin 17
```

si monitora il Canale 1 della porta 2. Poiché il numero di canale, in questa variante sintattica, è un'informazione in ingresso, gli *outlets* si riducono a due: pitch e velocity.

12.3.3 Oggetto `ctlin`

L'oggetto `ctlin` è dedicato ai messaggi CONTROL CHANGE rilevati in ingresso. Come *outlets* esso invia, nell'ordine, il valore del controller, il numero di controller e l'informazione aggregata su canale e porta.

Anche per questo oggetto c'è la possibilità di specificare alcuni argomenti. Limitandosi a esplicitare il primo, la ricezione dei messaggi CONTROL CHANGE viene filtrata rispetto a un dato *cc number*. Una sintassi quale

```
ctlin 7
```

continua a operare in *omni mode*, ma limitandosi ad ascoltare messaggi inerenti al volume di (qualsiasi) canale. In questo caso, scompare dalla lista degli *outlets* quello – ovviamente superfluo – relativo al numero di controller.

Esiste una seconda variante in cui è possibile indicare, oltre allo specifico *cc number*, un filtro sul numero di canale. Questa sintassi a due argomenti riduce gli *outlets* a un'unica uscita relativa al valore di un dato controller su un dato canale. Una sintassi quale

```
ctlin 7 19
```

restituisce in uscita il valore del volume associato al Canale 3 della porta 2.

12.3.4 Altri oggetti per messaggi *Channel Voice*

Pure Data è in grado di gestire altre tipologie di messaggi MIDI in ingresso appartenenti alla famiglia *Channel Voice*, e in particolare supporta gli oggetti:

- `bendin` per il messaggio PITCH BEND CHANGE;
- `pgmin` per il messaggio PROGRAM CHANGE;
- `touchin` per il messaggio CHANNEL PRESSURE, chiamato nelle specifiche *channel aftertouch*;
- `polytouchin` per il messaggio POLYPHONIC KEY PRESSURE, definito *polyphonic aftertouch*.

L'oggetto `polytouchin` è l'unico ad avere tre *outlets*, corrispondenti al valore di *aftertouch*, al pitch e all'informazione aggregata di canale e porta.

I restanti oggetti presentano solo due *outlets*, il primo relativo al valore e il secondo al numero di canale e di porta. Se questo è del tutto ovvio per PROGRAM CHANGE e CHANNEL PRESSURE, messaggi che – secondo le specifiche MIDI 1.0 – recano un solo byte di dati, potrebbe risultare sorprendente per PITCH BEND CHANGE, che, invece, si appoggia a due byte di dati. Va però ricordato che gli *outlets* di Pure Data non sono limitati a rappresentare singoli byte o – equivalentemente – valori numerici esprimibili su 8 bit. Il primo *outlet* di PITCH BEND CHANGE consente di rappresentare numeri nell'intervallo $[0, 2^{14}]$ che, in MIDI, richiederebbero 7 + 7 bit. In particolare, il valore emesso in posizione di riposo è 8192, all'estremo sinistro è 0 e all'estremo destro è 16383.

12.3.5 Oggetti per messaggi *System Real Time* e *System Exclusive*

L'oggetto `midiin` è sufficientemente generico per coprire la maggior parte dei messaggi MIDI in ingresso. La versione "vanilla" di Pure Data presenta, però, due oggetti specializzati, `midirealtimein` e `sysexin`, per gestire i messaggi delle famiglie *System Real Time* e *System Exclusive*.

La configurazione di *inlets* e *outlets* è la stessa di `midiin`: sono disponibili solo due uscite, la prima per la sequenza di byte del messaggio MIDI e la seconda per l'informazione di porta. Trattandosi di messaggi MIDI di sistema, non è necessario specificare il canale.

Nel caso di `midirealtimein` l'output del messaggio byte per byte si riduce al solo byte (di stato) che caratterizza ogni componente della famiglia *System Real Time*.

12.4 Uscite MIDI

In questo paragrafo vengono trattati gli oggetti per l'invio di messaggi MIDI messi a disposizione da Pure Data nella versione "vanilla".

Tutti gli oggetti inviano, di default, messaggi MIDI sul Canale 1 (ossia sul primo canale della prima porta), ma la loro sintassi supporta un argomento opzionale che consente di instradare i messaggi su un canale e su una porta diversi. Allo scopo, si sfrutta il consueto meccanismo del

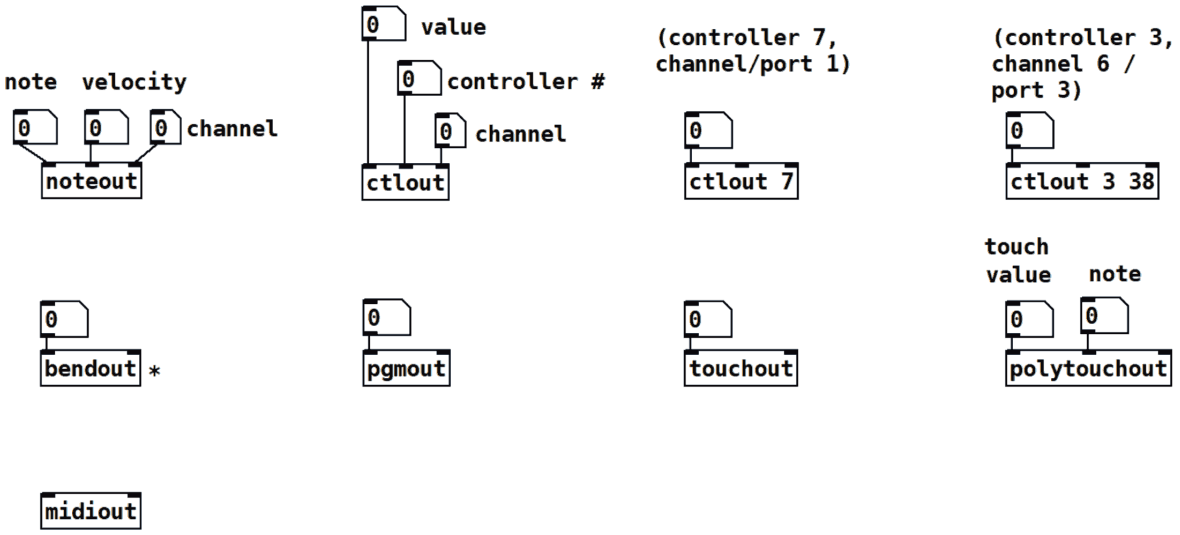


Figura 12.2. Rappresentazione grafica degli oggetti Pure Data per i messaggi MIDI in uscita (rielaborazione della guida in linea di Pure Data).

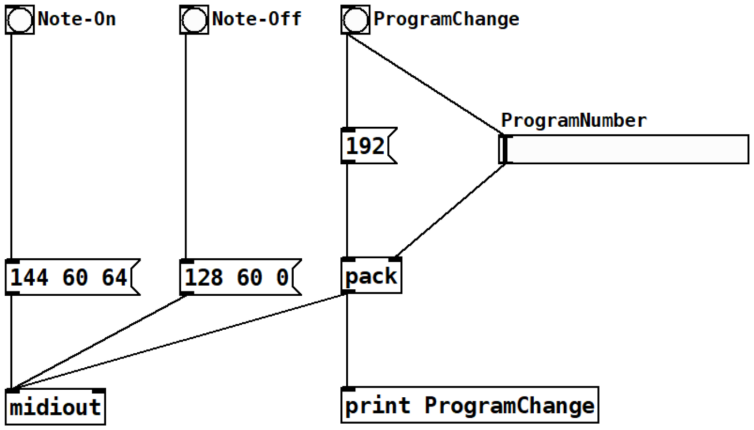


Figura 12.3. Esempio di utilizzo dell'oggetto `midiout`.

valore aggregato, ove 1 rappresenta il Canale 1 della porta 1, 2 il Canale 2 della porta 1, ..., 17 il Canale 1 della porta 2 e via dicendo.

In modo complementare a quanto visto per gli oggetti dedicati agli ingressi MIDI, quelli relativi alle uscite presentano solo *inlets* e nessun *outlet*.

La Figura 12.2, rielaborazione della guida in linea di Pure Data, riassume graficamente gli oggetti trattati nel seguito.

12.4.1 Oggetto `midiout`

L'oggetto `midiout` è la funzione più generale per inviare messaggi MIDI in uscita da Pure Data. Esso presenta due *inlets*, il primo per ricevere byte per byte i dati relativi al messaggio e il secondo per impostare il canale e la porta di uscita. Questo oggetto non supporta argomenti opzionali.

Un esempio di uso di `midiout` è mostrato nella Figura 12.3. Alla pressione dei rispettivi bang, la patch invia un messaggio di NOTE-ON con pitch e velocity predefinite (60 e 64), un messaggio di NOTE-OFF con pitch e velocity predefinite (60 e 0) e un messaggio di PROGRAM CHANGE con la possibilità di scegliere il timbro nell'intervallo [0, 127] attraverso lo slider orizzontale `ProgramNumber`.

12.4.2 Oggetti per messaggi *Channel Voice*

Gli oggetti per i messaggi *Channel Voice* in uscita sono complementari rispetto a quelli in ingresso. Essi presentano, sotto forma di *inlets*, le stesse connessioni – per numero, ordine e funzione – che erano *outlets* per le loro controparti dedicate ai messaggi in ingresso (Paragrafo 12.3).

In particolare:

- `noteout` si occupa dei messaggi NOTE-ON e NOTE-OFF ed è provvisto di tre *inlets*, rispettivamente, per pitch, velocity e numero di canale (e porta);
- `ctlout` invia messaggi CONTROL CHANGE ed è dotato di tre *inlets*, rispettivamente, per valore del controller, numero del controller e numero di canale (e porta). In analogia con `ctlin` è possibile esplicitare come primo argomento un numero di controller e, opzionalmente, anche un secondo argomento che rappresenta un numero di canale (e di porta) diverso da 1. Gli *inlets*, comunque, non vengono soppressi nel caso si specifichino gli argomenti; al contrario, eventuali valori in ingresso andrebbero a sovrascrivere tali parametri predefiniti;
- `bendout` si concentra sui messaggi PITCH BEND CHANGE, presentando due *inlets*, rispettivamente, per il valore di *pitch bend* e il numero di canale (e porta). A differenza di `bendin`, i valori attesi si collocano nell'intervallo [-8192, +8191], per cui la posizione centrale è 0 anziché 8192;
- `pgmout` è dedicato a messaggi PROGRAM CHANGE ed è provvisto di due *inlets*, rispettivamente, per il numero di programma e il numero di canale (e porta);
- `touchout` gestisce messaggi CHANNEL PRESSURE in uscita ed è dotato di due *inlets*, rispettivamente, per l'entità di variazione nella pressione e il numero di canale (e porta);
- `polytouchout` invia messaggi POLYPHONIC KEY PRESSURE e prevede tre *inlets*, rispettivamente, per l'entità di variazione di pressione, il pitch e il numero di canale (e porta).

12.5 Oggetti `makenote` e `stripnote`

In Pure Data è possibile costruire opportune strutture dati (tipicamente liste di valori numerici) da passare in ingresso a un oggetto `midiout` per generare messaggi MIDI NOTE-ON e NOTE-OFF. In alternativa, ci si può appoggiare all'oggetto `noteout` che presenta *inlets* differenziati per pitch, velocity e canale. In entrambi i casi la durata delle note deve essere gestita attraverso l'invio temporizzato di messaggi NOTE-OFF.

Per costruire agevolmente coppie di messaggi NOTE-ON e NOTE-OFF distanziati tra loro di n ms viene messo a disposizione l'oggetto `makenote`. I tre *inlets* riguardano, rispettivamente, pitch, velocity e scostamento temporale tra l'accensione (istantanea) e lo spegnimento (differito nel tempo) della nota. Non è prevista informazione sul canale e sulla porta; i due *outlets*, conseguentemente, riguardano il valore di pitch e velocity. Nell'ipotesi di collegare questo oggetto con *noteout*, il primo *outlet* di `makenote` va connesso al primo *inlet* di `noteout` e il secondo *outlet* di `makenote` al secondo *inlet* di `noteout`.

Obiettivo dell'oggetto `stripnote` è, invece, filtrare i messaggi di performance in ingresso eliminando dal flusso quelli riconosciuti come NOTE-OFF. L'oggetto accetta nel primo *inlet* valori strutturati come byte di dati dei messaggi NOTE-ON e NOTE-OFF. Esempi sono la lista in ingresso `60 69` per il messaggio NOTE-ON, che passerebbe inalterato, e la lista in ingresso `60 0` per il corrispettivo NOTE-OFF, che verrebbe eliminato.

Per i messaggi superstiti, il primo e il secondo valore ricevuti vengono emessi, rispettivamente, sul primo e sul secondo *outlet*. A titolo di esempio, tali *outlets* potrebbero essere connessi al primo e al secondo *inlet* dell'oggetto `noteout`.

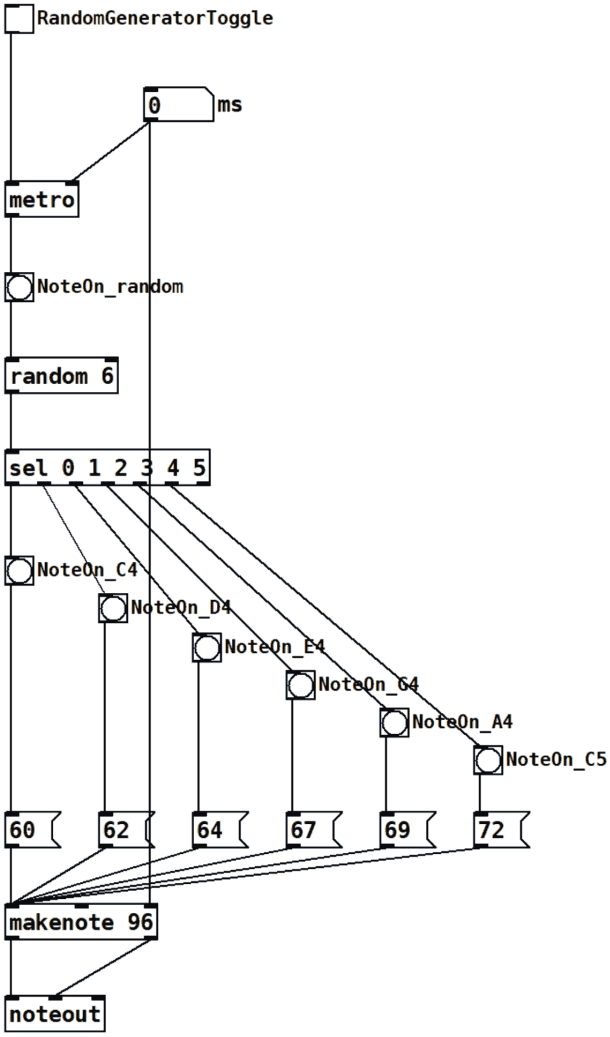


Figura 12.4. Accensione e spegnimento di note con `makenote`.

12.6 Esempi

12.6.1 Accensione e spegnimento di note

La Figura 12.4 mostra una patch in grado di generare una sequenza pseudo-casuale di pitch appartenenti a una scala pentatonica maggiore. La sequenza viene attivata e disattivata attraverso l'interruttore chiamato "RandomGeneratorToggle" che, a sua volta, avvia o ferma un metronomo che scandisce inizialmente il tempo di 120 bpm. È inoltre possibile far suonare singole note casuali agendo sul bang chiamato "NoteOn_random" o scegliere i pitch con i sottostanti bang.

I parametri delle singole note vengono combinati dall'oggetto `makenote`, cui viene imposto come argomento il valore 96 per la velocity (lasciando scollegato il secondo *inlet*) e passato il valore di scostamento temporale in millisecondi come terzo *inlet*, al fine di mantenerlo allineato con il metronomo corrente.

La produzione del suono viene delegata, infine, all'oggetto `noteout`.

Un'alternativa che non fa uso dell'oggetto `makenote` viene mostrata nella Figura 12.5. In questo caso, i messaggi di NOTE-OFF devono essere gestiti in modo esplicito, motivo per cui viene introdotto un oggetto `delay` controllato dal metronomo stesso. Per semplicità si è scelto di spegnere tutte le note potenzialmente attive in modo simultaneo, passando al primo *inlet* di `noteout` tutti i valori di pitch, e al secondo *inlet* il valore 0 per la velocity. Nel caso le note venissero accese attraverso i bang specifici, esse resterebbero in esecuzione fino all'invocazione esplicita del bang "AllNotesOff".

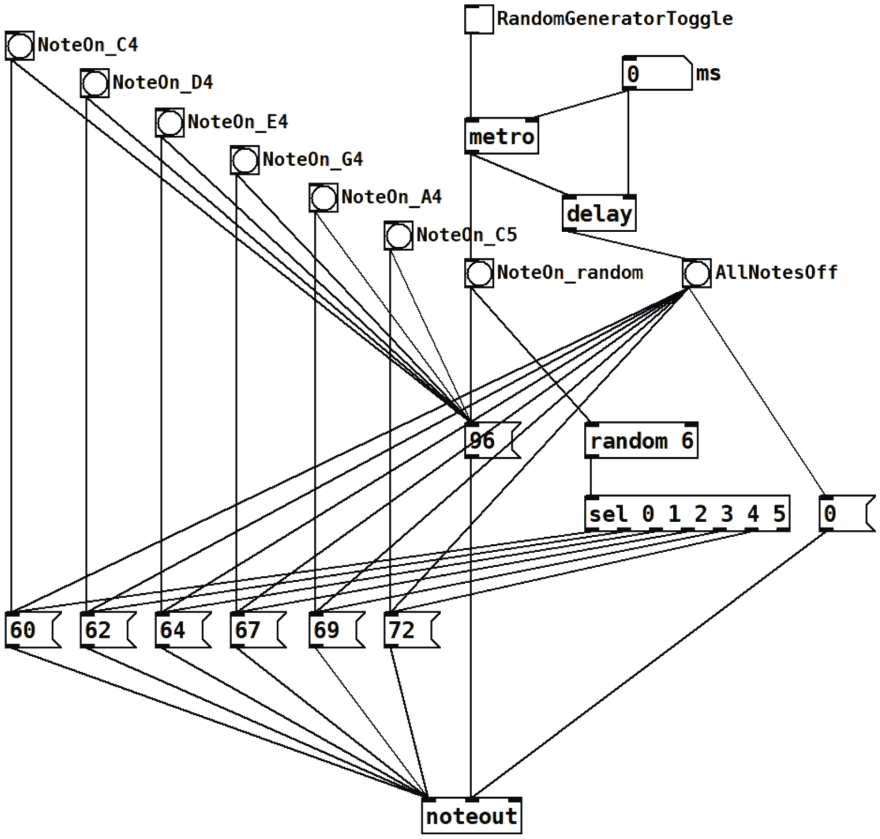


Figura 12.5. Accensione e spegnimento di note senza `makenote`.

12.6.2 Connessione di ingressi e uscite MIDI

La Figura 12.6 mostra una semplice patch che realizza un armonizzatore. Alla nota fondamentale, proveniente da ingresso MIDI, vengono sovrapposte altre due note la cui distanza dal basso in numero di semitoni è controllata dall'utente attraverso opportuni slider.

Gli slider sono configurati in modo da emettere valori nell'intervallo [0, 120] che, una volta sottoposti a divisione intera per 10, risultano interi e ricadono nell'intervallo [0, 12].

Il primo *outlet* dell'oggetto `notein` si propaga al primo *inlet* degli oggetti `noteout`, sommato in due casi ai valori in arrivo dagli slider. Il secondo e il terzo *outlet* vengono collegati direttamente al secondo e al terzo *inlet* degli oggetti `noteout`.

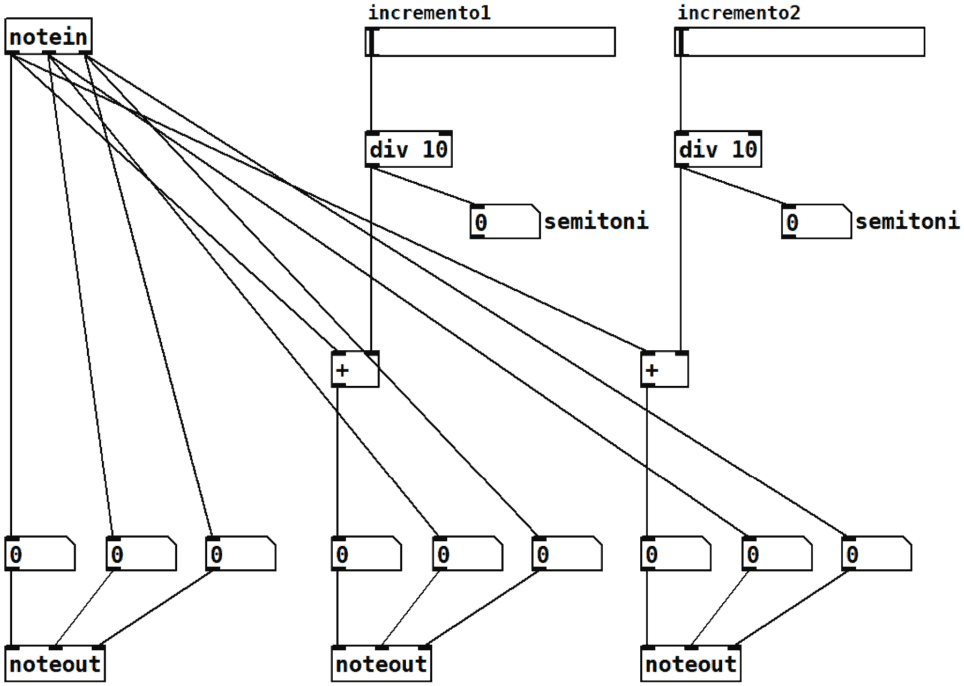


Figura 12.6. Lettura e invio di messaggi MIDI di performance.

Bibliografia

[1] AA. VV. Pure Data Floss manual. http://write.flossmanuals.net/pure-data/, 1991.

[2] Jim Aikin. *Csound Power! The Comprehensive Guide*. Cengage Learning Ptr, 2012.

[3] Stephen Barrass and Gregory Kramer. Using sonification. *Multimedia systems*, 7(1):23–31, 1999.

[4] Benjamin Belmudez, Sebastian Moeller, Blazej Lewcio, Alexander Raake, and Amir Mehmood. Audio and video channel impact on perceived audio-visual quality in different interactive contexts. In *2009 IEEE International Workshop on Multimedia Signal Processing*, pages 1–5, 2009.

[5] Francesco Bianchi. Inventare il suono con Pure Data. manuale introduttivo di musica elettronica. http://pditut.blogspot.com/, 2013.

[6] David Biedny. MIDI to the Macs. *MacUser*, pages 90–145, December 1985.

[7] Richard Charles Boulanger. *The Csound book: perspectives in software synthesis, sound design, signal processing, and programming*. MIT press, 2000.

[8] Allan D. Coop. Sonification, musification, and synthesis of absolute program music. In *The 22nd International Conference on Auditory Display (ICAD–2016)*, 2016.

[9] Roger B. Dannenberg. The interpretation of MIDI velocity. In *Proceedings of the 2006 International Computer Music Conference, November 6-11, 2006*, pages 193–196, 2006.

[10] Tuomas Eerola and Petri Toiviainen. *MIDI Toolbox: MATLAB tools for music research*. Department of Music, University of Jyväskylä, 2004.

[11] Tuomas Eerola and Petri Toiviainen. MIR in Matlab: The MIDI Toolbox. In *ISMIR*. Citeseer, 2004.

[12] Frederic Font and Giuseppe Bandiera. Freesound explorer: make music while discovering freesound! In *Web Audio Conference 2017*, 2017.

[13] Allen Forte. *The structure of atonal music*, volume 304. Yale University Press, 1973.

[14] Werner Goebl and Caroline Palmer. Temporal control and hand movement efficiency in skilled music performance. *PLOS ONE*, 8(1):1–10, 01 2013.

[15] Scot Gresham-Lancaster. Relationships of sonification to music and sound art. *AI & society*, 27(2):207–212, 2012.

[16] Corentin Gurtner. Applications of audio and MIDI API within a music notation editor. In *Web Audio Conference 2016*. Georgia Institute of Technology, 2016.

[17] Thomas Hermann, Andy Hunt, and John G. Neuhoff. *The sonification handbook*. Logos Verlag Berlin, 2011.

[18] IMA. *MIDI 1.0 Specification*. International MIDI Association (IMA), Sun Valley, CA, August 1983.

[19] Jari Kleimola and Oliver Larkin. Web audio modules. In *Proc. 12th Sound and Music Computing Conference*, 2015.

[20] Gregory Kramer. Auditory display: Sonification, audification, and auditory interfaces, 1993.

[21] Olivier Lartillot, Petri Toiviainen, and Tuomas Eerola. A MATLAB toolbox for music information retrieval. In *Data analysis, machine learning and applications*, pages 261–268. Springer, 2008.

[22] Victor Lazzarini, Steven Yi, Joachim Heintz, Øyvind Brandtsegg, Iain McCurdy, et al. *Csound: a sound and music computing system*. Springer, 2016.

[23] Paul D. Lehrman. MIDI 2.0: Promises and challenges. In *Music Encoding Conference 2020*, 2020.

[24] Paul D. Lehrman and Tim Tully. *MIDI for the Professional*. Amsco Publications, 1993.

[25] Luca Andrea Ludovico. The Web MIDI API in on-line applications for music education. In Luca Andrea Ludovico, editor, *Proceedings of the Ninth International Conference on Mobile, Hybrid, and On-line Learning (eLmL 2017)*, pages 72–77. IARIA XPS, 2017.

[26] Iain
McCurdy, Joachim Heintz, et al. Csound Floss manual. http://floss.booktype.pro/csound/, 2015.

[27] MMA. *MIDI DIN Electrical Specification*. MIDI Manufacturers Association (MMA), 1983.

[28] MMA. *The Complete MIDI 1.0 Detailed Specification*. MIDI Manufacturers Association (MMA), Los Angeles, CA, 1996.

[29] MMA. *Electrical Specification Update 1.1*. MIDI Manufacturers Association (MMA), 2014.

[30] MMA/AMEI. *MIDI 2.0 Specification*. Association of Musical Electronics Industry (AMEI) and MIDI Manufacturers Association (MMA), January 2020.

[31] Miller S. Puckette. Manual for pure data. http://puredata.info/docs/manuals/pd/, 2003.

[32] John Rahn. *Basic atonal theory*. Longman, 1980.

[33] Kaliappa Ravindran. Real-time synchronization of multimedia data streams in high speed networks. In *Proc. 1992 Workshop on Multimedia Information Systems, Tempe, Arizona*, pages 164–188, 1992.

[34] Dave Smith and Chet Wood. The 'USI', or Universal Synthesizer Interface. In *Audio Engineering Society Convention 70*. Audio Engineering Society, October 1981.

[35] Ralf Steinmetz. Human perception of jitter and media synchronization. *IEEE Journal on Selected Areas in Communications*, 14(1):61–72, 1996.

[36] Volker Straebel. *The sonification metaphor in instrumental music and sonification's romantic implications*. Georgia Institute of Technology, 2010.

[37] The MIDI Association. The history of MIDI. https://www.midi.org/midi-articles/the-history-of-midi, 2019.

[38] Barry Vercoe. *Csound. A Manual for the Audio Processing System and Supporting Programs with Tutorials*. Massachusetts Institute of Technology, 1986.

Nota su figure e tabelle

Dove non diversamente indicato, figure e tabelle si intendono prodotte dall'Autore.

Elenco fonti delle figure

Figura 1.1
Fonte: Columbia University Computer Music Center.
Permesso: richiesto e accordato da parte di Columbia University Computer Music Center.

Figura 1.2
Fonte: Moog Music Inc.
Permesso: richiesto e accordato da parte di Moog Music Inc.

Figura 1.3
Fonte: midi.org
Permesso: pubblico dominio.

Figura 1.4
Fonte: Sequential.
Permesso: richiesto e accordato da parte di Sequential.

Figura 1.5
Fonte: midi.org.
Permesso: pubblico dominio.

Figura 1.6a
Fonte: Roland.
Permesso: immagine pubblicamente disponibile sul sito Roland.
(https://www.roland.com/us/support/by_product/aerophone_go/product_images/).

Figura 1.6b
Fonte: AKAI.
Permesso: immagine pubblicamente disponibile sul sito AKAI. (https://www.akaipro.com/lpd8).

Figura 1.17
Fonte: Roland.
Permesso: immagine pubblicamente disponibile sul sito Roland.
(https://www.roland.com/us/support/by_product/um-one/product_images/).

Figura 4.1
Fonte: Yamaha.
Immagine di pubblico dominio: https://commons.wikimedia.org/wiki/File:Yamaha_ymf744b_v.jpg

Figura 4.2
Fonte: ROLI.
Permesso: richiesto e accordato da parte di ROLI.

MIDI

Una guida al protocollo, alle estensioni e alla programmazione

Il volume si pone come una guida allo studio del MIDI nelle sue molteplici sfaccettature, dall'originario protocollo di comunicazione digitale alla più recente versione 2.0. Altro tema centrale è lo sviluppo di applicazioni informatiche basate sul protocollo e rivolte a dispositivi MIDI, la cui trattazione include diversi paradigmi e linguaggi di programmazione. Il testo rappresenta un riferimento utile tanto per il principiante alle prime armi quanto per l'esperto desideroso di approfondire aspetti specifici della materia.

In copertina: Controller a tastiera e cavo MIDI. Fotografia dell'autore

ISBN 979-12-80325-11-2 (print)
ISBN 979-12-80325-28-0 (PDF)
ISBN 979-12-80325-32-7 (EPUB)
DOI 10.13130/milanoup.27