Università di Bologna
Facoltà di Ingegneria Seconda

Corso di Sistemi Multi-Agente L-S,
anno accademico 2009-10

Interactive Musical Agents

Studente:
Alessandro Montalti
Matr: 0000311979

Abstract

Questo approfondimento presenta l'approccio dei sistemi multi-agente applicato alla composizione ed all'interazione musicale tra agenti. Viene innanzitutto presentata l'architettura **MAMA**, che introduce i *musical acts* come ulteriore forma di comunicazione tra agenti, per poi analizzare il progetto **Andante**, un'infrastruttura open-source scritta in Java per la modellazione di applicazioni distribuite basate su *mobile agents* per la composizione e l'esecuzione di opere musicali. Infine viene presentata la proposta di un framework più generale realizzato sempre in Java e basato sulle due tecnologie introdotte in precedenza, per permettono l'esecuzione coordinata dei processi musicali da parte degli agenti presenti all'interno dello spazio virtuale.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

I compositori hanno sempre posto la loro attenzione ai risultati della scienza contemporanea per ottenere nuove forme con cui esprimere la propria arte. La musica occidentale stessa ha subito numerosi cambiamenti con l'introduzione della musica elettronica e l'utilizzo di nuovi strumenti. In sostanza, nelle ultime decadi, si è instaurata ed intensificata la relazione tra scienza e musica e, grazie a sistemi software all'avanguardia ed all'avvento pervasivo di internet, è stato rivoluzionato il modo di fare musica.

Ovviamente questo ambito risulta intrinsecamente multi-disciplinare, in quanto coinvolge aspetti che vanno dall'intelligenza artificiale (algoritmi compositivi incapsulati dagli agenti), ai sistemi concorrenti (vincoli real-time necessari per la performance), passando dai sistemi distribuiti e multi-agente.

In questo contesto si vuole dimostrare come i concetti avanzati propri della *programmazione ad agenti* possano fornire possibilità innovative per la composizione e l'esecuzione musicale. Utilizzando l'astrazione di agente per modellare un musicista umano, è possibile creare numerosissimi scenari a complessità crescente in cui agenti interagiscano tra loro o con musicisti umani posti all'esterno del sistema. A questo scopo sono stati analizzati tre sistemi software, di cui vengono presentati in dettaglio la struttura e l'implementazione: di ognuno di questi è inoltre presentata un'applicazione pratica, per valutarne le performance e l'output generato, dimostrando così come questi sistemi ottengano realmente risultati notevoli.

Nel capitolo 1 è analizzata l'*architettura MAMA*, un'infrastruttura software che modella una nuova forma di comunicazione tra gli agenti nel sistema, i *musical acts*. Oltre all'importanza rivolta alla modellazione della comunicazione tra gli agenti, è stata posta particolare attenzione alla rappresentazione interna della musica nel sistema.

Nel capitolo 2, invece, viene presentato *Andante*, un progetto open-source realizzato in Java per la modellazione di applicazioni distribuite basate sul concetto avanzato di *mobile musical agents*, con particolare attenzione all'interazione tra gli agenti ed alle performance del sistema.

Infine, nel capitolo 3 viene proposto un framework più generale che riprende alcune caratteristiche dei sistemi precedenti, ponendo però l'attenzione sul corretto incapsulamento delle operazioni di basso livello, con l'obiettivo di lasciare al programmatore unicamente il compito di definire gli aspetti di alto livello relativi alla composizione ed esecuzione musicale.

Ognuno di questi sistemi è stato quindi analizzato nell'ottica della teoria *Agents & Artifacts*, valutandone parallelismi e discrepanze con gli aspetti avanzati affrontati durante il corso.

1 L'architettura MAMA

In questo capitolo viene introdotta **MAMA**, un'architettura per *interactive musical agents*. L'obiettivo principale di questa ricerca è quello di fornire la descrizione di un'architettura generale per sistemi multi-agente musicali, focalizzandosi in particolare sugli aspetti di alto livello riguardanti l'interazione ed introducendo a questo proposito i *musical acts* come meta-informazioni scambiate tra gli agenti. Infine viene presentata la realizzazione di una versione semplificata di questa architettura, applicata ad un caso di studio noto, dando modo di analizzarne pregi e difetti.

1.1 Introduzione

Questo primo approccio alla *agent-based music* è influenzato dalla caratterizzazione di differenti forme di interazione tra agenti basate sulla nozione di **communicative act**, che a sua volta attinge alla nozione di *speech act* propria della semantica linguistica. Il motivo dell'utilizzo dei *communicative acts* per strutturare l'interazione tra agenti è che differenti approcci comunicativi (come, ad esempio, "agree" o "cancel", ecc...) richiedono differenti forme di interazione; accordarsi su un set di possibili "acts" permette il riuso di *communicative acts* composti e lo sviluppo di una piattaforma per l'analisi di queste interazioni, oltre che il supporto per l'interattività. Questo approccio rievoca in maniera naturale la nozione di *interoperabilità* degli agenti e la possibilità da parte di un agente di valutare le intenzioni di altri agenti durante la comunicazione.

Viene infatti sfruttata l'analogia tra la nozione di intenzionalità e quella applicata al contesto musicale, quando un musicista improvvisa e suona insieme ad altri musicisti; informalmente, questa forma di interazione musicale può essere descritta in termini di "conversazione" o "discussione" tra musicisti, validando l'approccio appena presentato. Nonostante questo però, il caso in questione presenta sostanziali differenze.

Innanzitutto, i musicisti tipicamente suonano tutti insieme contemporaneamente, invece che prendere la parola a turno come nei dialoghi in linguaggio naturale; l'architettura di supporto deve rispecchiarlo, in quanto l'output dei *musical statement* non può essere considerato uno step atomico poiché avviene in un contesto real-time in cui vengono monitorate simultaneamente le attività musicali di altri agenti. Inoltre, per lo sviluppo di un sistema stabile e completo, è necessario modellare più che l'interazione tra due agenti, come suggerirebbe l'approccio tipico della comunicazione. Infine, non è pre-

sente il concetto di "significato letterale" di un *musical statement*, per come inteso nel linguaggio naturale.

1.2 Musical Agents Architecture

Di seguito viene presentata una particolare soluzione al problema presentato in precedenza, l'architettura MAMA (Musical Acts - Musical Agents), concepita dal gruppo di ricerca della University of Edinburgh. Verrà descritta l'architettura e data una panoramica dei *musical acts*, per poi discutere alcuni problemi legati alla rappresentazione della musica per gli agenti.

L'obiettivo principale di questo lavoro è quello di creare una società di agenti che possano improvvisare musica insieme, interagendo con gli umani sia musicalmente che tramite interfacce di alto livello. Questi agenti utilizzeranno la teoria dei *musical acts*, analoga a quella degli *speech acts*, per dare informazione riguardo alla propria composizione musicale.

In particolare si focalizza l'attenzione sugli aspetti logici della composizione musicale, invece che quelli reattivi ed espressivi; inoltre l'aspetto chiave che viene modellato è l'interazione tra agenti, invece che le singole capacità compositive di ogni agente (trattate più approfonditamente nell'ambito dall'intelligenza artificiale).

1.2.1 Architettura

Creando un sistema di agenti musicali, si intende puntare ad un compromesso tra composizione assistita ed un sistema musicale interattivo. Gli agenti utilizzano rappresentazioni di alto livello di una porzione di musica, sia per dare informazioni riguardo alla loro interazione che per generare output musicale. Questo approccio è mirato a quelle situazioni in cui gli esecutori hanno un certo grado di autonomia, ma la loro esecuzione avviene all'interno di una struttura comune. Sono state teorizzate una serie di tipi di improvvisazione a libertà crescente. In particolare, di seguito vengono presentate le tipologie principali:

$\mathbf{I_1}$: gli esecutori "riempiono" certe porzioni che non sono già definite nello spartito;

$\mathbf{I_2}$: gli esecutori aggiungono note allo spartito, secondo una modalità attesa dal compositore;

$\mathbf{I_{3,5}}$: gli esecutori aggiungono intere battute, sezioni, ecc... allo spartito; se queste sono approvate o risultano attese dal compositore, sono considerate I_3, altrimenti I_5;

$I_{7,8}$: gli esecutori modificano sensibilmente lo spartito, inserendo re-armonizzazioni, alterazioni melodiche, ecc... ; la differenza tra I_7 e I_8 è che nel primo caso sono presenti ovvie connessioni tra lo spartito e l'interpretazione, mentre nel secondo non ve ne sono.

Questo sistema è inteso come un *musical middleware* per compositori e musicisti; un compositore, a questo scopo, deve innanzitutto:

- specificare alcuni aspetti della struttura di alto livello;
- fornire un **lessico**, cioè una serie di "frammenti" e vincoli sul loro utilizzo;
- fornire un set di librerie generali per il particolare stile di musica ricercato.

Il sistema ad agenti manterrà quindi una rappresentazione di alto livello basata sullo spartito. Analizzando lo stato corrente dello spartito e le azioni di altri agenti, oltre ovviamente al proprio stato interno, l'agente sarà in grado di costruirsi un piano di interazione con gli altri agenti. Questo potrebbe suggerire all'agente differenti ruoli da impersonare (come fare un assolo o tenere il tempo), e potrebbe portare ad una specifica serie di azioni, in base all'input ricevuto.

Dal piano di interazione, l'agente costruisce uno specifico set di *musical acts* da eseguire. Queste azioni comprendono l'introduzione, assestamento e sviluppo di diverse idee musicali e formano il substrato su cui è basata l'interazione.

Un *musical act* può definire unicamente una piccola frazione della superficie musicale e ci possono anche essere momenti in cui un agente non sta eseguendo niente. Ad esempio, quando un agente supporta un solista, l'esecuzione è relativamente priva di nuove idee in modo che la voce solista sia meglio udita. Questo significa che i *musical acts* sono un supplemento al processo di generazione, e che il carico della superficie musicale deve essere generato da altre sorgenti. Viene quindi utilizzata la combinazione di una rappresentazione di alto livello ed informazioni specifiche relative allo stile o al frammento relativo.

Oltre a generare output, un agente contemporaneamente ascolta l'output degli altri. Questo output è analizzato in base alle configurazioni conosciute dall'agente stesso. Da questa analisi l'agente estrae i *musical acts* che gli altri agenti hanno eseguito. Le attività degli altri agenti sono utilizzate nel processo di ragionamento ed inoltre determinano un feedback sulle azioni dell'agente stesso.

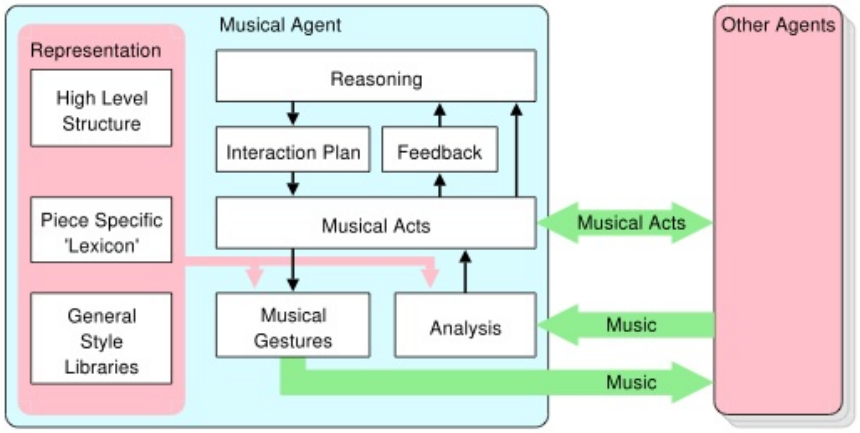

Figura 1: Musical agents system architecture overview

Come visibile in figura 1, ogni agente ha due canali di comunicazione con il resto del sistema: innanzitutto, c'è il canale musicale, dove le informazioni musicali, in una certa forma, sono trasferite tra gli agenti; oltre a questo, è presente un canale formale che permette la comunicazione diretta di *musical acts*. Questo può essere utilizzato per trasmettere l'equivalente del movimento di una mano nella "samba drumming", di un cenno come dimostrazione di apprezzamento da parte di un agente ad un altro o un altro segno tradizionale al di fuori della musica. Inoltre fornisce la possibilità di utilizzare un'interazione di alto livello tra agenti: un'interfaccia utente può trattare direttamente *musical acts*, dando la possibilità di organizzare la melodia suonata nell'insieme, senza dover definire dettagli di basso livello e mantenendo intatta l'autonomia individuale degli agenti. Infine, questo canale, dopo un'idonea processazione, fornisce una nuova fonte di dati in una modalità alternativa, come ad esempio un sensore in un'installazione interattiva.

Perché un agente possa ragionare in maniera efficiente, deve ricevere feedback riguardo la qualità delle sue azioni. Ci sono tre tipi di feedback a cui un agente può essere interessato:

- **Soddisfazione dei obiettivi interni**: devono essere forniti dal compositore o sorti durante l'esecuzione di una porzione di spartito;
- **Feedback diretto da altri partecipanti**: l'equivalente di un sorriso o un cenno da un'altro membro della band, può essere fornito anche da un partecipante umano tramite una determinata interfaccia;

- **Feedback d'interazione**: quando qualcuno sostiene un'idea proposta da un agente, questo è interpretato come un feedback positivo, ancora di più se qualcuno sviluppa un'idea o entra in discussione riguardo a questa.

1.2.2 Interazioni

Per supportare un ampio livello di interazioni, è stato utilizzato un sistema di *musical acts*, i quali risultano analoghi agli *speech acts* utilizzati nei comuni linguaggi di interazione tra agenti. Questo sistema è costruito su un set di operatori modali, presentati formalmente di seguito:

BM$_i$p (*BelMusic*): "L'agente i pensa che p faccia parte della rappresentazione condivisa di tutti gli agenti";

UM$_i$p (*UndefinedMusic*): "L'agente i non pensa che ci sia un valore comunemente accettato per la parte di rappresentazione realizzata da p";

IM$_i$p (*IntendsMusic*): "L'agente i pensa che p debba diventare una parte della rappresentazione condivisa";

PM$_i$p (*PersonalMusic*): "L'agente i considera p come parte della propria rappresentazione privata".

Viene inoltre introdotta l'idea di *compatibilità*: una proposizione musicale ψ è compatibile con ϕ ($Comp(\psi, \phi)$) se ψ è in qualche modo un'estensione di ϕ. La definizione di *compatibilità* in dettaglio può essere definita diversamente applicata a caratteristiche differenti, ma alcuni esempi possono essere l'estensione o l'aggiunta di un ornamento alla melodia, l'aggiunta di note ad un accordo, il cambiamento di una delle note dell'accordo (ovviamente compatibilmente con la teoria musicale).

Combinando i possibili stati del sistema ed i valori degli operatori modali, si può ottenere il seguente set di *musical acts*:

- **Inform**: se un agente sta esprimendo una porzione della struttura musicale che ritiene non ancora esplorata (cioè ancora priva di valori nella sua rappresentazione interna), allora semplicemente *informa* gli altri agenti del suo desiderio secondo cui modificare la rappresentazione;
- **Confirm**: se un agente A *informa* di una certa porzione di struttura, per la quale l'agente B non ha valori e ritiene soddisfacenti quelli proposti da A, allora B può *confermare* il suggerimento di A;

- **Disconfirm**: come sopra, ma nel caso in cui B non ritenga soddisfacenti i valori proposti da A, può *smentirli*. Questa operazione può concretizzarsi, ad esempio, con l'esecuzione da parte dell'agente B di una porzione non compatibile

- **Extend**: se A contiene un valore nella sua rappresentazione pubblica che desidera modificare in una maniera compatibile con il valore corrente, può *estendere* il valore in questione.

- **Alter**: se l'agente A contiene un valore nella sua rappresentazione pubblica che desidera modificare in maniera *non* compatibile con il valore corrente, può *alterare* il valore stesso.

Preconditions			**Act**
Agent i	$B_{i,j}$	Other	
$UM_i\phi \wedge IM_i\phi$	*	-	Inform-Music
$IM_i\phi$	$IM_j\phi$	-	Confirm-Music
$\neg IM_i\phi$	$IM_j\phi$	-	Disconfirm-Music
$BM_i\phi \wedge IM_i\psi$	*	$Comp(\phi, \psi)$	Extend-Music
$BM_i\phi \wedge IM_i\psi$	*	$\neg Comp(\phi, \psi)$	Alter-Music

Tabella 1: Musical Acts e pre-condizioni

Nella tabella 1 è presentato un riassunto delle interazioni presentate in precedenza e delle precondizioni necessarie.

1.2.3 Rappresentazione della musica

È stato tenuto conto di numerosi obiettivi durante la creazione del linguaggio per la rappresentazione della musica; innanzitutto dovrebbe essere *human-readable*, cioè interpretabile da un musicista umano, per permettere ad un compositore di interpretare il proprio spartito e quello degli altri partecipanti. Inoltre non dovrebbe essere legato alla tradizionale musica occidentale, ma dovrebbe potenzialmente rappresentare più tipologie possibili di musica. Infine dovrebbe supportare porzioni di spartito con una *struttura dinamica*, in modo che possano essere riordinate, estese o omesse a run-time.

La scomposizione gerarchica della musica non è un argomento nuovo ed è stata largamente utilizzata in molte rappresentazioni computazionali della musica. Nel sistema qui sviluppato, una porzione di musica è divisa in **sezioni** organizzate in una *struttura gerarchica* ad albero. Le sezioni-foglia

sono utilizzate per definire aspetti della musica da eseguire, mentre le sezioni lungo i rami sono utilizzati per l'ordinamento delle foglie. Gli attributi sono sono ereditati lungo la gerarchia, rendendo semplice per un compositore creare una serie di sezioni simili tra loro.

Le sezioni hanno numerosi controlli per gestire il loro *ordinamento*. In generale, le sezioni vengono suonate e processate secondo l'ordine in cui si presentano. È però possibile specificare differenti ordinamenti, in modo che una sezione possa utilizzare una regola decisionale per scegliere quale delle sue subordinate debba essere suonata dopo.

Allo scopo di scambiarsi informazioni sulla musica, gli agenti hanno la necessità di avere un modo formale ed univoco per indicare una determinata sezione dello spartito. In questo sistema, vengono utilizzati i *percorsi* come strutture, in modo che venga specificata l'ubicazione esatta della sezione utilizzata. Questo permette agli agenti di informarsi l'un l'altro su quale sezione stanno attualmente eseguendo e proporre modifiche ad una certa sezione del brano.

Molti spartiti integrano la loro notazione con un set di *istruzioni sull'esecuzione*. Per alcuni stili musicali, è essenziale la possibilità di rappresentare queste istruzioni in modo che un agente possa seguirle (ad esempio, in alcuni saggi accademici di riferimento le istruzioni costituiscono la maggior parte della struttura del pezzo).

Queste istruzioni sono rappresentate come *comportamenti* che un agente può mettere in atto, specificandoli come una combinazione di una condizione ed un'azione; l'agente è responsabile della gestione di certe variabili, grazie alla superficie musicale ed al proprio stato interno, le quali possono essere quindi utilizzate in un'espressione che rappresenti la condizione di un'istruzione. L'esecuzione di un'istruzione può avere un effetto strutturale, come ad esempio lo spostamento di una sezione al posto della successiva; può inoltre condizionare l'utilizzo di un determinato materiale musicale, come la risposta ad una chiamata; può infine modellare un output musicale già generato, andando ad esempio ad influenzare la dinamica o alcuni aspetti legati all'esecuzione.

1.2.4 Infrastruttura generale

Allo scopo di evitare di avere a che fare con problematiche real-time a più livelli, mantenendo però real-time l'output generato, sono state prese le seguenti decisioni e semplificazioni:

- è presente un *direttore* centralizzato nel sistema ad agenti (analogo al classico direttore d'orchestra), il quale è responsabile della raccolta dell'output di tutti gli agenti e la traduzione del risultato di questa operazione in dati MIDI o audio;

- una porzione di musica viene presentata in un certo momento, determinato dal direttore; gli agenti individuali producono il loro output in momenti determinati dalle battute;

- un agente produce il proprio output in un buffer, il quale è periodicamente inviato al direttore; il direttore richiederà una sezione in output che ricopra un determinato intervallo di tempo.

Questo significa che l'unica responsabilità temporale demandata ad un agente è assicurare il riempimento del buffer di esecuzione e di fornirne una porzione relativa ad un determinato intervallo temporale quando richiesto.

Un agente ha a disposizione le informazioni riguardo a quanto eseguito dagli altri agenti unicamente una porzione di tempo indietro: per questo motivo esiste un compromesso tra la reattività del sistema ad agenti e la latenza della rete (oltre ovviamente all'overhead introdotto per il meccanismo di message-passing).

Gli agenti sono situati all'interno di uno spazio virtuale e possono essere spostati all'interno tramite un'interfaccia grafica. La loro posizione all'interno dello spazio influisce sul suono che risulta nello spettro stereo e può essere utilizzata per dosare l'attenzione verso l'output degli altri agenti.

1.3 Caso di studio: "In C"

Il caso di studio scelto per l'applicazione di questo sistema è "In C" di Terry Riley, sia per il suo impatto all'inizio del movimento minimalista, sia per la sua idoneità alle tecniche di modellazione agent-based.

Lo spartito è costituito da una serie di brevi frammenti musicali, che vanno da mezza-battuta fino anche ad alcune battute. Ogni esecutore suona un frammento musicale a turno, ripetendo ognuno di questi indefinitamente prima di passare al successivo. Le istruzioni già introdotte in precedenza vengono fornite secondo le seguenti modalità:

- gli esecutori devono rimanere entro 2 o 3 pattern gli uni dagli altri;

- gli esecutori devono ascoltare attentamente la musica ed in alcuni momenti possono anche ritirarsi dall'esecuzione rimanendo in ascolto;

- un pattern può essere suonato all'unisono o canonicamente in ogni allineamento con sé stesso;
- un gruppo deve cercare di unire le melodie all'unisono almeno una o due volte durante l'intera esecuzione.

Per implementare queste specifiche, nel sistema ad agenti preso in considerazione, sono stati sviluppati i seguenti componenti:

Representation è stato implementato uno spartito (denominato appunto, *score*), che esponga in maniera dettagliata le note previste in ogni sezione. È stata quindi aggiunta una sezione di *silenzio* a cui gli agenti possono saltare per prendere una pausa, oltre ad una speciale sezione finale: arrivati alla fine dello spartito, il gruppo esegue dei *crescendo* e dei *diminuendo* diverse volte, mentre ogni esecutore via via si ritira ad ogni *diminuendo*;

Musical Acts quando un agente inizia una nuova sezione, invia un messaggio a tutti gli altri agenti per informarli della sezione che intende suonare. Questa operazione è effettuata attraverso il canale formale (come mostrato in figura 1);

Reasoning per seguire la direttiva secondo cui gli esecutori devono rimanere entro tre sezioni l'uno dall'altro, ogni agente traccia la progressione degli altri lungo il brano. La posizione relativa degli altri agenti nello spartito viene pesata (diviso per l'inverso della distanza al quadrato e per il volume) e sommata per ottenere un peso totale dell'avanzamento relativo del gruppo. Viene quindi utilizzata una funzione sigma (ad esempio, $S(t) = 1/(1 + e^{-t})$) per trasformare questo valore in una probabilità, utilizzata dagli agenti per discriminare il passaggio da una sezione alla successiva;

Analysis sono analizzate alcune caratteristiche di basso livello, come ad esempio le dinamiche, oltre alla pianificazione di particolarità più complesse come sincopi ed accordi. Queste operazioni vengono tutte inglobate nella fase di analisi, che può portare un agente a modificare il proprio output per aumentare o diminuire l'utilizzo di alcune delle particolarità d'esecuzione possibili.

In basso viene presentato un estratto del brano "In C": è possibile notare la strutturazione in *sezioni* dello spartito, oltre che alle linee guida generali che ogni agente dovrà seguire lungo la composizione.

```
( Piece                    InC
    ( Section         main
        ( ActiveBehaviour        offsets )
        ( ActiveBehaviour        breaks )
        ( ActiveBehaviour        followDynamics )
        ( ActiveBehaviour        varyDynamicRange )

        (Section            s1
            Derives:        baseSection
            Length:         3
            ( Channel Notes
                ( Gracenote: C, 0 )
                ( Note: E, 0.05, 0.9 )
                ( Gracenote: C, 1 )
                ( Note: E, 1.05, 0.9 )
                ( Gracenote: C, 2 )
                ( Note: E, 2.05, 0.9 )
            )
        )

        ( Section            s53a
            Length:          0.5
            Derives:         baseSection
            RepeatUntil:     everyoneArrived
            (Channel Notes
                ( Note: Bb, 0, 0.25 )
                ( Note: G, 0.25, 0.25 )
            )
        )
    )

    [ ... ]

)
```

1.3.1 Risultati

Sono stati effettuati numerosi test sull'infrastruttura del sistema, soprattutto per valutare quanti *musical agent* il sistema possa gestire contemporaneamente, senza interruzioni di suono. Inoltre, si è voluto testare se, introducendo un set ridotto di semplici direttive, sarebbe stato possibile ottenere un risultato musicalmente piacevole, che mostrasse i principali comportamenti attesi da un gruppo di musicisti umani.

Si è verificato che il sistema così definito può arrivare a gestire un numero utile di agenti, tanto che nel caso di studio considerato sono stati introdotti fino a 20 esecutori su un PC standard, utilizzando l'engine fornita da Java. È stato inoltre verificato che il sistema è sensibile all'esecuzione di altre applicazioni contemporaneamente sulla stessa macchina, anche se l'utilizzo della CPU tende a rimanere relativamente basso. È comunque consigliato l'utilizzo di un sintetizzatore esterno per migliorare il *timing* e la *qualità del suono* prodotti.

Per quanto riguarda invece il risultato musicale del sistema, si può affermare che l'output generato è generalmente piacevole, anche se questa caratteristica viene influenzata dal brano in questione e dal sistema ad agenti utilizzato.

1.4 Conclusioni

Analizzando il sistema descritto attraverso la teoria *Agents & Artefacts* presentata durante il corso si possono effettuare alcune importanti valutazioni.

Innanzitutto gli agenti musicali presenti nel sistema hanno un comportamento che possiamo definire *goal-governed*, cioè compongono musica con l'obiettivo esplicito di seguire le istruzioni stilistiche presenti nello spartito di riferimento, mediando la propria composizione con i feedback ricevuti dagli altri agenti nel sistema per ottenere un risultato musicalmente apprezzabile.

Viene quindi introdotta l'importanza della comunicazione tra agenti nella composizione, mediata dal **canale formale** rappresentato in figura 1: questo rappresenta in senso lato un *artefatto di co-ordinazione* tra gli agenti, permettendo lo scambio di informazioni accessorie alla composizione.

Lo **spartito**, invece, rappresenta l'artefatto fondamentale del sistema. Questo artefatto viene utilizzato dagli agenti a tutti e tre i livelli suggeriti dall'*activity theory*:

- a *livello di co-ordinazione*, per tenere traccia dell'avanzamento della composizione lungo il pezzo;

- a *livello di co-operazione*, permettendo agli agenti di modificare la struttura stessa del pezzo, definendo l'ordine di esecuzione delle sezioni;
- a *livello di co-costruzione*, incapsulando le direttive fornite dal programmatore o dal compositore, con l'obiettivo di controllare e direzionare la composizione.

Infine lo **spazio**, inteso come l'ambiente virtuale in cui risiedono gli agenti, assume un ruolo importante: infatti un agente, in base alla propria distanza dagli altri, risulta sensibile all'output prodotto, secondo una certa funzione di diffusione del suono. Questa caratteristica permette, come accennato in precedenza, di dosare i contributi di ciascun agente verso un altro.

Risulta ovvio che questa interpretazione del sistema MAMA non sempre calza alla perfezione, presentando nella pratica alcune discrepanze dalle definizioni formali presenti nella teoria A&A. Uno dei motivi principali di ciò si può ricercare nell'utilizzo di un paradigma di programmazione di più basso livello per la realizzazione del sistema. Rimane comunque interessante valutare questi aspetti, soprattutto nell'ottica di una futura nuova implementazione tramite linguaggi ad agenti più di alto livello che possano permettere al sistema di superare alcuni dei suoi limiti intrinsechi.

2 Il progetto Andante

In questo capitolo viene presentato lo sviluppo e l'implementazione del progetto **Andante**, il prototipo di un'infrastruttura open-source per la realizzazione di applicazioni distribuite per la composizione e l'esecuzione musicale basata sul concetto di *mobile musical agents*, introdotto in seguito. Verrà inoltre analizzata una semplice applicazione basate su questa tecnologia.

2.1 Introduzione

L'obiettivo di questo progetto è quello di applicare il concetto avanzato di *mobile agents* sviluppato dalla *computer science* ad un contesto musicale, per introdurre nuove forme di composizione, distribuzione e performace musicali.

Il progetto Andante offre un'infrastruttura software che permette la costruzione di applicazioni distribuite che utilizzino **mobile musical agents**. Utilizzando Andante, i programmatori possono implementare i propri agenti allo scopo di realizzare determinate applicazioni. Attualmente il progetto sta coinvolgendo sia compositori che ricercatori nella scrittura di nuovo materiale musicale, in modo da fornire una base di testing sufficientemente vasta per valutarne le performance e le future possibili estensioni.

Di seguito, verrà presentata in dettaglio l'infrastruttura e descritti alcuni dettagli implementativi non presenti nelle prime versioni del sistema, cercando di fornire un quadro generale dello stato attuale del progetto. Prima di tutto, però, verrà fornita una panoramica del concetto di *mobile agents* nel contesto dei sistemi distribuiti, come base per la successiva definizione applicata all'ambito musicale.

2.2 Il sistema ad agenti

2.2.1 Mobile Agents

Un *mobile agent* è un software che può interrompere la propria esecuzione in un host, migrare verso un altro host lungo la rete e riprendere lì la propria esecuzione.[1]. Questo programma enfatizza alcune caratteristiche proprie di un sistema ad agenti: in particolare, ogni agente ha l'*autonomia* per decidere quando e dove effettuare la migrazione e la *reattività* per affrontare i cambiamenti nell'environment dell'host in cui risiede.

Questo concetto porta numerosi vantaggi rispetto al tradizionale paradigma client/server. Una volta che un agente migra su un host esterno, i

[1] definizione presa dagli studi di Kotz e Gray, 1999.

comandi remoti vengono ora eseguiti localmente, quindi la latenza accumulata lungo la rete viene eliminata ed il traffico totale ridotto. Le applicazioni che coinvolgono grandi quantità di dati distribuiti possono quindi beneficiare dall'utilizzo dei *mobile agents*.

L'infrastruttura sviluppata si basa sul tool di sviluppo **Aglets Software Development Kit** (ASDK), sviluppato da Lange e Oshima nel 1998. *Aglets* è un sistema di agenti mobili, scritto in Java ed originalmente sviluppato da IBM [2]. Attualmente è un progetto open-source (disponibile all'indirizzo `http://aglets.sourceforge.net`) e mette a disposizione librerie ed applicazioni per implementare e distribuire *Java mobile agents*.

La scelta di *Aglets* è dovuta in particolar modo ai benefici introdotti dall'utilizzo di Java come middleware: Java fornisce infatti una buona soluzione in termini di indipendenza dalla piattaforma di utilizzo, requisito fondamentale per il contesto di riferimento. Il progetto Andante si rivolge appunto ad un serie di utenti, dal musicista al programmatore, che utilizzano spesso ambienti di lavoro differenti. In particolare, è richiesto che il sistema possa essere eseguito indistintamente su ambiente MacOS, Windows e Linux. A questo scopo la libreria *Java Swing* e le *Java Sound API* permettono rispettivamente la costruzione di interfacce grafiche e la gestione di periferiche audio assicurando la platform-indipendence.

Riassumendo, il modello utilizzato propone i seguenti vantaggi:

1. **riduzione del carico della rete e della latenza complessiva**: i sistemi distribuiti utilizzano spesso protocolli di comunicazione che implicano una grande quantità di messaggi scambiati attraverso la rete. Quando vengono utilizzati i *mobile agents*, invece, il programma migra nel host target e quindi lo scambio di messaggi avviene in locale. Ne consegue che il traffico nella rete è ridotto alla migrazione degli agenti, mentre viene completamente aggirata la latenza nelle comunicazioni.

2. **esecuzione autonoma ed asincrona**: dopo la migrazione, un agente mobile può diventare indipendente dall'applicazione che l'ha creata, permettendo un'esecuzione asincrona ed autonoma. Le applicazioni che si appoggiano su deboli connessioni possono trarre beneficio da questo perché non è necessario mantenere una connessione aperta tra le parti.

3. **adattività dinamica**: gli agenti mobili possono ricevere informazioni sull'ambiente di esecuzione e reagire ai cambiamenti che avvengono localmente.

[2] maggiori informazioni all'indirizzo `http://www.trl.ibm.com/aglets`.

2.2.2 Mobile Musical Agents

Si definisce *mobile musical agent* (d'ora in poi, solo "agente") un agente mobile che partecipi ad un processo musicale. In particolare, l'agente in questione deve svolgere una o più tra le seguenti attività.

- **Incapsulare un algoritmo**: come un software generico, un agente può implementare un algoritmo o meglio, la composizione di più algoritmi. Questi algoritmi potrebbero inoltre richiedere dati di input che possono essere contenuti nell'agente stesso o generati su necessità, permettendo all'agente di produrre musica autonomamente.

- **Interagire e scambiare informazioni con altri agenti**: come nel contesto reale in cui i musicisti suonano insieme sul palco, alcuni agenti possono interagire tra loro scambiandosi informazioni musicali.

- **Interagire con musicisti reali**: un agente può ricevere comandi o dati musicali da un musicista reale. I comandi possono essere sia semplici come il suonare una serie di note su una tastiera MIDI da parte di un musicista, o più complessi come i nuovi parametri necessari per un algoritmo. Inoltre, un agente può ricevere l'audio da uno strumento acustico e processare o riprodurre questo suono.

- **Reagire a sensori**: gli agenti possono ricevere comandi da altri programmi software che non sono un agente. Questi comandi possono essere determinati da un sensore, in modo che l'agente possa reagire ad eventi nel mondo fisico, reale, come ad esempio i movimenti di una ballerina sul palco o le attività del pubblico in un museo.

- **Migrare**: il processo di migrazione può essere causato da una delle seguenti azioni. In particolare, un agente decide di migrare:

  - stocasticamente o deterministicamente, in base ad un algoritmo;
  - basandosi sull'interazione con altri agenti;
  - basandosi con l'interazione con musicisti;
  - reagendo a sensori.

  Un agente che effettua una migrazione, riprende la propria esecuzione una volta giunto a destinazione: questa può essere sia un host situato nella stessa stanza, sia uno in un'altra stanza, città o paese.

2.2.3 Overview sui sistemi musicali

Con il modello ad agenti appena descritto, è possibile implementare numerosi sistemi musicali; di seguito ne vengono presentate alcune macro-tipologie.

1. *Melodie stocastiche*: in questo sistema, gli agenti incapsulano un algoritmo stocastico che genera una melodia. Un componente del sistema o uno degli agenti può lavorare come metronomo, fornendo agli altri agenti il timing corretto. Il risultato dell'esecuzione di diversi agenti di questo tipo sullo stesso host può suonare come una musica stocastica sincronizzata.

2. *Esecuzione distribuita*: ogni musicista umano è rappresentato da uno o più agenti. Ogni agente riceve la musica suonata dal musicista che rappresenta e la riproduce in real-time sulla macchina in cui risiede. In questo sistema, un musicista può essere virtualmente presente su più di un palco contemporaneamente. Questi agenti possono essere utilizzati per dirigere un'esecuzione distribuita: ogni musicista in un luogo differente riceve nel proprio computer gli agenti che rappresentano altri musicisti e manda i propri agenti nei computer degli altri musicisti.

3. *Musica collaborativa*: in sistemi come DASE[3], gli utenti interagiscono e si scambiano file audio attraverso la rete per comporre un brano musicale collaborativo. Un sistema ad agenti musicali mobili può utilizzare la stessa idea: in questo caso, però, sarebbero gli utenti ad implementare ed inviare i propri agenti musicali autonomi, che interagirebbero tra loro.

4. *Sistema musicale interattivo*: l'unione tra agenti musicali mobili e sistemi musicali interattivi è naturale, considerando che un agente può ricevere informazioni musicali e replicare.

5. *Musica distribuita*: consideriamo un museo o la sala di una mostra, fornita di diversi computer connessi in una rete wireless. Ogni computer potrebbe avere a disposizione sensori di movimento e ospitare qualche agente. Gli agenti potrebbero comunicare tra loro e suonare un brano di musica distribuita in maniera sincronizzata. Un agente specifico potrebbe ricevere informazioni dai sensori di movimento allo scopo di seguire una persona che cammina lungo la stanza, utilizzando la propria abilità di migrare). L'ascoltatore avrebbe la sensazione che

[3]ulteriori informazioni all'indirizzo `http://www.soundbyte.org`.

la musica lo stia seguendo. Un'altra parte di musica, d'altro canto, potrebbe allontanarsi dall'ascoltatore, migrando in computer lontano da dove si trova la persona. Inoltre, i suoni generati dal pubblico potrebbero essere dinamicamente incorporati nell'ambiente musicale creato da questo sistema.

2.3 Andante

Questa sezione descrive l'infrastruttura del sistema, discutendo le tecnologie impiegate, l'architettura ed i dettagli implementativi.

2.3.1 Tecnologie utilizzate

L'intero sistema è scritto in **Java**, per le seguenti ragioni, già introdotte in precedenza. Innanzitutto la *platform-indipendence*, assicurata dall'engine fornita da Java: ci si aspetta che i potenziali utenti utilizzino un ambiente hardware e software differente (in generale, per i programmatori Linux, per i compositori MacOS e per i musicisti Windows). Inoltre Java fornisce una libreria, *Java Swing*, per la costruzione di solide interfacce grafiche e delle primitive di sistema, le *Java Sound API*[4], che permettono la gestione di file multimediali in maniera molto intuitiva (anche se non soddisfano completamente tutti i bisogni, presenti e futuri, del sistema).

La generazione del suono è attualmente basata sulle classi MIDI fornite dalle *Java Sound API*, quindi sul **protocollo MIDI** stesso. Nonostante questo, si è cercato di evitare troppa influenza di questo protocollo a causa dei suoi limiti nello sviluppo di applicazioni musicali sofisticate.

Benché finora sia stato utilizzato solo Java, è preciso obiettivo quello di permettere che parti del sistema interagiscano con componenti scritti in altri linguaggi di programmazione. La ragione è quella di poter utilizzare, in un futuro, altre tecnologie per la generazione del suono, come ad esempio l'ambiente MAX/MSP, già testato in numerosi esperimenti.

Inoltre è stato utilizzato come middleware **CORBA**, che permette a programmi scritti in linguaggi di programmazione differenti ed in esecuzioni su sistemi operativi differenti, di comunicare. Tutte le comunicazioni tra i componenti di questo sistema sono realizzate tramite *CORBA*. Questo permette di integrare l'infrastruttura di Andante con sistemi come *CSound*, scritto in C, o *Siren*, scritto in SmallTalk.

[4]disponibili all'indirizzo `http://java.sun.com/products/java-media/sound`.

Come già affrontato in precedenza, l'infrastruttura generale è costruita su **Aglets Software Development Kit**, un progetto open-source realizzato in Java e originariamente ideato da IBM, che mette a disposizione le librerie e le applicazioni necessarie per implementare e gestire facilmente agenti mobili.

2.3.2 Architettura

Un agente esegue le proprie azioni all'interno di un ambiente di rete eterogeneo. Sui computer presenti in rete deve essere in esecuzione un software host chiamato *Stage*: questo software è un componente dell'architettura e rappresenta il luogo dove gli agenti si incontrano ed interagiscono, analogicamente a quanto accade con i musicisti su un palco.

Stage permette agli agenti di eseguire le proprie azioni: in particolare, per produrre un suono, un agente deve utilizzare il servizio fornito da *Stage*, il quale utilizza un altro componente del sistema, *Audio Device*.

I tre componenti introdotti finora (l'Agente, Stage e Audio Device), sono mostrati in figura 2.

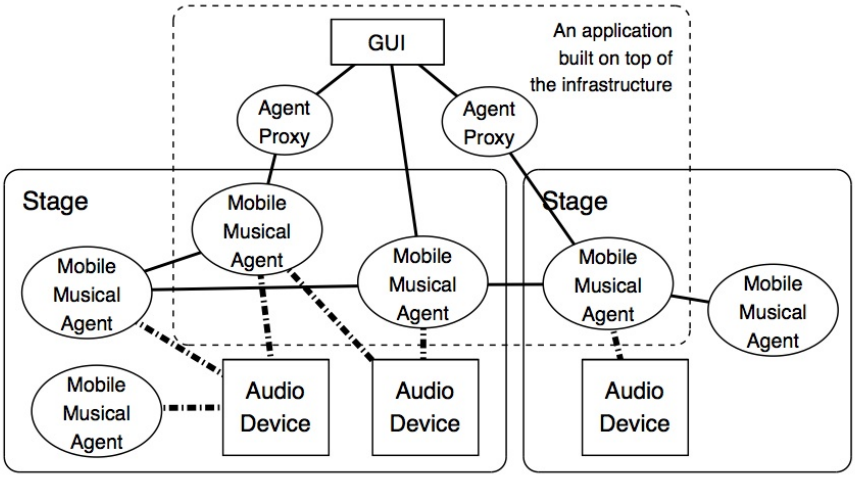

Figura 2: Andante architecture overview

Un altro componente, necessario per il corretto utilizzo dell'applicazione, è la *GUI*. Non rappresenta un componente strettamente necessario dell'architettura, ma è il principale componente delle applicazioni realizzato su questa infrastruttura e gioca l'importante ruolo di supportare l'interazione umana con il sistema.

L'ultimo componente dell'architettura è l'*Agent Proxy*, che gioca un ruolo fondamentale: assicura agli agenti la trasparenza del luogo o host in cui risiedono. Quando un agente decide di migrare, questo informa il proxy della propria nuova posizione, il quale poi è responsabile della comunicazione tra l'agente e la GUI. Quest'ultima può comunque decidere di comunicare direttamente con gli agente o scegliere il proxy come tramite per uno o più agenti.

2.3.3 Implementazione

In figura 3 è mostrato un diagramma UML semplificato delle classi Java già introdotte nella sezione precedente.

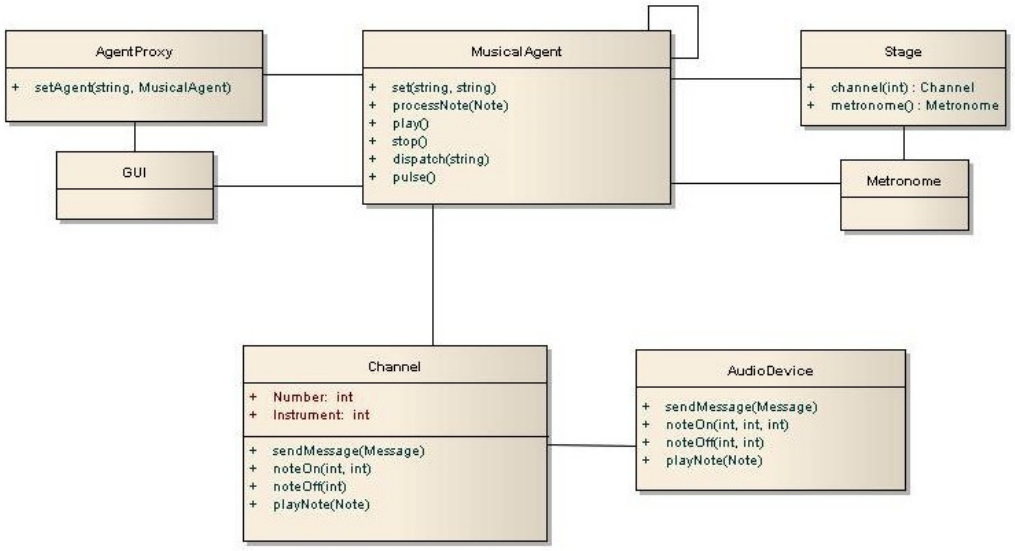

Figura 3: Andante class diagram

La classe `MusicalAgent` rappresenta l'agente musicale mobile, mentre la classe `Stage` è responsabile dell'hosting degli agenti su una determinata macchina. Tutte le istanze delle classi mostrate in figura si registrano al *CORBA Naming Service* (un servizio ovviamente centralizzato), in modo che possano agilmente conoscere le posizioni di ogni altra istanza nel sistema distribuito.

Entrambe le classi sono implementate utilizzando *Aglets Software Development Kit*, per cui è imposto l'utilizzo di Java come linguaggio di programmazione. Lo `Stage` offre vari servizi agli agenti: i principali vengono descritti di seguito.

- `channel`: fornisce un canale di comunicazione attraverso il quale i messaggi di richiesta di generazione del suono sono inviati, in maniera similare al protocollo MIDI (ma senza limitarsi a questo). Ogni `Stage` ha vari canali disponibili, ognuno dei quali permette differenti settaggi (ad esempio, il timbro utilizzato per suonare le note).

- `metronome`: fornisce un oggetto che agisce come un metronomo. Questo oggetto riceve le richieste di registrazione dagli agenti e, a quelli registrati, invia un messaggio di `pulse` ad intervalli regolari. L'intervallo di tempo tra due pulsazioni è determinato dalle proprietà del metronomo e, teoricamente, tutti gli agenti registrati ricevono il messaggio di `pulse` nel medesimo istante.

Per implementare una nuova tipologia di agente, l'utente dell'infrastruttura deve implementare i messaggi della classe `MusicalAgent` (alcuni hanno già un'implementazione generale). I messaggi più importanti sono i seguenti:

- `play`: comanda all'agente di iniziare o riprendere la sua esecuzione;

- `stop`: comanda all'agente di interrompere la sua esecuzione;

- `set`: impone alla proprietà `propName` il valore `propVal`;

- `pulse`: se l'agente è registrato ad un metronomo, riceverà questo messaggio ad intervalli regolari, come rappresentazione dell'impulso ad intervalli regolari di un tempo fissato. Si suppone che un agente effettui un'azione quando questo viene ricevuto;

- `dispatch`: comanda all'agente di migrare verso la destinazione specificata in `address`.

2.3.4 A simple scenario

Di seguito viene presentata l'implementazione di un semplice agente nell'infrastruttura Andante.

```
public class RandomMelodyAgent extends MobileMusicalAgent {

    boolean _play = false;
    short [] _cMaj = {60,62,64,65,67,69,71};
    java.util.Random _rand;
    Channel _channel;
```

```
    public void init() {
        _rand = new java.util.Random();
        _channel = _stage.channel(1);
        play();
    }

    public void play() {
        _play = true;
        int pitch, intensity, duration;

        while (_play) {
            pitch = _cMaj[ _rand.nextInt( _cMaj.length)];
            intensity = _rand.nextInt(128);
            duration = _rand.nextInt(1000);
            _channel.noteOn(pitch, intensity);

            try {
                Thread.sleep(duration);
            } catch (InterruptedException ie) {}
            _channel.noteOff(pitch);
        }
    }

    public void stop() {
        play = false;
    }
}
```

Questo agente suonerà una melodia casuale suonata in real-time. Di seguito viene presentato un semplice scenario di utilizzo del `RandomMelodyAgent`.

Implementazione: l'implementazione reale dell'agente in questione è stata già fornita. È necessaria ora un'istanza di questo tipo di agente, che chiameremo `rmAgent`. Questa istanza rappresenta un singolo agente.

Invio dell'agente: una volta creato, `rmAgent` deve essere inviato ad un'istanza di `Stage`.

Procedura di arrivo: quando `rmAgent` arriva allo stage di riferimento, l'infrastruttura si prende la responsabilità di eseguire un certo numero di azioni. Prima di tutto, il campo `_stage` interno all'agente comincia a rappresentare lo stage in cui risiede l'agente. A questo punto, viene chiamato il metodo `init()` dell'agente `rmAgent`. Infine, l'agente prende possesso di un canale dello stage, rappresentato d'ora in avanti dal campo interno `_channel`, e richiama il proprio metodo `play()`.

Esecuzione: come risultato del metodo `play()`, `rmAgent` inizia la sua esecuzione ed utilizza i metodi forniti da `_channel` per suonare note casuali in scala di C+ (do maggiore).

Controllo dell'agente: non è stato preso in considerazione in questo caso, ma è possibile implementare un'interfaccia grafica per richiamare i metodi di `rmAgent`. In questo caso, potrebbe essere utilizzata per l'invio del messaggio di `stop()` all'agente o per comandargli il trasferimento verso un altro stage.

2.4 Caso di studio: "NoiseWeaver"

Per dimostrare concretamente la solidità e l'utilità dell'infrastruttura di Andante, sono stati sviluppate numerose applicazioni pratiche. In questa sezione verrà analizzata **NoiseWeaver**, un'applicazione per generare ed eseguire musica stocastica in real-time.

Questa applicazione utilizza un solo tipo di agente: il `NoiseAgent`, il quale genera una singola melodia in real-time. Nella melodia generata, alcuni parametri vengono determinati stocasticamente, come ad esempio il tono o la durata di una nota. I generatori di questi valori vengono chiamati `noise`, perche simulano le frequenze che ricorrono nello spettro $1/f^{\beta}$. In altre parole, quello che viene chiamato `noise` è una sequenza di numeri generata tramite un algoritmo stocastico, che verrà poi mappata sui parametri musicali di una melodia[5].

L'agente, inoltre, deve effettuare la registrazione al metronomo dello stage in cui esegue la propria performance, in modo tale che tutti gli agenti siano sincronizzati.

[5]per ulteriori informazioni sull'algoritmo utilizzato in questa applicazione, si fa riferimento all'*algoritmo frattale* introdotto da Roads in "The Computer Music Tutorial" (MIT Press, 1996).

L'applicazione *NoiseWeaver* fornisce un'interfaccia grafica lato utente per il controllo degli agenti all'interno dello stage. Innanzitutto, tramite un'apposita schermata, è necessario scegliere, tra quelli disponibili, lo stage di riferimento. A questo punto verrà proposta una schermata simile a quella in figura 4, che permette all'utente di cambiare la frequenza del metronomo e le proprietà dell'agente, anche durante la generazione della melodia.

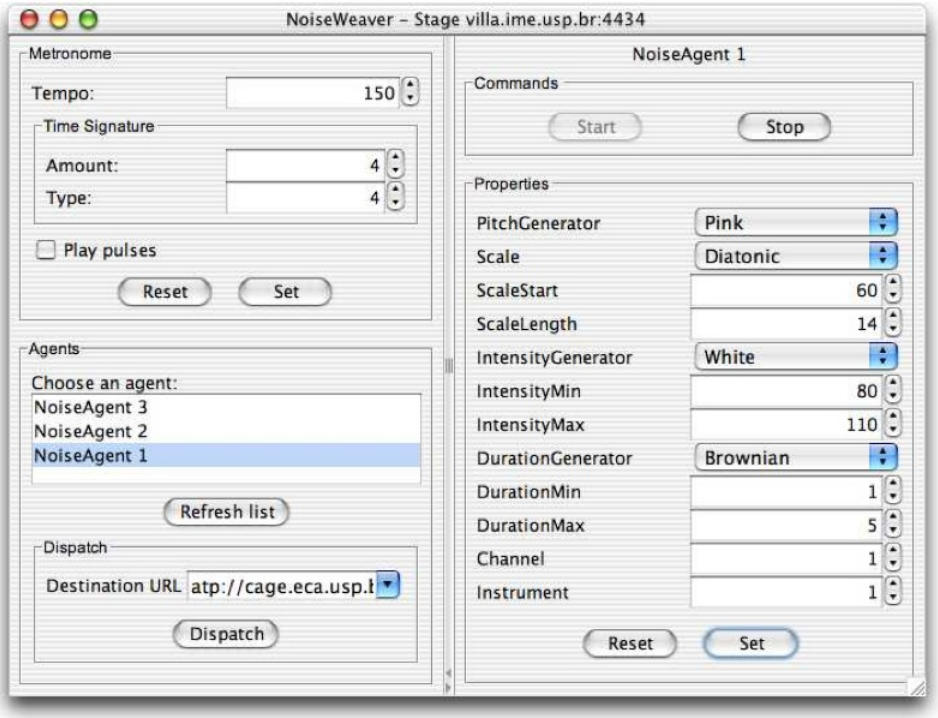

Figura 4: Interfaccia dell'applicazione NoiseWeaver

Il pannello *Metronomo* permette all'utente di definire i parametri del metronomo. La variabile *Tempo* è in battute al minuto, mentre *Amount* e *Type* definiscono i parametri temporali; *Play pulses* determina se il metronomo suona una nota ogni battuta. Il pannello *Agents* contiene una lista di tutti i NoiseAgent presenti nello Stage. Il pannello *dispatch* si riferisce all'agente selezionato e gli invia il comando di migrare in un altro Stage.

Quando un agente viene selezionato, viene attivato il pannello nel lato destro della finestra. Questo pannello da' la possibilità all'utente di cambiare le proprietà dell'agente, che ovviamente condizionano la sua composizione. Nel pannello *Commands* sono presenti i due comandi principali, *Start* e *Stop*, che non necessitano di una particolare descrizione.

Nel pannello *Proprietà*, in basso, sono presenti invece tutte le proprietà relative alla composizione ed all'esecuzione della musica da parte dell'agente.

In questo modo è possibile avere numerosi agenti di tipo NoiseAgent che suonano ognuno una melodia, controllando l'esecuzione di ognuno di questi tramite i comandi e le proprietà fornite dalla GUI dell'applicazione.

2.4.1 Risultati

Nei numerosi test effettuati su questa semplice applicazione, sono stati riscontrati buoni risultati relativamente alla composizione musicale. L'applicazione ha gestito correttamente l'introduzione e l'esecuzione simultanea di più agenti sullo stesso host, senza subire sensibili rallentamenti.

Nell'ottica dello sviluppo di applicazioni più complesse, sarà però necessario implementare un supporto più stabile per assicurare il rispetto dei vincoli *real-time* ed alla *quality of service* minima richiesta. Ovviamente il rispetto di questi requisiti è dipendente dal supporto fornito dalla rete a cui i computer sono collegati: se il servizio risulta di buona qualità, lo spostamento degli agenti e la loro interazione avviene in maniera relativamente trasparente rispetto alla loro effettiva posizione geografica.

2.5 Conclusioni

Il progetto Andante vuole essere più che un sistema software: l'obiettivo è quello di creare una community di sviluppatori e musicisti che collaborino nella creazione e nello sviluppo di idee musicali, con l'obiettivo di sviluppare una nuova piattaforma software[6]. Infatti è possibile notare come sia posta particolare enfasi nell'aspetto relativo all'esecuzione distribuita e all'interazione umana, lasciando leggermente in secondo piano la parte relativa alla composizione musicale ad agenti. Anche per questo motivo un parallelismo con la teoria A&A risulta più difficile per questo sistema, in cui spesso gli agenti vogliono essere la rappresentazione nello spazio virtuale di musicisti umani.

Possiamo comunque riscontrare un comportamento *goal-governed* degli agenti, il cui obiettivo esplicito è comporre una melodia secondo determinati parametri modificabili dall'utente anche durante l'esecuzione. Ognuno di essi risiede all'interno di un **ambiente virtuale distribuito**, in cui viene propa-

[6] a questo scopo, è stato aperto un sito internet all'indirizzo `http://gds.ime.usp.br/andante`.

gato il suono. In particolare, ogni istanza di `Stage` ne è la rappresentazione locale.

L'ambiente, inoltre, mette a disposizione degli agenti vari artefatti, tra cui il **canale audio**, necessario per la riproduzione di suoni durante la performance distribuita. Viene inoltre messo a disposizione un artefatto che funge da **metronomo**, utilizzato a *livello di co-ordinazione* dagli agenti, sia che essi risiedano sullo stesso host o meno.

In questo caso, il divario concettuale che si può riscontrare è che gli artefatti non sono "situati" nell'ambiente come suggerisce la teoria A&A, ma vengono "forniti" da un'istanza dell'ambiente stesso quando richiesto. Ovviamente anche in questo caso la discrepanza è verosimilmente dovuta al paradigma concettuale utilizzato nell'implementazione del sistema, del quale sarebbe però interessante valutare una futura realizzazione nell'ottica della teoria A&A, visto gli ottimi risultati ottenuti nelle performance distribuite.

3 Un framework per Musical Multiagent Systems

Viene ora presentata la proposta di un framework che riassuma i precedenti lavori nell'ambito dei sistemi multi-agente applicati alla composizione ed esecuzione musicale, con l'obiettivo di risolvere i problemi di basso livello più comuni, come la sincronizzazione real-time, la comunicazione dei suoni e la mobilità spaziale degli agenti. Utilizzando questo framework, gli utenti possono sviluppare il proprio sistema multi-agente focalizzandosi principalmente sui propri vincoli musicali, lasciando i problemi tecnici al sistema. Per validare questo sistema, verrà anche presentato un caso di studio reale, *Living Melodies*, che affronta alcune di queste tematiche.

3.1 Introduzione

Molte delle opere riguardanti la *computer music* ed i *sistemi multi-agente* sono generalmente molto limitate nel proprio scopo, affrontando le principali problematiche spesso in maniera non esaustiva. Lavori che invece mirano ad un contesto più generale sono invece l'*architettura MAMA* (descritta nel capitolo 1), dove gli agenti comunicano tramite una rappresentazione simbolica della porzione di musica che stanno eseguendo, ed il *progetto Andante* (presentato nel capitolo 2), che fornisce agli agenti la capacità di migrare da una macchina all'altra in un ambiente distribuito.

Questo lavoro mira alla definizione ed all'implementazione di un framework generale per sistemi musicali multi-agente, trattando ovviamente lo studio delle problematiche connesse. Verrà estesa la piattaforma MAMA, permettendo varie forme di comunicazione sincrona e asincrona (segnali audio, stream di musica simbolica, segnali video ed altre forme), verrà introdotto il concetto di vita artificiale (vita, morte e riproduzione degli agenti) e predisposta una simulazione fisica (la propagazione del suono ed il movimento degli agenti). Gli agenti musicali introdotti possono inoltre beneficiare dell'abilità di migrare, similmente agli agenti presentati in Andante, visto che entrambi i sistemi sono realizzati in Java.

Un aspetto centrale in questo progetto è il trattamento dello *spazio*, un attributo musicale di grande importanza nella composizione musicale contemporanea, che è stato però esplorato superficialmente nei precedenti lavori nel contesto dei sistemi multi-agente. Infatti uno degli obiettivi principali del framework è il permettere la simulazione della propagazione del suono in un ambiente virtuale, modificando automaticamente le informazioni relative ai

suoni ricevuti da ogni agente, in base alla propria posizione ed alle condizioni di ascolto.

Attraverso l'osservazione degli elementi comuni nei sistemi musicali multi-agente già sviluppati e proponendo nuove caratteristiche e strumenti utili, si intende realizzare un framework concettuale e computazionale che semplifichi l'implementazione di sistemi multi-agente che rispecchino le caratteristiche richieste dall'utente.

3.2 Tassonomia del sistema

In questa sezione verrà presentata brevemente una tassonomia riassuntiva dei concetti principali relativi ai sistemi musicali multi-agente, generalizzando quanto appreso dai precedenti lavori. Le sotto-sezioni presentate di seguito rappresentano le categorie di alto livello presenti concettualmente nel sistema, come mostrato in figura 5.

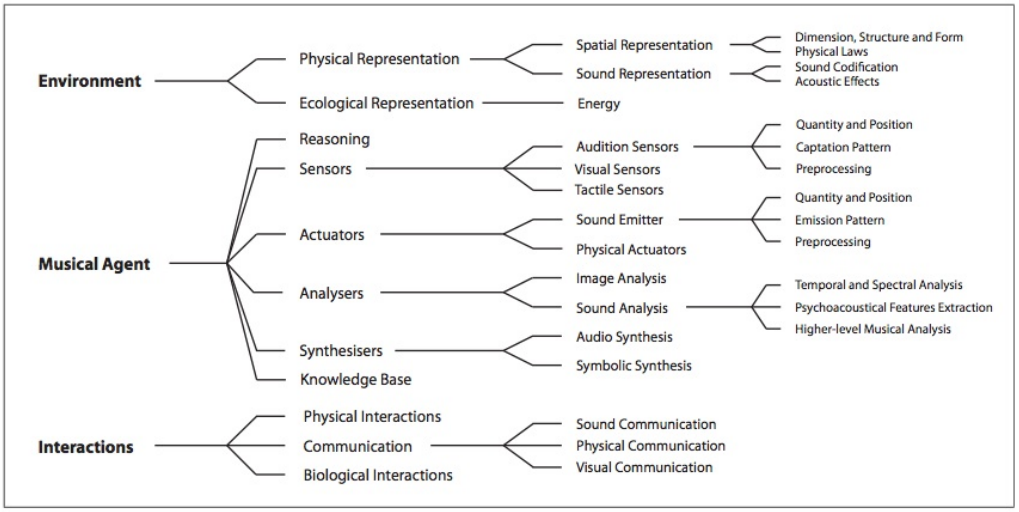

Figura 5: Tassonomia del sistema multi-agente

3.2.1 Ambiente virtuale

Un ambiente, nel contesto dei sistemi musicali multi-agente, può essere definito come uno spazio virtuale nel quale gli agenti computazionali sono immersi ed interagiscono con lo stesso tramite sensori ed attuatori.

Rappresentazione fisica. La rappresentazione fisica di un ambiente include la definizione di un mondo virtuale insieme con ogni informazione fisica rilevante per l'applicazione musicale. Queste possono includere la rappresentazione dello spazio (dimensionalità, ecc...), la rappresentazione del suono (MIDI, audio) e la propagazione (inclusi gli effetti acustici come l'assorbimento dell'aria, riflessione, diffrazione o effetto Doppler) e le leggi meccaniche (gravità, inerzia, attrazione e collisione).

Rappresentazione ecologica. I sistemi trattati utilizzano spesso una rappresentazione per l'energia in modo da controllare alcuni aspetti nella vita di un agente, in particolare il mangiare, muoversi, crescere. L'energia può essere definita ed utilizzata per controllare quali azioni (richiedenti una quantità di energia) un agente può eseguire.

3.2.2 Musical Agent

L'agente musicale è un agente computazionale specializzato nel processare suoni ed informazioni musicali. Tipicamente, questo agente è capace di analizzare suoni in ingresso, eseguendo una sorta di ragionamento musicale e formulando una risposta tramite la processazione del suono e la sintesi. Di seguito vengono presentati i componenti di un agente musicale.

Base di conoscenza. La base di conoscenza è un'area di memoria in cui l'agente tiene traccia di ogni cosa relativa al proprio know-how nella composizione della musica, oltre alla memoria degli eventi passati.

Ragionamento. Viene denotato come ragionamento l'insieme dei meccanismi che un agente utilizza per decidere le proprie azioni future, musicali o meno.

Sensori ed attuatori. I sensori catturano informazioni riguardo all'ambiente e le inoltrano al centro cognitivo dell'agente (*base di conoscenza* e *ragionamento*). Questi sono solitamente specializzati nel ricevere un particolare tipo di informazioni sensoriali.

Gli attuatori invece sono l'attiva controparte dei sensori e sono responsabili della modifica dell'ambiente, basandosi sul ragionamento dell'agente stesso.

Analisi e sintesi. Oltre ad ogni tipologia di analisi automaticamente eseguita dai sensori dell'agente, può essere richiesta un'analisi di alto livello per il raggiungimento degli obiettivi musicali. Questa può includere un'analisi contestuale (comparando gli input istantanei con i dati precedenti) o la pianificazione di eventi futuri (comparando i possibili output con quelli attesi dagli altri agenti), sfruttando una serie di tecniche specifiche di processazione del suono.

La sintesi può essere considerata la parte fondamentale dell'agente musicale, visto che, tramite questa, l'agente partecipa musicalmente ad un'esecuzione collettiva. Essa include una sintesi simbolica di alto livello degli eventi, sintesi del suono e tecniche di processazione dei segnali, ma anche informazioni non musicali come le traiettorie spaziali e l'output grafico (per comunicare visivamente con altri agenti o con la controparte musicale durante la performance musicale).

3.2.3 Interazioni

In un'esecuzione musicale collaborativa ci sono molti tipi di interazioni che possono essere simulate in un sistema musicale multi-agente. Oltre l'ovvio scambio di suoni durante la performance (attraverso l'ascolto e l'esecuzione contemporanea), possono essere scambiate anche altre informazioni, come segni e gesti per guidare l'esecuzione.

Quando la simulazione include elementi fisici o biologici, si possono osservare altre interazioni, tra gli agenti o tra l'agente e l'ambiente, che modificano il loro stato interno. Queste includono ad esempio la mobilità degli agenti a causa di forze esterne o per il libero arbitrio, il sostentamento e la riproduzione.

3.3 Il framework

Il framework sviluppato presuppone l'utilizzo di un middleware multi-agente che fornisca l'infrastruttura necessaria per assicurare l'esecuzione degli agenti e controllare i messaggi scambiati. L'implementazione corrente utilizza *JADE 3.6*[7], sviluppato con la versione 6 del linguaggio Java.

È inoltre importante sottolineare che l'architettura proposta è stata realizzata con il principale scopo di permettere lo sviluppo da parte di terzi di componenti aggiuntivi che facilmente possano interfacciarsi ed utilizzare la struttura finora realizzata, demandando a questa le operazioni di basso livello.

[7] ulteriori informazioni disponibili all'indirizzo `http://jade.tilab.com`.

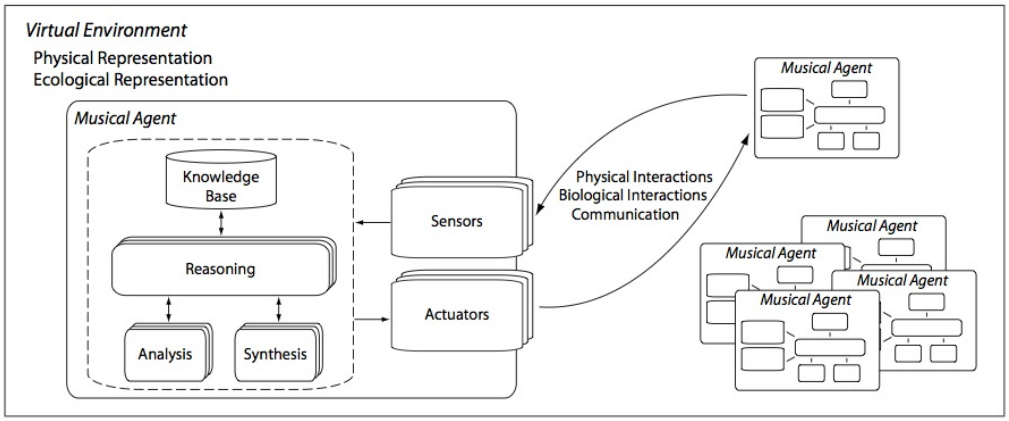

Figura 6: Gli agenti musicali nell'ambiente virtuale

3.3.1 Attori del sistema

Riferendoci alla tassonomia presentata nella sezione 3.2, un *agente musicale* è modellato come un aggregato di componenti connessi, come il ragionamento, la base di conoscenza, i sensori, gli attuatori, e via dicendo. Allo scopo di creare un agente di questo tipo, si possono definire questi componenti utilizzandone altri già sviluppati o estendendo le classi base per creare nuovi componenti.

L'ambiente virtuale è rappresentato da un unico agente chiamato *environment agent*. Questo agente è composto dalla rappresentazione fisica del mondo virtuale nel quale l'agente vive e dagli eventi composti. Le interazioni sono rappresentate dagli eventi che avvengono tra gli agenti e l'environment agent, dove ogni tipologia rappresenta una determinata interazione, come lo scambio di suono, i messaggi verbali, ecc... .

Un *agente esterno* è un musicista umano o un sistema all'esterno che interagisce a run-time con il framework. Questo agente si incarna nel mondo virtuale come un agente musicale standard, in modo che possa interagire con l'ambiente e con gli altri agenti musicali utilizzando i meccanismi di default.

3.3.2 Tempo virtuale

Nell'ambiente virtuale il tempo è controllato da un clock virtuale che permette agli agenti di ottenere il tempo corrente e pianificare i compiti. Questo clock virtuale può essere gestito in due modi differenti: *tramite il clock interno*, che significa che il clock virtuale è legato al clock interno del computer e quindi allo scorrere del tempo all'esterno; oppure a discrezione *dell'uten-*

te, che tramite l'environment agent aggiorna il valore del clock, lasciando la possibilità di effettuare misure temporali non omogenee.

3.3.3 Comunicazione

La comunicazione nel middleware multi-agente ha due principali scopi: permettere le operazioni ed il controllo del framework, ed implementare le interazioni nel mondo virtuale. Il primo compito è realizzato tramite i *comandi*, mentre il secondo tramite la comunicazione via *eventi*.

Comandi. Un comando è un messaggio passato tra agenti che controlla il funzionamento interno del framwork. Ogni agente ha un canale di comunicazione asincrona dedicato, che può inviare e ricevere comandi e viene utilizzato per passare messaggi di controllo (come registrare un agente, informare di un cambio-turno o aggiornare la conoscenza base).

Eventi. Le interazioni tra gli agenti musicali e l'environment è realizzata tramite lo scambio di eventi, che è controllato dai server degli eventi nell'environment. Gli eventi sono utilizzati per modellare ogni sorta di interazione nel mondo virtuale, come ad esempio lo scambio di segmenti audio o la collisione di due agenti.

Ci sono due tipologie di scambio per gli eventi: gli *eventi sporadici*, come gesti o messaggi verbali, che possono essere inviati in ogni momento e ad ogni frequenza; oltre a questi, sono modellati gli *eventi periodici*, come i segmenti audio, che devono seguire una frequenza di scambio precedentemente definita tra tutti i componenti. La modalità di scambio periodico è un processo di comunicazione sincrona che coinvolge un set di sensori, attuatori ed un server di eventi. In questo caso, il tempo virtuale è diviso in finestre che hanno come durata il periodo dell'evento periodico stesso, e la processazione è sempre eseguita in anticipo.

L'interfaccia di comunicazione, presente in ogni sensore, attuatore e server di eventi, è responsabile dell'invio e della ricezione degli eventi tramite la chiamata del metodo `send()` e `receive` rispettivamente. Sono state sviluppate e comparate due interfacce: *comunicazione tramite messaggi*, che utilizza il sistema di message-passing nativo di JADE, basato su messaggi FIPA-ACL; il suo funzionamento si basa sull'incapsulamento di ogni evento dentro un messaggio, lasciando a JADE il suo trasporto. Questo tipo di comunicazione può essere utilizzata nei sistemi di rete, ma può risultare più lenta e quindi non adatta ad eventi sensibili al tempo.

Può essere invece utilizzata la *comunicazione tramite direct call*, implementata come un'istanza di un singolo servizio in JADE, accessibile da ogni agente. Un componente deve registrarsi all'access point del servizio, che verrà utilizzato dal servizio stesso per la consegna degli eventi tramite chiamate dirette al metodo `sense()` fornito dall'interfaccia. Questa implementazione ha il vantaggio di essere molto più veloce rispetto al tradizionale message-passing, ma d'altro canto blocca il servizio fintanto che il metodo `sense()` non ha restituito il controllo, per cui deve essere utilizzato con cautela.

3.3.4 Esecuzione

Sono presenti due possibili modalità di esecuzione: la *modalità Batch*, dove il tempo è discreto e diviso in turni, aggiornando il clock tramite l'environment agent solo dopo che ogni agente ha effettuato le proprie operazioni, senza nessun vincolo temporale. Può essere utilizzata quando c'è la necessità di controllare la sequenza delle azioni oppure nei processi computazionalmente molto intensi ma che non presuppongono interattività nella creazione dell'output musicale.

Oppure è possibile utilizzare la *modalità Real-Time* dove il tempo è controllato dal clock interno e ogni agente è responsabile di produrre il proprio output in anticipo, oppure rimanere in silenzio (cioè l'environment agent non attende che ogni agente completi le proprie azioni, semplicemente inoltra gli eventi che sono arrivati prima di una certa deadline).

3.4 Caso di studio: "Living Melodies"

Per validare lo stato del framework finora realizzato, sono stati realizzati numerosi sistemi reali che ne testassero la robustezza, versatilità e performance computazionali. Di seguito viene presentato il caso del sistema **Living Melodies**, un esempio complesso di come un sistema musicale multi-agente abbia vari sensori ed attuatori per gestire i suoni, l'energia, il ciclo della vita, il movimento ed il contatto. Gli agenti suonano, si muovono, si riproducono e muoiono, e la loro fisiologia non include unicamente i dispositivi per ascoltare e suonare, ma anche sensori tattili, codici genetici e regole per la riproduzione e la gestione dell'energia. Il sistema utilizza una codifica simbolica del suono come evento musicale, che si propaga attraverso uno spazio discreto bidimensionale in una maniera simile ai fronti d'onda.

La simulazione del sistema *Living Melodies* è stata basata sul software realizzato[8] e sul lavoro di Dahlstedt e Nordahl[9]. Anche se non affermato esplicitamente dagli autori, sembra che utilizzi un'architettura monolitica non real-time, in cui la processazione delle azioni degli agenti avviene secondo una politica round-robin ed eseguita da un singolo thread. Il mapping al framework descritto in precedenza richiede quindi la sola esecuzione in modalità *batch*.

La re-implementazione di questo sistema all'interno del framework permette all'utente di gestire meglio alcuni parametri della simulazione, incluso il numero degli agenti, il materiale genetico, il limite di età degli agenti, l'assorbimento del suono e molti altri. Fornisce inoltre un'interfaccia grafica lato utente che mostra la distribuzione spaziale degli agenti nel mondo, oltre ai fronti d'onda della propagazione del suono.

3.4.1 Risultati

L'implementazione di questo sistema ha dimostrato come l'attuale stato di sviluppo del framework sia capace di coprire sistemi complessi sotto molteplici aspetti, come l'audio digitale real-time e lo scambio di informazioni simboliche di suono. Le feature messe a disposizione dal framework permettono all'utente di concentrarsi nella risoluzione del proprio specifico problema musicale senza preoccuparsi di problemi di basso livello, come la comunicazione tra agenti e la sincronizzazione.

Come per lo scambio di eventi periodici, i test hanno mostrato che è possibile realizzare una comunicazione audio real-time tra gli agenti, mantenendo un rate audio costante e scegliendo accuratamente la dimensione dei segmenti audio come funzione del numero degli agenti nel sistema. Tuttavia, siccome la dimensione dei segmenti audio determina la latenza generale nella percezione delle performance degli agenti tra di loro, è desiderabile migliorare i meccanismi di comunicazione allo scopo di utilizzare piccoli segmenti audio con un gran numero di agenti.

Un miglioramento ulteriore delle performance potrebbe essere ottenuto tramite un approccio strutturale differente, utilizzando le *Java Real-Time Specification*[10], che forniscono un clock temporale ad alta risoluzione (in nanosecondi) e la possibilità di effettuare scheduling real-time a questa risoluzio-

[8] disponibile all'indirizzo `http://www.ituniv.se/~palle/projects/living-melodies/`.

[9] dall'articolo, P. Dahlstedt e M. G. Nordahl, *Living Melodies: Coevolution of sonic communication*, Leonardo 2001.

[10] ulteriori informazioni disponibili all'indirizzo `http://java.sun.com/javase/technologies/realtime/index.jsp`.

ne. Per non complicare eccessivamente applicazioni standard e non imporre complesse richieste di sistema all'utente comune, l'obiettivo rimane quello di rendere disponibili entrambe le implementazioni, da utilizzare in base alla necessità.

3.5 Conclusioni

Il framework presentato rappresenta sicuramente l'esempio che maggiormente si presta alla modellazione tramite la teoria A&A, in quanto si pone l'obiettivo di modellare tutti gli aspetti di basso livello lasciando il più puro possibile il sistema nella sue specifiche di funzionamento.

Gli agenti modellati dal sistema presentano un comportamento *goal-governed*, con l'obiettivo di realizzare una composizione musicale pertinente all'interno dei vincoli temporali imposti. Ovviamente, nell'ottica di un sistema ad agenti, va assolutamente evitata la modalità di interazione a *direct-call* che, trasferendo il controllo, non assicura l'autonomia degli agenti in ogni istante, pur diminuendo l'overhead dovuto alla comunicazione.

Lo spazio viene modellato attraverso l'**environment agent** e assume una concezione fisica notevolmente più marcata: vengono infatti presi in considerazione aspetti avanzati come la propagazione del suono e l'assorbimento dell'aria, che possono modificare l'ascolto dell'output da parte degli agenti, influendo quindi sulla composizione.

L'interazione viene invece realizzata principalmente tramite l'ausilio di due tipologie di artefatti, i **sensori** e gli **attuatori**: entrambi vengono utilizzati a *livello di co-ordinazione*, ma mentre i primi colgono i cambiamenti o i flussi informativi dall'esterno, i secondi modificano l'ambiente circostante a seconda di quanto disposto dall'agente.

Ovviamente è necessario concludere anche in questo caso che il parallelismo con la teoria A&A non può essere perfetto, in quanto è presente, se pur in misura inferiore ai casi precedenti, un *paradigm shift* sensibile. Analizzando comunque la struttura del framework, realizzato unendo alcune tra le migliori caratteristiche dei due sistemi presentati in precedenza, si può notare come questo sia il miglior punto di partenza per un'implementazione con un linguaggio ad agenti, visto l'ottimo incapsulamento degli aspetti di basso livello rispetto al funzionamento del sistema dal punto di vista dell'utilizzatore.

Riferimenti bibliografici

[1] D. Murray-Rust, A. Smaill, M. Edwards, *"MAMA: An architecture for interactive musical agents"* Proceeding of the 2006 conference on ECAI 2006, IOS Press, 2006

[2] David Murray-Rust, *"Musical Acts and Musical Agents: theory, implementation and practice"* School of Informatics, University of Edinburgh, 2008

[3] Leo Kazuhiro Ueda, Kon Fabio Kon *"Mobile musical agents: the andante project"* OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, ACM, 2004

[4] Leo Kazuhiro Ueda, Fabio Kon *"Andante: A Mobile Musical Agents Infrastructure"* In Proceedings of the 9th Brazilian Symposium on Computer Music, 2003

[5] Leo Kazuhiro Ueda, Fabio Kon *"Andante: Composition and Performance with Mobile Musical Agents"* In Proceedings of the International Computer Music Conference 2004, 2004

[6] Leandro Ferrari Thomaz, Marcelo Queiroz *"A Framework for Musical Multiagent Systems"* In Proceedings of the SMC 2009 - 6th Sound and Music Computing Conference, 2009