UNIVERSITÀ DEGLI STUDI DI TRENTO
Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea (triennale) in Informatica

Elaborato Finale

Sistema Adattivo per Internet Radio Basato su Cultura Implicita

Relatore:
prof. Enrico Blanzieri

Correlatore:
dott. Aliaksandr Birukou

Laureando:
Paolo Alberto Spada

Anno Accademico 2007 - 2008

Alla mia Famiglia
che sempre mi è vicina e mi sostiene

Ringraziamenti

Il lavoro svolto nella presente tesi non sarebbe stato possibile senza il supporto di molte persone, a ciascuna delle quali vorrei giungesse la mia riconoscenza.

Ringrazio il Relatore per i preziosi suggerimenti datimi ed il Correlatore per la disponibilità dimostratami nell'assistermi in questo mio lavoro. In particolare modo ringrazio gli amici che mi sono stati vicini, ognuno a proprio modo, dandomi un supporto sia morale che pratico. Infine, e sopratutto, ringrazio la mia famiglia che sempre mi sostiene.

Indice

1 Introduzione

In linea generale è definibile “adattivo” un qualsiasi sistema che abbia una connotazione dinamica e che sia in grado cambiare il proprio comportamento in base ai cambiamenti che l'ambiente in cui esso opera, subisce in modo da adattarsi a qualcosa o qualcuno. Si pensi, ad esempio, all'essere umano. Esso rappresenta, per sua natura, un sistema in grado di adattarsi ai cambiamenti che l'ambiente in cui vive ed il contesto in cui opera subiscono. E' un sistema capace di apprendere e di modificare, conseguentemente, il proprio comportamento.

In ambito strettamente informatico, un sistema adattivo è un'applicazione capace di apprendere in base agli input ricevuti, e di modificare, quindi, il proprio output in modo da adattarsi ad un determinato “soggetto”. Gli ambiti in cui i sistemi adattivi trovano applicazione sono numerosi, sia nel campo della ricerca che in quello commerciale. Fanno parte di questo tipo di sistemi i cosiddetti sistemi di raccomandazione.

Si consideri il caso in cui ci si trovi nella situazione di voler noleggiare un film, ma non si abbia una idea precisa su quale far cadere la propria scelta. Nell'indecisione si chiede consiglio ad un amico che ci indirizza sulla scelta, ad esempio, di “V for vendetta”. Il giorno seguente, avendo gradito la pellicola, si ringrazia l'amico per il consiglio datoci (ovvero la “raccomandazione”). In questo esempio, consigliandoci il film da vedere, l'amico ha svolto la funzione di un sistema di raccomandazione.

I sistemi di raccomandazione sono quindi delle applicazioni che hanno lo scopo di proporre ad una persona informazioni che riscontrino gli interessi e le preferenze della stessa.

Un campo nel quale questi sistemi hanno avuto una grande applicazione è quello della rete Internet e del World Wide Web (WWW) in particolare. Poiché la quantità di informazioni fruibili attraverso tale mezzo di comunicazione ha raggiunto livelli considerevoli tanto da generare il un problema di “sovraccarico dei contenuti”, l'utilizzo di strumenti capaci di proporre in maniera automatica quei contenuti che effettivamente si rivelino di proprio interesse è diventato fondamentale.

Un famoso esempio di sistema di raccomandazione è quello utilizzato dal portale di commercio elettronico Amazon.com [1]. Il sistema in questione, attraverso un'attività di monitoraggio delle azioni compiute da un cliente all'interno del portale di vendite, come ad esempio gli acquisti effettuati in passato piuttosto che le ultime pagine dei prodotti visualizzate, ne costruisce un profilo delle preferenze che utilizza per proporgli raccomandazioni d'acquisto che possano essere di suo interesse. Un altro settore in cui questi sistemi hanno avuto un grande successo è quello ricreativo. Esistono, tra gli altri, servizi che effettuano raccomandazioni sui film da vedere o che propongono dei suggerimenti sulla musica da ascoltare. E' proprio il dominio delle raccomandazioni in campo musicale il contesto nel quale il lavoro svolto in questa tesi si colloca. L'obiettivo è quello di analizzare, come il concetto di “Cultura Implicita” possa essere applicato a questo particolare dominio attraverso la realizzazione di un sistema adattivo per internet radio basato su di esso.

2 Sistemi di raccomandazione musicale e Cultura Implicita

Questo capitolo ha come obiettivo, quello di descrivere il contesto generale all'interno del quale il lavoro svolto nella presente tesi si colloca. La prima parte introduce il concetto di "raccomandazione" e delinea i principali aspetti teorici alla base dei sistemi di raccomandazione, dopodiché presenta un breve excursus storico di questi sistemi ed i principali ambiti in cui essi vengono impiegati. In una seconda parte saranno descritti in modo sommario i principali approcci utilizzati per l'implementazione di questi sistemi. Nella terza parte verrà effettuata una panoramica sullo stato dell'arte dei sistemi di raccomandazione musicale presentandone due esempi popolari. Infine, l'ultima parte del capitolo, descrive l'approccio denominato "Cultura Implicita" sul quale si basa il sistema realizzato.

2.1 La raccomandazione di contenuti informativi

Nel capitolo introduttivo, sulla base dell'esempio proposto, è stato definito un sistema di raccomandazione come un'applicazione atta a proporre ad un utente dei contenuti che riscontrino le sue preferenze, ovvero delle "raccomandazioni". Sebbene tale definizione sia corretta ed adatta per una descrizione generale, dal punto di vista formale e tecnico si rende necessaria una più approfondita analisi. A questo proposito vengono in primo luogo introdotti i concetti di Information Retrieval (IR) e di Information Filtering (IF) evidenziandone gli aspetti comuni e le principali differenze ed in secondo luogo sarà presentato un breve excursus storico di questi sistemi.

2.1.1 Information Retrieval ed Information Filtering

L'IR ("recupero di informazione") è un concetto che si pone come obiettivo quello di sviluppare strumenti in grado di processare una grande quantità di informazioni in modo da assistere le persone nel compito di ricerca e recupero di quelle di cui "necessitano". A partire da una esplicita richiesta dell'utente, generalmente espressa mediante delle "parole chiave", questi sistemi recuperano quei contenuti informativi che risultano essere rilevanti. Uno dei tanti contesti in cui i sistemi che svolgono IR vengono impiegati è quello della ricerca di informazione nel WWW, dove i motori di ricerca ne sono l'esempio principe. Attraverso l'utilizzo di complesse tecniche algoritmiche, essi raccolgono, processano ed indicizzano una ingente quantità di informazione in modo da renderla, poi, facilmente "interrogabile" a partire da una richiesta effettuata da un utente mediante l'uso di "parole chiave". Risulta quindi corretta l'affermazione che i motori di ricerca forniscono all'utente l'informazione di cui egli "necessita". La stessa affermazione risulterebbe altrettanto corretta se riferita ai sistemi di raccomandazione [2].

E', a questo punto, importante introdurre il concetto di IF ed evidenziarne la differenza rispetto a quello di IR. Se da un lato, i motori di ricerca hanno come obiettivo quello di aiutare l'utente a trovare l'informazione specifica che sta ricercando, dall'altro, un sistema

di raccomandazione si differenzia in quanto esso si focalizza principalmente nel proporre all'utente dei contenuti che potrebbero essere di suo interesse, ma che, generalmente, non corrispondono alla specifica richiesta; pertanto essi aiutano l'utente a scoprire nuovi contenuti ed informazioni riguardanti un particolare dominio [3]. E' questo ciò che il concetto di IF cerca di catturare. Per poter applicare tecniche di IF, i sistemi di raccomandazione hanno la necessità di raccogliere informazioni riguardo l'utente per crearne un profilo che ne rispecchi le preferenze. Tali informazioni possono essere recuperate in due maniere: esplicita o implicita. La maniera esplicita consiste nel chiedere in modo diretto all'utente di esprimere un giudizio riguardo ad un qualche "contenuto informativo" propostogli. La maniera implicita al contrario consiste nel tracciare in modo non invasivo i comportamenti e le azioni che l'utente compie.

Maggiori dettagli sulle principali tecniche per svolgere IF utilizzate nei sistemi di raccomandazione sono presentati nella sezione 2.2.

2.1.2 Cenni storici ed ambiti di applicazione

Il concetto di IF, sul quale si basano i sistemi di raccomandazione, nasce a sua volta dalla ricerca effettuata nel campo dell'IR di cui ne eredita, in parte, tecniche ed algoritmi.

Volendo però fare un breve excursus storico per risalire all'origine di questi sistemi occorre andare nella preistoria. A questo proposito, Fox A. [3] riporta una citazione di Riedl & Konstan in [citazione necessaria] secondo la quale quando, in epoca preistorica, l'uomo si imbatteva in un nuovo frutto di bosco, non tutti i componenti della sua tribù sarebbero stati disposti a cibarsene immediatamente. Molti, infatti, avrebbero atteso di verificare se qualcuno si fosse ammalato dopo averne mangiato. Il fatto che nessuno si ammalasse sarebbe stato interpretato come raccomandazione per mangiare il nuovo frutto. Al contrario, se qualcuno si ammalava, allora tale evento sarebbe servito come raccomandazione negativa circa il frutto in questione. Questo semplice, ma efficace esempio, ci permette di comprendere che dalla preistoria ad oggi, l'uomo effettua le proprie scelte basandosi sulle informazioni apprese, in modo implicito (come nel caso dell'esempio appena presentato) o in modo esplicito (come nell'esempio della raccomandazione del film presentato nel capitolo introduttivo), dall'ambiente in cui vive.

L'"informazione", e le modalità con le quali essa viene acquisita ed elaborata dagli attori presenti in un determinato sistema, assume un ruolo centrale.

La nascita e lo sviluppo della rete Internet, ed il suo utilizzo come mezzo principe di divulgazione, scambio e fruizione di "contenuti informativi" rappresenta sicuramente la più grande rivoluzione tecnologica che ha investito il campo dell'informazione. In poco tempo, il WWW diventa un "luogo" in cui la quantità di informazione presente raggiunge livelli inimmaginabili, tanto da causare il problema conosciuto come "sovraccarico dei contenuti" e dove si rende necessario avere a disposizione strumenti in grado di assistere ed agevolare l'utente nel compito di "recupero" e "filtraggio" delle informazioni.

Le prime ricerche nel campo dei sistemi di raccomandazione risalgono agli inizi degli anni '90. Nel 1992 Goldberg et al. realizzano il primo sistema di raccomandazione, denominato

“Tapestry” [4] con lo scopo di filtrare documenti elettronici come e-mail e notizie. GroupLens [5], un sistema di raccomandazione di notizie ed articoli basato sul giudizio degli utenti, viene sviluppato da Resnick et al. nell'ambito della ricerca sul “Collaborative Filtering” (CF). Lo stesso gruppo di ricerca sviluppa anche un sistema per la raccomandazione di film denominato MovieLens [6].

Come accennato nel capitolo introduttivo, due degli ambiti in cui i sistemi di raccomandazione sono stati applicati con maggior successo sono quello del commercio elettronico e quello ricreativo. Per quanto riguarda il primo, è di immediata constatazione l'importanza che i sistemi di raccomandazione hanno in questo contesto. Avere, infatti la possibilità di effettuare della “pubblicità mirata” e personalizzata secondo le preferenze di ciascun cliente e capace di adattarsi ai cambiamenti che tali preferenze possono manifestare nel tempo, assume un valore commerciale determinante. Per quanto riguarda il settore dell'intrattenimento, sebbene sussista anche in esso un valore commerciale, l'applicazione di sistemi di raccomandazione ha generalmente il duplice obiettivo di, da un lato, portare l'utente attraverso un processo di “scoperta” (“discovery”) ad accrescere la propria conoscenza nell'ambito specifico (es. cinema , musica, letteratura) e, dall'altro, di creare delle comunità di “social networking” in cui le persone sono legate l'una con l'altra da passioni e gusti comuni.

Si segnala tra gli altri, nell'ambito del commercio elettronico, Amazon.com quale esempio di sito web di elevato profilo che fa uso di tecnologie per la raccomandazione (CF, nello specifico) [7] a cui, in campo ricreativo si aggiungono Last.fm [8] e Pandora [9] per quanto riguarda il settore musicale, e MovieLens [6] per quello cinematografico.

2.2 Metodologie per la raccomandazione

Sebbene gli algoritmi utilizzati dai sistemi di raccomandazione per l'elaborazione dei dati e la conseguente generazione di raccomandazioni siano innumerevoli, in quanto dipendenti dal dallo specifico problema da risolvere, gli “approcci tecnologici” adottati dalla maggior parte dei sistemi sono tre: il “Content-based Filtering”, il “Collaborative Filtering” e le soluzioni di tipo ibrido.

Di seguito viene svolta una presentazione sommaria di questi tre tipi di sistemi, ed infine sarà fatta una breve panoramica su altre soluzioni.

2.2.1 Content-based Filtering

L'approccio tecnologico del “content-based filtering” (filtraggio basato sul contenuto) consiste nell'analizzare le informazioni estratte, mediante varie tecniche, dal contenuto dell'elemento oggetto della raccomandazione e di utilizzarle per crearne una sorta di “firma” o “profilo”, sotto forma di una serie di attributi, che caratterizzi l'elemento in questione [10]. La firma così creata viene quindi memorizzata in un database che costituisce la base sulla quale i sistemi di raccomandazione che adottano questo approccio tecnologico andranno ad operare. Il processo di creazione e memorizzazione del profilo deve essere svolto per ogni elemento che si intende prendere in considerazione. Nel

momento in cui viene richiesta una raccomandazione, il sistema analizza la similarità tra l'elemento "corrente" e gli altri elementi presenti nel database e, sulla base delle preferenze precedenti dell'utente, sceglie come oggetto della raccomandazione l'elemento con il grado maggiore di similarità.

La difficoltà costituita dalla fase di analisi del contenuto di un elemento al fine di estrarne informazioni significative rappresenta una delle principali problematiche per cui questa tecnologia non viene impiegata ampiamente in ambiti in cui il contenuto dell'elemento è diverso da quello testuale (per esempio immagini o contenuti multimediali). Solo in tempi recenti, grazie agli sviluppi tecnologici nel campo dell'analisi automatizzata dei contenuti multimediali, il content-based filtering viene utilizzato in alcuni sistemi di raccomandazione. A questo proposito si faccia riferimento, per esempio, a [11]. Nel caso in cui, invece, la fase di analisi venga svolta mediante l'intervento umano, i sistemi di raccomandazione basati su questa tecnologia possono rivelarsi molto validi. In questo caso però altri, quali gli elevati costi sia monetari che in termini di tempo necessario all'analisi, risaltano tra i principali svantaggi.

Un altro importante problema insito nella tecnologia del content-based filtering è quello della "sovra-specializzazione" ("overspecialization"). Esso consiste nel fatto che i sistemi di raccomandazione basati su questa tecnologia, tendono a raccomandare soltanto gli elementi che risultano essere altamente simili a quelli già presenti nelle preferenze di un utente. In questo modo egli riceverà unicamente raccomandazioni molto simili tra loro, vedendosi preclusa la possibilità di scoprire elementi differenti.

2.2.2 Collaborative Filtering

A differenza del content-based filtering, il "collaborative filtering" (Filtraggio Collaborativo o sociale), non prevede alcun tipo di analisi sul contenuto degli elementi. Questa tecnologia, invece, computa le correlazioni tra utenti simili generalmente denominati "vicini" [10]. Per poter fare ciò, il sistema crea dei profili utente, atti a rispecchiare le preferenze dello stesso, che verranno perfezionati man mano che del feedback esplicito, o implicito sarà ricevuto. Il feedback esplicito consiste nel chiedere in modo diretto ad un utente di esprimere un giudizio di gradimento su un determinato elemento. Il feedback implicito, al contrario, non prevede alcuna azione diretta da parte dell'utente, ma viene estratto in base al monitoraggio delle azioni che egli compie nel tempo, ovvero delle sue "consuetudini". Il contenuto informativo, quindi, viene classificato in base all'opinione che gli utenti esprimono su di esso invece che sul contenuto stesso [12]. Una volta che il sistema colleziona sufficiente informazione sui profili utente, esso computa il "grado di vicinanza" fra gli stessi mettendo in relazione un utente con utenti che hanno interessi o gusti simili. In un secondo momento, le valutazioni di questi utenti simili sono utilizzate per generare le raccomandazioni per l'utente finale [7]. La procedura di raccomandazione può, in linea generale, essere così descritta: presi in oggetto due profili con elevato "grado di vicinanza" (corrispondenti, quindi, ad utenti che hanno assegnato giudizi simili ad elementi comuni), gli elementi che risulteranno raccomandabili all'utente finale saranno quelli da lui non ancora "conosciuti" ma presenti nel profilo delle preferenze

del suo “vicino” (o dei suoi vicini).

La Fig. 2.1 illustra graficamente quanto sopra riportato proponendo un semplice esempio di Collaborative Filtering tra due profili preferenziali simili. Gli elementi *{1,2,3}*, comuni ai due profili, contribuiscono a stabilirne un elevato grado si similarità, sulla base del quale, l'algoritmo che implementa il CF, seleziona, per esempio gli elementi *{8}* e *{5}* come possibili raccomandazioni rispettivamente per i profili *A* e *B*.

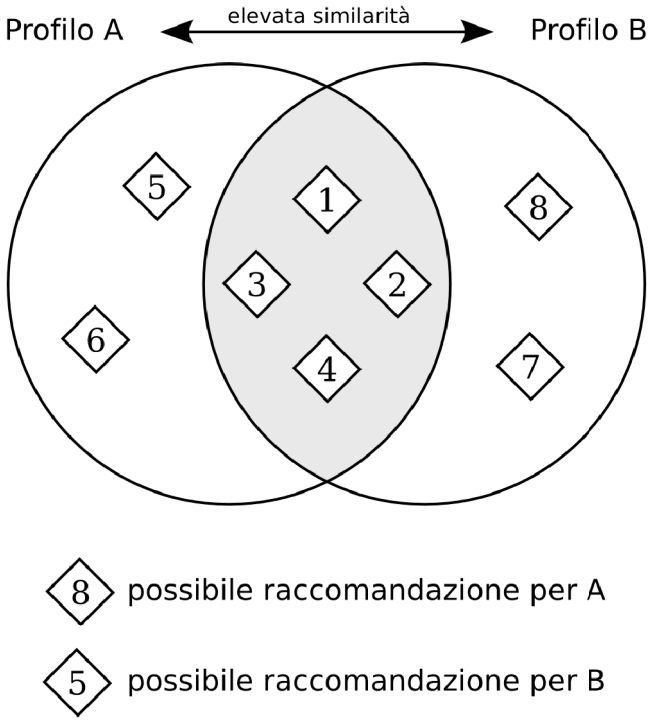

Fig. 2.1: Semplice esempio di Collaborative Filtering

Herlocker et al. in [7], sostengono che il collaborative filtering possiede vantaggi significativi rispetto al più tradizionale content-based filtering, principalmente perché esso non dipende da da una analisi del contenuto prona ad errori effettuata da una macchina. I vantaggi includono inoltre la capacità di filtrare ogni tipologia di contenuto, e di filtrare in base a concetti complessi e difficilmente rappresentabili.

Questa tecnica presenta però anche degli svantaggi. Due di questi sono rappresentati dal problema del “nuovo utente” (“new user problem”) e da quello del “cold start” (“partenza a freddo”).

Il primo problema si verifica nel momento in cui un nuovo utente entra in interazione con il sistema di raccomandazione. Poiché, infatti, il sistema necessita di una certa quantità di tempo per collezionare informazioni significative all'interno del profilo del nuovo utente, questi non riceverà raccomandazioni accurate nel primo periodo, anzi, in certi casi le raccomandazioni ricevute potranno rivelarsi anche molto errate.

Il secondo problema riguarda invece gli elementi oggetto delle raccomandazioni. Quando infatti, il sistema non possiede alcun tipo di valutazione sugli elementi presenti (in particolar modo nel momento dell'inizializzazione dello stesso), tali elementi non saranno raccomandabili poiché nessun utente, ancora, ne avrà assegnato un giudizio. Questo problema si verifica anche nel caso in cui, con il sistema a regime, un nuovo elemento diventa disponibile, ma non possiede una valutazione, tendendo quindi a restare "invisibile" per gli utenti [10]. Per cercare di limitare questo tipo di problemi, sono state proposte molte tecniche ed approcci differenti [13].

Infine, si fa notare che i sistemi di raccomandazione basati su collaborative filetering sono molto utilizzati in ambiti a "basso rischio" come quelli dell'intrattenimento. Non vengono, invece impiegati negli ambiti ad "alto rischio". Se, infatti, un utente può rischiare di acquistare un CD in base ad una raccomandazione ricevuta, probabilmente egli non rischierà di scegliere allo stesso modo la meta della luna di miele. Questo è principalmente dovuto al fatto che oggi i sistemi basati sul collaborative filtering (automatizzato) sono scatole nere, oracoli computerizzati che danno consigli, ma che non possono essere interpellati [7].

2.2.3 Sistemi ibridi e altri "flavours"

Nonostante i due approcci proposti siano molto differenti l'uno dall'altro è importante sottolineare che essi non rappresentano delle opzioni escludenti. Content-based filtering e Collaborative filtering possono essere integrati costituendo così delle soluzioni di tipo ibrido. I sistemi sviluppati secondo questo approccio misto, tentano di combinare le potenzialità delle due tecnologie e di sopperire agli svantaggi dell'una con i vantaggi dell'altra.

Questo approccio misto sembra rappresentare la soluzione "ideale" ed è, infatti, la direzione verso la quale i sistemi di raccomandazione sembrano essere indirizzati [14] [15] [16] [17]. Ciò nonostante, il fatto stesso dell'integrazione di due approcci differenti può comportare dei problemi di tipo collaterale che dovranno essere affrontati.

Infine, a titolo meramente informativo, si segnala l'esistenza di numerosi altri approcci che tentano di estendere le suddette tecnologie in modo da contemplare altri aspetti che possano essere di interesse come, ad esempio, le informazioni contestuali (sistemi di tipo context-based) o quelle conoscitive (sistemi knowledge-based). Ci si riferisca a [18] e [19] per una più completa panoramica.

2.3 Stato dell'arte dei sistemi di raccomandazione musicale

Come anticipato nella sezione 2.1.2, il settore ricreativo legato alla musica fruita e distribuita online rappresenta un dominio ideale per l'applicazione di tecnologie per la raccomandazione, in quanto racchiude in sé un duplice valore: culturale e commerciale.

Nell'ambito di questa tesi, all'interno del grande panorama dei sistemi disponibili, ne verranno analizzati brevemente due, scelti secondo i parametri di popolarità e di qualità (secondo i risultati degli studi svolti in [3] e in [10]): Last.fm e Pandora.

2.3.1 Last.fm

Last.fm è una piattaforma che integra le caratteristiche di un social networking che gravita attorno al mondo della musica con quelle di un sistema di raccomandazione. Il sistema è in grado di tracciare i gusti musicali ciascun utente, raccomandare nuove tracce musicali di suo gradimento, generare stazioni radio personalizzate e connettere, uno con l'altro, gli utenti che hanno preferenze musicali simili. "Last.fm ti connette con la tua musica preferita, ed utilizza il tuo gusto unico per trovare nuova musica, persone e concerti che ti piacciono" [8]. La peculiarità di Last.fm sta nel fatto di riuscire a collezionare una quantità molto cospicua di dati relativi alle preferenze musicali degli utenti e di abbinare questo a semplici algoritmi di raccomandazione che si basano sull'approccio tecnologico del CF.

Il tracciamento delle abitudini musicali

Per poter usufruire dei servizi messi a disposizione da Last.fm, un utente sottoscrive un account direttamente sul sito; in questo modo il sistema potrà associarvi un profilo musicale che ne rispecchierà le abitudini di ascolto. La collezione delle informazioni sulle preferenze musicali dell'utente può avvenire secondo due metodi differenti, ma non escludenti.

La prima opzione è quella di utilizzare un player (Last.fm player), che necessita di installazione locale, messo a disposizione e reperibile sul sito stesso. Mediante il player, l'utente inizia ad interagire con il sistema specificando il nome di un artista o di un particolare genere musicale che servirà come punto di partenza per ricevere le raccomandazioni correlate alla scelta effettuata. A partire, dunque, da tale scelta, il player trasmette in sequenza tracce musicali ad essa correlate, dando inoltre la possibilità di esprimerne esplicitamente un giudizio riguardo il gradimento del brano proposto. Il giudizio permette di raffinare ulteriormente le raccomandazioni che saranno ricevute e viene espresso scegliendo una delle seguenti opzioni disponibili: *love*, *ban* e *skip*. Scegliere l'opzione *love* significa esprimere un apprezzamento particolare per la traccia che si sta ascoltando. Al contrario, con l'opzione *ban*, si esprime un giudizio totalmente negativo in quanto la traccia trasmessa non è di proprio gradimento. Questa scelta comporta che tale traccia non venga mai più raccomandata, e quindi trasmessa, dal sistema. Infine, scegliendo l'opzione *skip*, l'utente fa in modo che la traccia trasmessa venga "saltata", passando quindi alla successiva, senza però esprimere alcun tipo di giudizio sulla stessa. Si fa presente, inoltre, che è anche possibile non esprimere alcun giudizio limitandosi ad ascoltare i brani musicali proposti. In questo caso, però, il sistema non potrà perfezionare le raccomandazioni future.

Mano a mano che l'utente ascolta i brani proposti, e ne esprime un giudizio, il sistema ne arricchisce e perfeziona il profilo musicale. Maggiore è la quantità di informazioni che il sistema riesce a collezionare, maggiore si rivelerà la "qualità" delle tracce musicali che verranno proposte. Si noti come questo metodo di interfacciamento a Last.fm corrisponda all'ascolto di una stazione radio personalizzata. Una delle tante funzionalità messe a disposizione dal player consente all'utente di assegnare delle "etichette" ('tags') ai brani o agli artisti ascoltati atte a descriverli in maniera soggettiva creando, quindi, delle

categorizzazioni del tutto personali. E' molto probabile che anche questo tipo di dati vengano presi in considerazione e processati dal sistema per perfezionare le raccomandazioni effettuate.

Per completezza, si noti che è possibile utilizzare una versione "online" del player direttamente sul sito di Last.fm. A differenza della versione installabile in locale, la versione "online" del player si presenta con un interfaccia grafica del tutto integrata in una pagina web.

La seconda opzione utilizzata per il tracciamento delle preferenze musicali, è quella propriamente conosciuta con il nome di "scrobblig". Questa modalità consiste nel permettere al player multimediale in uso dall'utente, mediante l'installazione di un apposito "plug-in", di inviare le informazioni sulle proprie abitudini di ascolto ad una base di dati centralizzata. Ogni volta che una nuova traccia viene riprodotta, il "plug-ing" effettua il collegamento e l'autenticazione ad un server centrale ed invia, quindi, le meta-informazioni relative al file musicale in esecuzione quali artista, album, titolo del brano. Si noti come, in questa seconda modalità, non si ricevano direttamente delle raccomandazioni, bensì l'interfacciamento con Last.fm avvenga in senso unico dall'utente al sistema. Non è possibile, inoltre, esprimere dei giudizi di gradimento sulle tracce riprodotte (si sottintende, infatti, che se un utente riproduce volontariamente tali tracce esse siano di suo gradimento).

I dati raccolti con entrambe i metodi descritti, vanno ad arricchire la base di dati centrale, e contribuiscono a definire il profilo musicale di ciascun utente sulla base del quale Last.fm elabora le future raccomandazioni che renderà fruibili all'utente attraverso il player o, sul sito stesso, nella propria pagina web personale.

La raccomandazione musicale

La grande quantità di dati collezionati in questa prima fase, costituisce la base su cui, in un secondo momento, il motore del sistema di raccomandazione di Last.fm, andrà ad operare per produrre i suggerimenti d'ascolto. Il metodo utilizzato da Last.fm per generare le raccomandazioni, come anticipato, è quello del Collaborative Filtering. Nonostante risulti piuttosto banale, il seguente esempio illustra in maniera semplice il procedimento di raccomandazione adottato da Last.fm.

Si considerino X ed Y due utenti ed A, B e C degli artisti. Ora, si supponga che l'utente X ascolti spesso gli artisti A, B e C e che l'utente Y ascolti spesso A e B determinando, quindi, una similarità tra i due profili. Il sistema raccomanderà all'utente Y l'artista C che è ad esso sconosciuto, ma che, con buona probabilità, risulterà ad esso gradito poiché presente tra gli artisti preferiti dell'utente X che è suo "vicino di gusto musicale".

E' chiaro che con due utenti e con tre soli artisti le raccomandazioni possano risultare ovvie. Ma se si considera un numero molto elevato di utenti, ed un altrettanto cospicuo numero di artisti e brani musicali, ecco che l'esempio risulta molto più interessante.

Al momento in cui si scrive, Last.fm possiede una libreria musicale che conta milioni di brani ed un'utenza che raggiungerebbe i 21 milioni individui [20]. Con queste cifre è facilmente intuibile quale possa essere il valore e la qualità delle raccomandazioni che

Last.fm è in grado di proporre aiutando, quindi, gli utenti nel processo di “scoperta” di nuova musica.

2.3.2 Pandora

Pandora è anch'esso un sistema di raccomandazione musicale che ha raggiunto un notevole grado di popolarità e di qualità delle proposte musicali elaborate. Si fonda sulle basi poste da dal progetto conosciuto come “Music Genome Progect” (“Progetto del Genoma Musicale”). Il “Music Genome Progect” nasce con l'obiettivo di catalogare i brani musicali. La peculiarità di questo progetto risiede nella modalità con la quale tale catalogazione viene effettuata. Volendo, infatti, catturare “l'essenza” di ogni singolo brano, esso viene codificato in un vettore composto da centinaia di attributi musicali (i “geni”) ottenendo, una volta conclusa l'analisi, una “sequenza genetica” che identifica il brano in modo univoco, ossia il suo DNA. Gli attributi scelti per essere analizzati cercano di considerare ogni aspetto musicale del brano e comprendono, tra gli altri, melodia, armonia e ritmo, strumentazione, orchestrazione, arrangiamenti, liriche ed anche il ricco mondo del canto e dell'armonia vocale [21]. E' importante sottolineare che l'intera fase di analisi musicale dei brani è eseguita “manualmente” da specialisti in campo musicale quali musicisti professionisti o altre figure con grande preparazione specifica.

Pandora si serve di questa accurata catalogazione per fornire il proprio servizio di raccomandazione musicale in forma di stazioni radio personalizzate. La fruizione del servizio avviene direttamente mediante il sito web attraverso un interfaccia grafica in Adobe Flash®. Ogni utente ha la possibilità di creare una o più stazioni radio che, a partire dalla selezione di un artista o del titolo di una canzone, trasmettono brani musicali relazionati alla scelta effettuata. Durante l'ascolto, è possibile affinare la propria stazione radio esprimendo un giudizio sulla traccia proposta attraverso una delle seguenti scelte: *thumb up* nel caso la traccia sia di proprio gradimento, *thumb down* in caso contrario e *skip* nel caso si voglia interrompere l'ascolto della traccia corrente per passare a quella successiva. Il giudizio così espresso verrà preso in considerazione dal sistema per le successive raccomandazioni.

Sebbene in quest'ultimo aspetto descritto, Pandora si mostri similare al player di Last.fm è fondamentale notare l'importante differenza tra i due sistemi per quanto riguarda l'approccio utilizzato per effettuare la raccomandazione. Contrariamente a Pandora, il sistema alla base di Last.fm, stando alle informazioni che è stato possibile reperire, non sembra effettuare alcuna analisi riguardo contenuto dei brani musicali, ma si limita all'osservazione del feedback esplicito o implicito, fornito dagli utenti e, in base ad esso, genera le raccomandazioni mediante l'applicazione della tecnica del Collaborative Filtering. Pandora invece, seppur contempli la possibilità di ricevere del feedback esplicito, è un sistema caratterizzato, per la maggior parte, da una forte componente di analisi del contenuto, ed è perciò considerato essere un esempio di sistema di tipo content-based. Nonostante non sia stato possibile reperire informazioni certe riguardo l'esatto peso attribuibile alle due differenti tecniche di filtraggio in Pandora, a tal proposito Fox A. E., in [3] sostiene che una stima pari al 75% di content-based filtering e ad un restante 25% di

collaborative filtering sia da ritenersi “verosimile”. Da un punto di vista strettamente formale, quindi, Pandora risulterebbe più correttamente catalogabile tra i sistemi ad approccio ibrido.

2.4 Un nuovo approccio: Cultura Implicita

All'interno del contesto fin qui analizzato si colloca “Cultura Implicita” (IC). Inizialmente proposto da Blanzieri E. e Giorgini P. in [22], IC è un approccio tecnologico mirato ai sistemi di raccomandazione e, più in generale, ai sistemi nei quali la condivisione della conoscenza tra gli attori presenti sia un fattore di centrale importanza.

In questa sezione sarà presentata l'idea di fondo che ha portato allo sviluppo di IC, verrà quindi descritto SICS, un sistema per il supporto di IC e l'implementazione realizzatane da IC-Service, infine, alcuni esempi di applicazioni basate su questo approccio saranno brevemente illustrati.

Idea di fondo

L'idea alla base Cultura Implicita è che in una comunità esiste una cultura di base usata dai suoi membri per agire in modo efficace all'interno di un ambiente. Nel momento in cui un nuovo membro entra a far parte di questa comunità, egli si ritrova in un ambiente a lui sconosciuto ed è sprovvisto della cultura della comunità stessa. Questo fatto pregiudica il suo comportamento rendendolo poco efficace. Certo, il comportamento sarebbe efficace se il nuovo membro fosse a conoscenza della cultura del gruppo. La rappresentazione di tale cultura non è però una cosa semplice, specialmente in ambienti dinamici dove la conoscenza non è statica ed oggettiva. Il concetto di Cultura Implicita prevede la possibilità di indurre la cultura di base di una comunità attraverso l'osservazione del comportamento dei suoi membri senza, quindi, renderne una rappresentazione esplicita. Tale conoscenza è utilizzata per aiutare il nuovo membro ad agire efficacemente suggerendogli le azioni da compiere a seconda delle situazioni che si trova ad affrontare nell'ambiente. Nel momento in cui il nuovo membro inizia a compiere azioni consistenti con la cultura della comunità (cioè agisce come uno dei componenti della comunità stessa), sarà avvenuto un fenomeno di trasferimento di conoscenza (“fenomeno di Cultura Implicita”).

IC è quindi definibile formalmente come la relazione tra un insieme ed un gruppo di agenti tale che gli elementi dell'insieme agiscono secondo la cultura del gruppo [22] [23].

Affinché un nuovo membro agisca secondo la cultura del gruppo, IC interviene sulla “vista” che l'agente ha dell'ambiente piuttosto che sulle “capacità” dell'agente stesso. Questa seconda possibilità, infatti, non è sempre applicabile e comporta alcuni svantaggi [22] [23]. Il fatto di modificare la vista che l'agente ha dell'ambiente causa, come conseguenza, la modifica delle azioni che lo stesso agente può eseguire al verificarsi di una determinata situazione. In questo modo il nuovo membro viene indotto a compiere alcune azioni (quelle che rispettano la cultura del gruppo denominate “azioni culturali”) piuttosto che altre, rendendo possibile lo stabilirsi della relazione di IC.

Come si nota, il concetto che IC racchiude è del tutto generico, risultando quindi

applicabile in tutti i sistemi definibili in termini di ambiente, attori, azioni, oggetti ed attributi. Inoltre, IC rappresenta una generalizzazione del Collaborative Filtering [24], quest'ultimo, infatti, ne è una particolare istanza [22].

2.4.1 Sistema per il supporto di Cultura Implicita (SICS)

Sia in [22] che in [23] Blanzieri et al. propongono un sistema per il supporto di Cultura Implicita (SICS). La definizione che ne viene data è la seguente. Un sistema per il supporto di Cultura Implicita ha come obiettivo quello di stabilire un fenomeno di Cultura Implicita. Il sistema consegue tale obiettivo mediante la generazione di "vincoli culturali validati" a partire dalle osservazioni delle azioni eseguite in determinate situazioni, proponendo agli agenti delle scene in modo tale che le loro "azioni attese" in una determinata situazione soddisfino il vincolo culturale.

Al fine di comprendere appieno la notazione e la terminologia utilizzate in seguito, si rimanda alla formale definizione di IC presentata in [22] e [23].

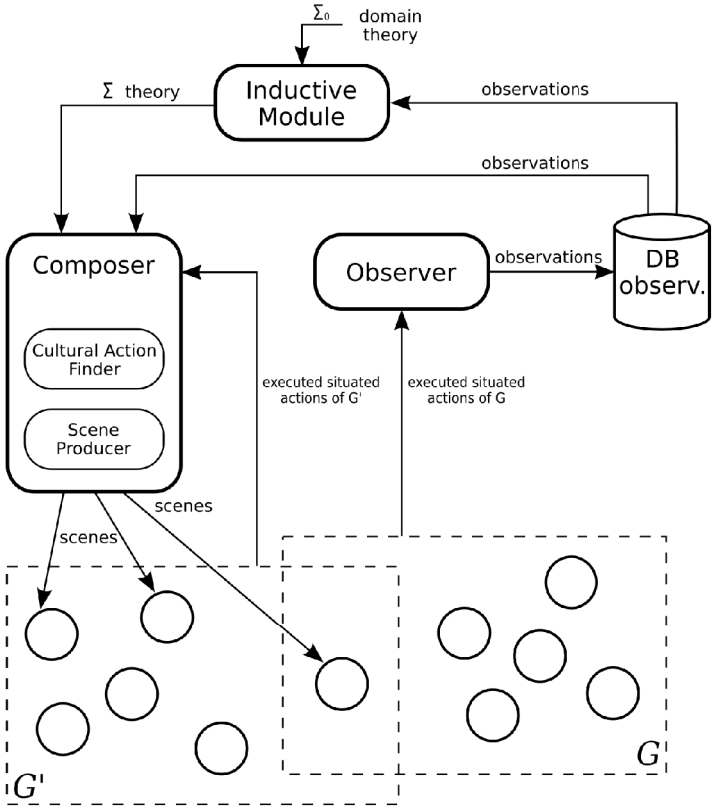

Fig. 2.2: Architettura generale di SICS

Cultura Implicita contempla il fatto che ogni agente (o attore) agisce in un ambiente eseguendo delle azioni su degli oggetti. L'ambiente è composto da oggetti e da altri attori. Oggetti, azioni ed attori possono inoltre possedere degli attributi. Una "scena" (o scenario) è una porzione dell'ambiente di fronte alla quale un attore si trova nella circostanza di dover eseguire una azione. Più precisamente un attore esegue una azione nel contesto di una

"situazione", ovvero, quando si trova ad affrontare una scena in un determinato istante di tempo. Si considera, quindi, che un agente compia delle "azioni situate". Nel momento in cui una "azione situata" sarà stata eseguita, l'ambiente subirà una modificazione e l'attore si troverà a dover fronteggiare un nuovo scenario.

L'architettura di SICS è rappresentata in Fig. 2.2. Tre sono le componenti di base: Observer (Osservatore), Inductive Module (Modulo Induttivo) e Composer (Compositore).

La componente Observer ha il compito di osservare le "azioni situate" eseguite dagli agenti appartenenti al gruppo *G* che immagazzinerà in una base di dati (DB Observer) in modo da renderle disponibili per le altre componenti del sistema.

Il Modulo Induttivo analizza le osservazioni presenti nel database e servendosi di una "teoria a priori" Σ_0, propria del dominio, induce una teoria Σ, sul vincolo culturale per il gruppo *G*. La teoria definisce le "azioni attese" ovvero quelle azioni che ci si attente vengano eseguite dai membri del gruppo *G* (denominate anche "azioni culturali"). A titolo esemplificativo si consideri la generica teoria espressa dalla Relazione 2.1. Essa stabilisce che se l'attore *att* esegue l'azione *azioneA* su un determinato oggetto *ogg*, l'azione culturale attesa sarà l'azione *azioneB* eseguita sullo stesso oggetto dallo stesso attore.

$$azioneA(att, ogg) \rightarrow azioneB(att, ogg)$$

Relazione 2.1: Esempio di teoria generica

Il Composer, infine, utilizzando la teoria Σ e le azioni eseguite, produce le scene che verranno proposte ai membri dell'insieme G' manipolandole in modo tale che le loro "azioni attese" corrispondano ad "azioni culturali" rispetto al gruppo *G*. Questo implica che le azioni eseguite dagli agenti di G' saranno, in media, azioni culturali rispetto a *G* ottenendo, di conseguenza, un fenomeno di Cultura Implicita.

Il modulo Composer è a sua volta suddiviso in due sotto-moduli: il Cultural Action Finder (CAF) e il Scenes Producer (SP). Il CAF filtra le azioni che soddisfano la teoria, mentre il SP genera quelle scene in cui le azioni filtrate dal CAF avranno una buona probabilità di essere eseguite.

E' importante sottolineare che i due gruppi, *G* e *G',* possono essere sia disgiunti che sovrapposti o addirittura coincidenti. Il fatto che *G* e G' siano sovrapposti o coincidenti implica che le azioni eseguite dai loro membri saranno prese in considerazione per elaborare il vincolo culturale e che, a loro volta, tali membri riceveranno le raccomandazioni generate dal sistema.

IC-Service

In [25], Birukou et. al propongono IC-Service, una implementazione opensource generica di SICS sviluppata secondo una architettura orientata al servizio (SOA). IC-Service è facilmente integrabile in applicazioni software già esistenti in qualsiasi dominio descrivibile in termini di agenti, oggetti, azioni ed attributi. Esistono tre modalità che

possono essere adottate per l'utilizzo di IC-Service in una applicazione. La prima consiste nell'includere tale servizio all'interno dell'applicazione stessa come libreria. La seconda consiste nel utilizzare IC-Service come servizio remoto invocabile come web-service mediante SOAP o come componente EJB. La terza modalità consiste nel servirsi di un client (SICS Remote Client) che astrae l'applicazione da ogni dettaglio tecnico del meccanismo di comunicazione con IC-Service permettendo, così, l'integrazione di questo servizio in applicazioni complesse in un modo completamente "svincolato". IC-Service rappresenta, in sostanza, un servizio di raccomandazione generico, web-based ed indipendente dal dominio applicativo.

Per una dettagliata descrizione architetturale ed implementativa di IC-Service si faccia riferimento a [25].

2.4.2 Esempi

Implicit è un sistema multi-agente di raccomandazione basato su IC con lo scopo di supportare una comunità di utenti nel ricercare il web mediante motori di ricerca. In Implicit, ogni utente è assistito da un agente personale il cui obiettivo è quello proporre all'utente collegamenti a pagine web considerate rilevanti in base alla ricerca effettuata. A partire da un parametro di ricerca (parola chiave) il sistema utilizza le informazioni estratte dalle osservazioni effettuate in passato per suggerire collegamenti a pagine web rilevanti o ad altri agenti da contattare. In questo modo le raccomandazioni ricevute da ciascun utente si basano sulle valutazioni di rilevanza date dagli altri utenti a ricerche simili effettuate in passato. Implicit realizza, quindi, l'idea di una "ricerca collaborativa" atta a migliorare la qualità dei risultati ricevuti. Una descrizione più approfondita e dei risultati sperimentali sono presentati in [24].

In [26] Birukou et al., propongono un'altra applicazione simile ad Implicit, con lo scopo di facilitare la ricerca di pubblicazioni scientifiche nel web. Servendosi di IC, il sistema osserva i pattern comportamentali dei "ricercatori esperti" rispetto alle pubblicazioni scientifiche da essi utilizzate. In questo modo, quando un "ricercatore novizio" necessita di documentazione scientifica su un determinato argomento, il sistema eviterà a tale ricercatore di perdere tempo e consumare energie nel leggere pubblicazioni poco rilevati, proponendogli, invece il materiale ritenuto rilevante dai ricercatori più esperti.

Un sistema per la raccomandazione di web-services basato sull'esperienza passata degli utenti è proposto da Kokash et al. in [27]. L'idea di fondo si basa sul fatto che se un web-service viene utilizzato da molti utenti allora tale servizio garantisce un certo livello di utilità e qualità. Al contrario lo scarso utilizzo di un servizio web può essere considerato come segnale di poca utilità del servizio stesso. Alla luce di ciò, il sistema proposto osserva i web-services maggiormente utilizzati dagli utenti in un determinato ambito, ed usa le osservazioni effettuate per raccomandare ai nuovi utenti i web-services più appropriati in base ad una esplicita richiesta.

3 Descrizione del sistema

In questo capitolo viene presentato il sistema implementato. In primo luogo verrà delineato il problema che si desidera affrontare, ovverosia l'obiettivo del sistema realizzato e ne sarà data una descrizione, ad alto livello, dell'architettura generale e delle principali componenti. Ciascuna di esse sarà infine descritta in maniera più approfondita attraverso dettagli implementativi, configurativi e funzionali.

3.1 Definizione del problema e architettura generale

Definizione del problema

Come si evince dai capitoli precedenti, il problema di assistere una persona nella scoperta di nuova musica è stato affrontato, tanto in ambito accademico quanto in quello commerciale, sviluppando appositi sistemi di raccomandazione. Questi sistemi utilizzano, per la maggior parte, gli approcci di Collaborative Filtering e Content-based filtering, a volte in maniera combinata creando così dei sistemi ibridi, descritti nel capitolo 2.2. Il Sistema realizzato, denominato "sistema adattivo per internet-radio basato su Cultura Implicita", è il primo tentativo di applicare il concetto di Cultura Implicita nello specifico ambito delle raccomandazioni musicali. L'obiettivo prefisso è quello di svolgere un preliminare studio delle potenzialità di IC in questo particolare dominio e di mettere a confronto il sistema con altre soluzioni esistenti.

Nel suo complesso, il sistema realizza l'idea di una internet-radio personalizzata. Essa, cioè, tenta di catturare le preferenze musicali di ciascun utente, ed in base a queste, di trasmettere brani che le soddisfino.

Architettura generale

Il sistema adotta il paradigma *client/server*. In questo paradigma, generalmente, il server fornisce un servizio di cui le macchine client ne fanno uso. Ciascun client invia una richiesta di servizio al server, il quale dopo un qualche tipo di elaborazione, invia al client una risposta al servizio richiesto. A titolo esemplificativo si consideri l'interazione tra un client rappresentato dal browser internet di un utente, ed un sito web ospitato in una macchina server. Nel momento in cui l'utente desidera visualizzare una determinata pagina web, il client invia una richiesta al server, il quale dopo una qualche elaborazione invia, in risposta al client, la pagina desiderata.

Si desidera far presente che, a partire dal linguaggio utilizzato (Java) e dalle singole librerie impiegate (come IC-Service e JOrbis) fino alle altre componenti descritte in seguito, l'intero sistema realizzato utilizza esclusivamente tecnologie opensource.

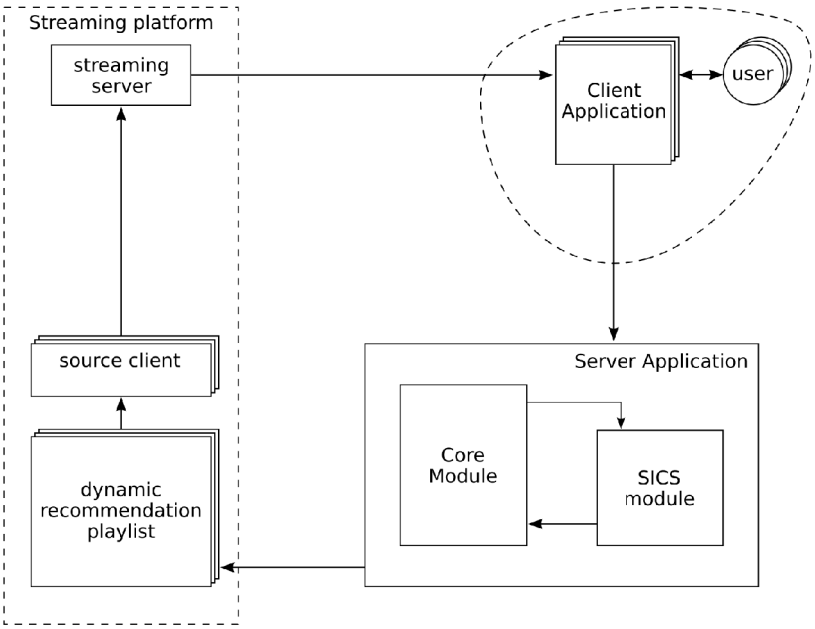

Fig. 3.1: Architettura generale

L'architettura generale del sistema, illustrata in Fig. 3.1, è costituita da tre componenti principali: l'applicazione client, l'applicazione server e la piattaforma di streaming. Quest'ultima, però può essere considerata come una parte (modulo esterno) specifica dell'applicazione server. Una presentazione esaustiva ed approfondita di ciascuna componente sarà argomento delle sezioni successive nelle quali, inoltre, i principali dettagli implementativi saranno proposti. In questa sezione si vuole dare una visione d'insieme del sistema realizzato e a questo proposito una generale descrizione del funzionamento viene di seguito riportata.

Ciascun utente interagisce con il sistema, ascoltando le tracce propostegli ed esprimendone un giudizio circa il gradimento, mediante l'utilizzo dell'applicazione client la quale, poi, invia tali informazioni al server. Applicazione client e applicazione server comunicano l'una con l'altra servendosi del protocollo TCP/IP. Questo permette loro di poter essere localizzate su macchine differenti purché collegate da una rete, come ad esempio la rete Internet. Il server utilizza le informazioni ricevute per elaborare le raccomandazioni musicali future.

Due sono i moduli che compongono l'applicazione server: il modulo Core ed il modulo SICS. Il primo modulo gestisce la logica dell'applicazione server e coordina l'interazione con il resto delle componenti. Il secondo, invece, ha il compito di elaborare le raccomandazioni sfruttando Cultura Implicita come approccio tecnologico. Quando l'applicazione server genera una raccomandazione, ovvero in base alle informazioni ricevute dal client e alle elaborazioni eseguite dal modulo SICS sceglie dalla libreria musicale il brano che meglio si adatta alle preferenze dell'utente, essa utilizzerà la piattaforma di streaming per trasmetterlo, come flusso audio, all'applicazione client. Più specificamente, il modulo Core interroga una base di dati, che funge appunto da libreria

musicale, per recuperare il brano che andrà poi ad accodare nella playlist delle raccomandazioni.

La piattaforma di streaming è costituita a sua volta da due elementi: il source client ed il server di streaming. Il source client ha il compito di generare il flusso audio a partire dai singoli file musicali elencati sequenzialmente dall'applicazione server nella playlist delle raccomandazioni. Il flusso audio viene quindi inviato al server di streaming il quale lo mette a disposizione dell'applicazione client. Quest'ultima, una volta collegata al server di streaming riceve il flusso audio che riprodurrà trasformandolo in segnale acustico in modo da renderlo ascoltabile all'utente finale. Viene così realizzata l'idea di un sistema paragonabile ad una stazione radio personalizzata. La comunicazione tra il source client ed il server di streaming e quella tra quest'ultimo e l'applicazione client avvengono entrambe mediante il protocollo HTTP.

Il sistema realizzato prevede che l'intera piattaforma di streaming e l'applicazione server risiedano entrambe su una stessa macchina, sebbene source client e server di streaming siano indipendenti l'uno dall'altro. Al contrario, come già anticipato, l'applicazione client, in uno scenario standard, risiederà in una macchina diversa da quella server, e diversa anche dalle altre macchine client. Essa infatti è legata ad un singolo utente del sistema.

L'architettura prevede infine la presenza di una base di dati ad uso dell'applicazione server. Oltre alla libreria musicale, essa contiene le informazioni relative agli utenti del sistema quali, per esempio, nome utente e password utilizzate nella fase di autenticazione. Soltanto gli utenti registrati, infatti, potranno fruire del servizio messo a disposizione dal sistema. Si sottolinea, per concludere, che i file audio componenti la libreria musicale dovranno risiedere sulla macchina server o, perlomeno, dovranno essere da questa accessibili in modo diretto.

La libreria musicale è composta da 375 brani musicali categorizzati in sette differenti "macro-generi": *Pop*, *Rock*, *Classical-Soundtrack*, *Reggae-Ska*, *Jazz*, *Folk* e *HipHop-Electro*. Questa categorizzazione si è resa necessaria poiché il genere verrà utilizzato dall'applicazione come uno dei parametri per valutare la similarità tra i brani musicali. Tutti i brani che compongono la libreria sono stati rilasciati sotto licenze *Creative Commons* [28] e messi a libera disposizione su *Jamendo* [29]. Essi inoltre sono tutti codificati nel formato audio libero *OGG Vorbis* [30].

3.2 Applicazione Server

Come anticipato nella sezione dedicata alla presentazione dell'architettura generale, l'applicazione server è composta da due moduli: il modulo Core ed il modulo SICS. Seppur l'applicazione server non implementi internamente le funzionalità della piattaforma di steaming, quest'ultima può comunque essere considerata come un suo modulo esterno in quanto viene gestita dal server stesso.

Di seguito vengono presentati i requisiti richiesti all'applicazione server, mentre una dettagliata descrizione dei moduli Core, SICS, e della piattaforma di streaming sarà argomento delle sezioni successive.

3.2.1 Requisiti

I requisiti che l'applicazione server dovrà soddisfare sono tre. Il primo consiste nel rendere fruibile il proprio servizio a più applicazioni client in modo simultaneo e, poiché tali applicazioni risiederanno potenzialmente in macchine remote, il server dovrà inoltre prevedere un meccanismo di comunicazione mediante protocollo internet su una rete. Il secondo riguarda la capacità di elaborare le informazioni ricevute dai client in modo da produrre raccomandazioni musicali che si rivelino "adeguate" alle preferenze del singolo utente. Infine, il terzo requisito richiesto è quello di rendere disponibili le tracce raccomandate ai client in forma di radio personalizzata remota fruibile attraverso la rete internet.

3.2.2 Il modulo Core

Il modulo Core realizza la logica dell'applicazione server. Ha il compito di gestire le altre componenti del sistema e di coordinarne l'interazione. Questo modulo affronta in modo specifico il primo dei requisiti richiesti all'applicazione server, ma svolgendo le suddette funzionalità di coordinamento e gestione, esso rende possibile che anche gli altri requisiti richiesti vengano allo stesso tempo soddisfatti.

Al fine di servire in modo simultaneo più applicazioni client il modulo core è stato implementato con l'approccio "mult-threading". Infatti, dal momento in cui modulo Core stabilisce una connessione con l'applicazione client, viene istanziato un thread che si occuperà di gestire l'interazione con il singolo client rendendo in questo modo possibile che le richieste di altri client possano essere servite.

```
1  java -jar ThesisServer.jar <portNumber> <path/to/music>
```

Listato 3.1: Avvio dell'applicazione Server

Il modulo Core implementa la classe *Main* per mezzo della quale viene avviata l'intera applicazione server (Listato 3.1). Al momento dell'avvio due parametri possono essere specificati. Il primo è facoltativo e corrisponde al numero della porta sulla quale il server resterà in ascolto delle connessioni dei client. Questo parametro è richiesto in quanto le applicazioni server e client comunicano l'una con l'altra attraverso la rete mediante il meccanismo delle socket java su protocollo TCP/IP; tale sistema di comunicazione sarà descritto, in maggior dettaglio, nella sotto-sezione successiva. Se nessun valore viene passato, la porta utilizzata sarà la numero *5555*. Lo stesso numero di porta dovrà, inoltre, essere specificato all'avvio dell'applicazione client come descritto nella sezione 3.3.

Il secondo parametro, invece, è obbligatorio e dovrà corrispondere alla directory locale in cui i file musicali risiedono. Una volta avviata l'applicazione, il modulo Core inizializza la base di dati in modo da creare la libreria musicale. La directory specificata come secondo parametro viene quindi scandita ricorsivamente per ricercare i file musicali presenti i quali vengono poi "mappati" in una tabella nella base di dati. La tabella, che rappresenta la libreria musicale, è composta dai campi: *id*, *path*, *title*, *album* e *genre*. Sempre nella base di

dati sono collezionati i dati relativi agli utenti del sistema, che vengono in questa fase inizializzati. Quando la base di dati è inizializzata il server crea, per ciascun utente registrato al sistema (e quindi presente nella base di dati), i file di configurazione della propria radio personale che saranno necessari per l'avvio del source client associato al singolo utente. I file in questione sono due: la playlist delle raccomandazioni e il file di configurazione dello specifico source client utilizzato dal sistema che è *ices2*. Si rimanda alla sezione 3.2.4 per ulteriori chiarimenti a riguardo. A questo punto il modulo Core procede con la configurazione del modulo SICS, avvia poi il server di streaming e, infine, inizia l'interazione con le applicazioni client mettendosi in ascolto di connessioni.

Interazione con l'applicazione client

Come è stato precedentemente anticipato, il server comunica con ciascuna delle applicazioni client attraverso il meccanismo delle socket java su protocollo TCP/IP. In specifico, il sistema prevede che i messaggi che il client invia al server vengano "inglobati" nell'oggetto *ClientObservation*. La classe *ClientObservation.java* implementa l'interfaccia *java.io.Serializable* in modo che oggetti di tipo *ClientObservation* possano essere trasmessi attraverso le socket. E' importante notare che per poter deserializzare correttamente l'oggetto scambiato, entrambe le applicazioni dovranno essere a conoscenza della stessa classe *ClientObservation*.

Come primo passo il server si mette in ascolto delle connessioni dei client sulla porta specificata all'avvio. A questo è dedicato il thread *ConnectionServer* che viene lanciato direttamente dal *Main*. *ConnectionServer* resterà costantemente in ascolto fino a che l'applicazione server non verrà terminata. Non appena una richiesta di connessione viene accettata, *ConnectionServer* lancia a sua volta il thread *ConnectionServerThread*, il quale avrà il compito di gestire la connessione e la comunicazione con il singolo client. Delegando tale compito ad un nuovo thread, il server resterà sempre disponibile ad accettare nuove connessioni.

ConnectionServerThread gestisce l'intera procedura di comunicazione con l'applicazione client (Listato 3.2). Dopo aver ricevuto da *ConnectionServer* la socket per la comunicazione con il client, il thread corrente invia all'applicazione client un messaggio di benvenuto. A questo punto si attente che il client invii sulla socket un oggetto di tipo *ClientObservation*, che conterrà l'azione eseguita dallo stesso. Per poter deserializzare l'oggetto ricevuto, il server include la classe *ClientObservation* come libreria esterna.

Nel caso venga ricevuta l'azione *connect* il server dovrà avviare il source client che genererà il flusso audio associato allo specifico utente. Nella fattispecie, invocando il metodo *lauchIces()*, viene avviato il processo esterno *ices2* al quale viene passato come parametro il file xml contenente la specifica configurazione legata alla singola applicazione client e quindi al singolo utente, che sarà stato precedentemente automaticamente creato. Il valore ritornato dal metodo *launchIces()* è l'indirizzo remoto del flusso audio attraverso il quale il client potrà riprodurre i brani raccomandati dal server. Tale indirizzo viene dunque inviato all'applicazione client attraverso la socket. Per tutte le altre azioni, invece, l'unico messaggio che il server invia in risposta al client è un

messaggio di saluto.

```
1  public ConnectionServerThread(Socket s){this.socket = s;}
2
3  private String launchIces(String user){
4      //  when an authenticated client connects
5      //  launch its personal radio source instance by calling external program ices2
6      boolean found = false;
7      for(int i=0; i<Main.processes.size(); i++){
8              if(((String)(Main.processes.get(i)[0])).equalsIgnoreCase(user)){
9                      found = true;
10             }
11     }
12     if(!found){
13             RunCommand ices = new RunCommand("ices2"+
14                                                    Main.streamConfDir+"ices-"+user+".xml");
15             ices.start();
16             Main.processes.add(new Object[]{user,ices});
17     }
18     return Main.icecastServerAddress+user+".ogg";
19 }
20
21 public void run(){
22     try{   ...
23             ObjectOutputStream out = new
   ObjectOutputStream(this.socket.getOutputStream());
24             ObjectInputStream in = new ObjectInputStream(this.socket.getInputStream());
25             out.writeObject("Welcome");
26             out.flush();
27
28             ClientObservation ob = (ClientObservation)in.readObject();
29
30             if(ob.getAction().equalsIgnoreCase("connect")){
31                     out.writeObject(new String[]{this.launchIces(ob.getUser())});
32                     out.flush();
33             }else{
34                     out.writeObject("Good bye");
35                     out.flush();
36             }
37             this.socket.close();
38
39             String action = ob.getAction();
40             if(action.equalsIgnoreCase("connect")){return;}
41             else if(action.equalsIgnoreCase("kill")){
42                     for(int i=0; i<Main.processes.size(); i++){
43             if(((String)Main.processes.get(i)[0]).equalsIgnoreCase(ob.getUser())){
44                     //kill ices2 instance for the current user
45                     ((RunCommand)Main.processes.remove(i)[1]).destroy();
46                             }
47                     }
48                     return;
49             }
50             ...
51             Track track = new Track();
52             track.setTitle(ob.getTitle());
53             track.setArtist(ob.getArtist());
54             track.setAlbum(ob.getAlbum());
55             track.setGenre(ob.getGenre());
56
57             // launch ObservationSender Thread, which creates and sends
58             //the observation to the SICS Module
59             ObservationSender sender = new ObservationSender(ob.getUser(),
60                                                             ob.getAction(),track);
61             sender.start();
62
63     }catch(Exception ex){...}
64 }
```

Listato 3.2: Codice thread ConnectionServerThread

A questo punto, con la chiusura della socket, termina la connessione con l'applicazione

client. Il thread continua la sua esecuzione verificando nuovamente l'azione ricevuta. Se l'azione ricevuta è *connect ConnectionServerThread* termina la propria esecuzione poiché avrà già eseguito l'avvio del source client illustrato poc'anzi. Nel caso venga ricevuta l'azione *kill* il compito del server è quello di terminare il source client associato all'utente in modo da evitare sprechi di risorse. Una volta terminato *ices2* invocando il metodo *destroy()* sul processo precedentemente avviato e referenziato in una apposita lista, anche *ConnectionServerThread* potrà terminare la propria esecuzione. In tutti gli altri casi, ovverosia quando le azioni ricevute sono *listen*, *accept* o *reject*, viene avviato il thread *ObservationSender* che avrà il compito strutturare le informazioni ricevute in modo adeguato per essere utilizzate dal modulo SICS.

Interazione con il modulo SICS

Il modulo Core interagisce con il modulo SICS in tre occasioni. Lo fa, una prima volta, nel momento in cui il server viene avviato. In questa fase l'interazione si limita ad una semplice configurazione del modulo SICS. Il secondo momento si verifica ogni qual volta il server invia al modulo SICS le informazioni ricevute dal client sotto forma di "osservazioni". Il terzo momento, infine, si verifica quando il server richiede al modulo SICS di generare una raccomandazione in base alle osservazioni precedentemente inviategli. La classe *SicsConfigSetup* agirà da interfaccia tra il modulo Core ed il modulo SICS in tutte le suddette occasioni. Essa, infatti, implementa e mette a disposizione del modulo Core i metodi per poter gestire in modo semplice il modulo SICS, astraendo così il primo dai dettagli tecnici del secondo.

La fase di configurazione del modulo SICS avviene nella classe *Main*. In primo luogo viene istanziato l'oggetto *sics* di tipo *SicsConfigSetup*. Il parametro passato ad *sisc* rappresenterà l'identificatore della specifica configurazione che SICS andrà ad utilizzare. Il *Main* invoca quindi il metodo *setupSics()*. Questo metodo, tra le altre cose, crea i gruppi, gli attori e le associazioni attore-gruppo di cui SICS necessita e li invia al modulo "storage" di SICS per la loro memorizzazione. I principali passaggi di questo metodo sono riportati nel Listato 3.3.

Per rendere fruibili dal modulo SICS le informazioni ricevute dall'applicazione client, il Core si serve della classe *ObservationSender*. In realtà si tratta di un thread che viene lanciato ogni volta in cui l'invio di una osservazione si rende necessario. Una volta avviato, il thread genera un oggetto di tipo *ObservationDTO* che realizza il concetto di "osservazione" in Cultura Implicita. Ogni osservazione, nello specifico caso del sistema implementato, sarà costituita da una azione (*listen*, *accept* o *reject*), da una serie di attributi di tale azione (le informazioni contestuali del brano ascoltato) e dall'attore che ha eseguito tale azione (il nome utente). Queste informazioni vengono qui generate ed aggiunte all'osservazione. Il metodo *sendObservation()* messo a disposizione del modulo Composer di SICS è quindi invocato per effettuare l'invio dell'osservazione.

La fase di richiesta al modulo SICS di raccomandazioni, viene eseguita anch'essa dal thread *ObservationSender*, invocando il metodo *getRecommendations()* che prende come parametro un oggetto di tipo *ObservationDTO*. Saranno quindi le osservazioni ricevute, ad

essere passate come argomento a tale metodo. Il metodo *getRecommendations()* è implementato nella classe *SicsConfigSetup* e svolge il compito di analisi delle raccomandazioni ricevute dal modulo SICS al fine di restituire le tracce musicali candidate ad essere accodate alla playlist delle raccomandazioni personali di ciascun utente per essere poi trasmesse all'applicazione client dalla piattaforma di streaming.

```
65 private void setupSics(){
66     ...
67     ComposerConstants constants = null;
68     try{   //retrieve configured composer constants configured in
69            //<sicsInstanceConfiguration.xml>
70            constants =
   ComposerConstantsGetter.getComposerConstants(sics.getIdentifier());
71     }catch(SicsException se){...}
72
73     //create groups G and G'
74     GroupDTO group1 = this.sics.createGroup(constants.getGGroupName());
75     GroupDTO group2 = this.sics.createGroup(constants.getGPrimeGroupName());
76
77     //add groups to Store
78     this.sics.addGroupToStore(group1);
79     this.sics.addGroupToStore(group2);
80
81     //create actors
82     List<User> usersList = database.getUsers();
83     List<ActorGroupDTO>actorsList = new Vector<ActorGroupDTO>(usersList.size());
84     for(int i=0; i<usersList.size(); i++){
85            actorsList.add(sics.createActor(usersList.get(i).getUsername()));
86     }
87     ...
88     //associate actors to g_group and g_prime_group and add them to the Store
89     for(int i=0; i<actorsList.size(); i++){
90            if(i%2 == 0)
91                   sics.addActorToStore(actorsList.get(i), group1);
92            else
93                   sics.addActorToStore(actorsList.get(i), group2);
94     }
95 }
```

Listato 3.3: Codice metodo setupSics()

A causa di un parametro impostato nella configurazione del modulo SICS che definisce un valore soglia (c*fr.* sezione 3.2.3), esso, in risposta alla richiesta di raccomandazioni, ritornerà non solo quella che avrà ottenuto il valore di similarità maggiore rispetto all'osservazione inviata, ma tutte quelle il cui valore di similarità risulterà essere superiore al valore soglia. Il thread *ObservationSender* termina la propria esecuzione non appena passa le raccomandazioni ricevute al thread *PostProcessor* il quale ne svolge un filtraggio al fine di selezionare il brano musicale che sarà effettivamente accodato alla playlist personale dell'utente. Il filtraggio consiste nel selezionare, fra tutti quelli proposti, un brano che non sia già stato raccomandato all'utente. Nel caso tutti i brani musicali proposti fossero già presenti nella playlist, quello da proporre all'utente finale, sarà scelto in modo casuale dalla libreria musicale a patto che sia dello stesso genere di uno di quelli proposti dal modulo SICS.

In entrambe i casi, l'operazione di accodamento della traccia raccomandata viene svolta dal metodo *addItem()* della classe *Enqueuer* invocato direttamente dal thread *PostProcessor* il quale, quindi, termina la propria esecuzione.

Per una migliore comprensione di quanto sopra descritto, i passaggi più significativi del thread *ObservationSender* e dei metodi messi a disposizione dalla classe *SicsConfigSetup* sono stati riportati rispettivamente nei listati 3.4, 3.6 e 3.5.

```
1  public void run(){
2       ...
3       //create the actor who executed the action
4       ActorGroupDTO actor  = this.sics.createActor(this.actorName);
5       //generate action attributes
6       AttributeDTO att_Title = this.sics.generateAttribute("title",
   this.track.getTitle());
7       AttributeDTO att_Artist = this.sics.generateAttribute("artist",
   this.track.getArtist());
8       AttributeDTO att_Album = this.sics.generateAttribute("album",
   this.track.getAlbum());
9       AttributeDTO att_Genre = this.sics.generateAttribute("genre",
   this.track.getGenre());
10      AttributeDTO[] attributes = {att_Genre, att_Album, att_Artist, att_Title};
11      //generate the observation to be sent to the SICS
12      ObservationDTO ob = this.sics.generateObservation(this.actionName, attributes,
   actor);
13      try{
14              ...
15              //send the observation to the SICS Module
16              this.sics.getSicsComposer().sendObservation(ob);
17              //ask for recommendations
18              new PostProcessor(this.actorName, this.sics.getReccomendations(ob)).start();
19              ...
20      }catch(SicsException e){...}
21
22 }
```

Listato 3.4: Metodo run() del thread ObservationSender

```
1  public ObservationDTO generateObservation(String actionName, AttributeDTO[] attributes,
   ActorGroupDTO actor){
2       ExecutedActionDTO execAction = new ExecutedActionDTO();
3       execAction.setName(actionName);
4       execAction.setTimestamp(new Timestamp(new Date().getTime()));
5       for(int i=0; i<attributes.length; i++)
6               execAction.addAttribute(attributes[i]);
7       execAction.addActor(actor);
8
9       ObservationDTO observation = new ObservationDTO();
10      observation.setIdentifier(this.defaultIdentifier);
11      observation.setExecutedAction(execAction);
12
13      return observation;
14 }
```

Listato 3.5: Metodo generateObservation() della classe SicsConfigSetup

```
15 public Vector<Track> getRecommendations(ObservationDTO observation){
16     Vector<Track> tracks = null;
17     Track currentTrack = null;
18     try {
19       //ask the composer module for recommendations
20       ResultDTO suggestion = this.getSicsComposer().performQuery(observation);
21       if(suggestion.getScenes() == null || suggestion.getScenes().isEmpty())
22            System.out.println("WARN - There are 0 suggested scenes");
23       else{
24            tracks = new Vector<Track>();
25            SceneDTO scene = null;
26            Collection<SceneDTO> result = (Collection<SceneDTO>)suggestion.getScenes();
27            Iterator<SceneDTO> it_scenes = result.iterator();
28            while(it_scenes.hasNext()){
29                   currentTrack = new Track();
30                   //iterate through results
31                   ...
32
33                   if(name.equalsIgnoreCase(new String("title"))){
34                          currentTrack.setTitle((String)value);
35                   }else if(name.equalsIgnoreCase(new String("artist"))){
36                          currentTrack.setArtist((String)value);
37                   }else if(name.equalsIgnoreCase(new String("album"))){
38                          currentTrack.setAlbum((String)value);
39                   }else if(name.equalsIgnoreCase(new String("genre"))){
40                          currentTrack.setGenre((String)value);
41                   }
42                   ...
43                   tracks.add(currentTrack);
44            }
45            ...
46     } catch (SicsException e) {...}
47     //if there are no suggested scenes null, will be returned
48     return tracks;
49 }
```

Listato 3.6: Metodo getRecommendations() della classe SicsConfigSetup

3.2.3 Il modulo SICS

Le funzionalità di elaborazione delle raccomandazioni basate su Cultura Implicita, sono messe a disposizione del sistema dal modulo SICS. Questo modulo è costituito dalle sole librerie IC-Servrice. IC-Service, presentato nella sezione 2.4.1, implementa un sistema per il supporto di Cultura Implicita secondo un'architettura orientata al servizio. Poiché il sistema sviluppato non presenta un'architettura di elevata complessità, è stato scelto di integrare IC-Service (in versione 1.6.1) come libreria esterna.

Il requisito dell'applicazione server che questo modulo affronta in modo specifico è il secondo, il quale consiste nella capacità di elaborare le informazioni ricevute dai client in modo da produrre raccomandazioni che soddisfino le preferenze musicali dei singoli utenti.

Il metodo utilizzato per l'interfacciamento tra questo modulo e il modulo Core è già stato delineato in modo approfondito nella sezione precedente. In quella corrente si vuole, invece, descrivere come il dominio di applicazione è stato modellato nei termini di Cultura Implicita, ovvero di come SICS è stato configurato al fine di effettuare raccomandazioni musicali personalizzate. Per una presentazione esaustiva dei concetti teorici che la trattazione svolta in questa sezione presuppone, si rimanda alla sezione 2.4 e, soprattutto, ai documenti in essa referenziati.

Modellazione del dominio (problema)

SICS richiede che il problema venga modellato in termini di attori (o agenti), oggetti, azioni ed attributi. Nel caso specifico, il problema da modellare è così descrivibile: una serie di utenti ascoltano, in sequenza, delle tracce musicali in forma di stazione radio personale e ne esprimono un giudizio circa il gradimento. Data questa definizione del problema di dominio, la mappatura è stata così inizialmente delineata: ogni utente del sistema sarebbe stato considerato un attore, i giudizi espressi dagli utenti sarebbero corrisposti alle azioni e, infine, ogni traccia musicale sarebbe stata considerata come un oggetto. Inoltre, ogni oggetto avrebbe posseduto gli attributi titolo, artista, album e genere.

Poiché, però, nella suddetta modellazione la traccia musicale costituisce l'unico oggetto del sistema su cui gli attori eseguono le azioni, in un secondo momento è stato deciso di non considerare la traccia musicale come oggetto ma di rappresentarla unicamente mediante gli attributi delle singole azioni ottenendo quindi la mappatura presentata nella seguente tabella.

problema di dominio	***modello SICS***	***esempio***
`utente`	`attore`	`john`
`giudizio su traccia musicale`	`azione(titolo, artista, album, genere)`	`accept("Now We Are Free", "Hans Zimmer", "Gladiator", "Soundtrack")`

Tabella 3.1: Mappatura del problema di dominio in SICS

Le azioni contemplate dal modello sono le tre azioni introdotte nella descrizione dell'applicazione client, ossia *accept*, *reject* e *listen*. Mentre l'azione *listen* viene eseguita in modo automatico ogni qual volta il client inizia la riproduzione di un brano musicale, la scelta tra *accept* e *reject* viene lasciata a totale discrezione dell'utente, il quale può anche decidere di non esprimere alcun giudizio esplicito sulla traccia ascoltata semplicemente non eseguendo alcuna azione. Questo particolare evento non è stato associato ad alcuna azione in SICS, ma un tentativo di contemplarlo sarà svolto in nella fase di valutazione di similarità tra le azioni. Gli attributi di ciascuna azione identificano il brano musicale a cui l'azione compiuta dall'utente (attore) si riferisce. Gli attori invece vengono identificati dal nome utente di ciascun utente del sistema. Sebbene la modellazione effettuata non preveda, al momento, alcun tipo di attributo per l'attore, essa risulta facilmente estensibile nel caso questo si renda necessario.

Il vincolo culturale per il problema di dominio descritto, verrà formalmente presentato in seguito. Ciò che comunque esso dovrà esprimere sarà ovviamente il fatto che le tracce raccomandate ai singoli utenti dovranno essere da loro gradite, ovvero che le azioni da essi eseguite sui brani musicali proposti dovranno essere delle azioni *accept*.

Si fa presente, infine, che nella modellazione effettuata tutti gli utenti appartengono ad un singolo gruppo in SICS. In altre parole i gruppi G e G', previsti da SICS, saranno coincidenti. Ciò significa che le azioni compiute da ciascun utente saranno osservate e

prese in considerazione per la generazione delle raccomandazioni per tutti gli utenti, compreso egli stesso. Questa scelta è stata effettuata in quanto, in questo particolare dominio, non esiste un gruppo di attori più esperti (oggetto delle osservazioni) la cui conoscenza debba essere trasmessa ad un altro gruppo di attori meno esperti (oggetto delle raccomandazioni), bensì tutti concorrono allo stesso modo alla creazione della cultura di comunità.

Configurazione di base

La configurazione di IC-Service è molto articolata e viene mantenuta su una serie di file in formato XML. Apportando modifiche ad uno o più di questi file è possibile configurare sia SICS a livello generale che ciascuno dei suoi moduli. In questa sezione sarà illustrata la configurazione relativa soltanto agli elementi più importanti e significativi, rimandando alla documentazione di IC-Service [31] per ogni ulteriore dettaglio.

Il primo passo consiste nell'indicare ad IC-Service "come e dove" le informazioni da elaborare debbano essere salvate; con "informazioni" si intende le osservazioni, gli utenti/gruppi e le regole. Le possibilità offerte sono due: l'utilizzo di una base di dati, o l'utilizzo di file xml. La prima opzione è indicata nel caso in cui si preveda effettuare un numero molto elevato di osservazioni, ed è generalmente adatta a sistemi complessi. Per il sistema realizzato è stata scelta la seconda opzione in quanto il numero delle osservazioni non è elevato ed inoltre essa richiede una quantità inferiore di risorse rispetto alla memorizzazione su database. I due file che assolvono tale scopo sono *sics_groupActor.xml* per la memorizzazione degli attori e dei gruppi e *sics_store.xml* per la memorizzazione delle osservazioni.

Merita una particolare attenzione la configurazione di alcuni dei parametri presenti nel file *sicsInstanceConfiguration.xml*. Tali parametri, relativi al modulo Composer di SICS, sono l'identificatore di configurazione, il valore della soglia minima di similarità i nomi dei gruppi G e G'. Il sistema realizzato utilizza il nome *webRadio* come identificatore di configurazione, al quale corrispondono *0.75* come valore della soglia di similarità (*cfr.* sezione "Similarità") e *g_group* come nome di entrambe i gruppi G e G'. Per comprendere il significato della soglia di similarità e del valore impostato si faccia riferimento alla sotto-sezione denominata "Similarità" riportata in seguito.

Gli altri file che assumono una particolare importanza per quanto riguarda la configurazione di SICS sono il file delle regole della teoria di dominio *sics_rules.xml* e quello delle regole di similarità *similarity_config.xml.*

Teoria e regole

Lo scopo primario di SICS è che si verifichi il cosiddetto fenomeno di Cultura Implicita. Tale fenomeno ha luogo nel momento in cui gli attori, oggetto delle raccomandazioni, eseguono azioni consistenti con il vincolo culturale denominate anche "azioni culturali". Il vincolo culturale, ovverosia la teoria di dominio, per il problema modellato è formalmente espresso dalla seguente relazione:

$$\forall\ u \in Users, t \in Tracks:$$
$$listen(u,t) \rightarrow accept(u,t)$$
$$accept(u,t) \rightarrow listen(u,t)$$
$$accept(u,t) \wedge reject(u,t) \rightarrow listen(u,t)$$

Relazione 3.1: Vincolo culturale

Sebbene il concetto che il vincolo culturale deve catturare sia semplice, in quanto deve esprimere il fatto che i brani musicali proposti ad un singolo utente debbano essere da lui graditi (e quindi "accettati"), la teoria sopra riportata è un poco più complessa ed è il frutto dei risultati ottenuti testando differenti soluzioni le quali verranno di seguito brevemente discusse.

La prima regola prevede che un utente accetti la traccia musicale a lui propostagli. La seconda regola esprime il concetto opposto, ovvero che se un utente accetta una traccia musicale, allora il sistema dovrà proporgli, per l'ascolto, una traccia simile. La terza regola, infine, formalizza il fatto che se un utente, in un determinato momento, accetta un brano musicale e, in un altro momento, ne rifiuta uno simile, allora il sistema potrà comunque riproporgli un altro brano simile al primo.

Per completezza si vogliono di seguito presentare i motivi per i quali altre due possibili teorie non sono state utilizzate.

$$\forall\ u \in Users, t \in Tracks:$$
$$listen(u,t) \rightarrow accept(u,t)$$

Relazione 3.2: Prima teoria

$$\forall\ u \in Users, t \in Tracks:$$
$$accept(u,t) \rightarrow listen(u,t)$$

Relazione 3.3: Seconda teoria

La prima teoria, espressa dalla Relazione 3.2, non si è rilevata adeguata in quanto genera raccomandazioni musicali di brani appartenenti ai generi musicali rifiutati in più circostanze e, comunque, non graditi all'utente.

La seconda teoria (Relazione 3.3), è stata scartata in quanto nel momento in cui un utente avesse rifiutato un genere musicale, indipendentemente dalle precedenti azioni compiute, il sistema non avrebbe più generato raccomandazioni musicali di quel tipo. Questo, nel caso peggiore, avrebbe portato allo stallo del sistema in quanto nessun utente avrebbe più ricevuto come raccomandazione alcun brano musicale.

Si noti che le due teorie sopra descritte sono state inglobate come singole regole nella teoria utilizzata dal sistema per esprimere il vincolo culturale descritto dalla Relazione 3.1. Infatti, combinate con la terza regola, sono risultate essere adeguate.

I listati 3.7 e 3.8 illustrano rispettivamente la prima e la terza regola del vincolo culturale espresso dalla Relazione 3.1. La seconda regola corrisponde alla inversione della prima.

```
1  <!-- listen(u,t) => accept(u,t) -->
2  <rule identifier="webRadio">
3     <antecedents>
4        <action-predicate>
5           <action-rule name="listen" timestamp="*"
                         timestamp_type="variable" name_type="constant">
6              <actors> <actor-rule name="_actor" name_type="variable" /> </actors>
7              <attributes>
8                 <attribute-rule type="String" variable_type="variable" name="title">
9                    _title
10                </attribute-rule>
11                <attribute-rule type="String" variable_type="variable" name="artist">
12                   _artist
13                </attribute-rule>
14                <attribute-rule type="String" variable_type="variable" name="album">
15                   _album
16                </attribute-rule>
17                <attribute-rule type="String" variable_type="variable" name="genre">
18                   _genre
19                </attribute-rule>
20             </attributes>
21          </action-rule>
22          <scene-rule />
23       </action-predicate>
24    </antecedents>
25    <consequents>
26       <action-predicate>
27          <action-rule name="accept" timestamp="*"
                         timestamp_type="variable" name_type="constant">
28             <actors> <actor-rule name="_actor" name_type="variable" /> </actors>
29             <attributes>
30                <attribute-rule type="String" variable_type="variable" name="title">
31                   _title
32                </attribute-rule>
33                <attribute-rule type="String" variable_type="variable" name="artist">
34                   _artist
35                </attribute-rule>
36                <attribute-rule type="String" variable_type="variable" name="album">
37                   _album
38                </attribute-rule>
39                <attribute-rule type="String" variable_type="variable" name="genre">
40                   _genre
41                </attribute-rule>
42             </attributes>
43          </action-rule>
44          <scene-rule />
45       </action-predicate>
46    </consequents>
47 </rule>
```

Listato 3.7: prima regola della teoria

```
1  <!-- accept(u,t) && reject(u,t) => listen(u,t) -->
2  <rule identifier="webRadio">
3    <antecedents>
4      <action-predicate>
5        <action-rule name="accept" timestamp="*" timestamp_type="variable"
                      name_type="constant">
6          <actors> <actor-rule name="_actor" name_type="variable" /> </actors>
7          <attributes>
8            <attribute-rule type="String" variable_type="variable" name="title">
9              *
10           </attribute-rule>
11           <attribute-rule type="String" variable_type="variable" name="artist">
12             *
13           </attribute-rule>
14           <attribute-rule type="String" variable_type="variable" name="album">
15             *
16           </attribute-rule>
17           <attribute-rule type="String" variable_type="variable" name="genre">
18             _genre
19           </attribute-rule>
20         </attributes>
21       </action-rule>
22       <scene-rule />
23     </action-predicate>
24     <action-predicate>
25       <action-rule name="reject" timestamp="*" timestamp_type="variable"
                      name_type="constant">
26         <actors> <actor-rule name="_actor" name_type="variable" /> </actors>
27         <attributes>
28           <attribute-rule type="String" variable_type="variable" name="title">
29             *
30           </attribute-rule>
31           <attribute-rule type="String" variable_type="variable" name="artist">
32             _artist
33           </attribute-rule>
34           <attribute-rule type="String" variable_type="variable" name="album">
35             *
36           </attribute-rule>
37           <attribute-rule type="String" variable_type="variable" name="genre">
38             _genre
39           </attribute-rule>
40         </attributes>
41       </action-rule>
42       <scene-rule />
43     </action-predicate>
44   </antecedents>
45   <consequents>
46     <action-predicate>
47       <action-rule name="listen" timestamp="*" timestamp_type="variable"
                      name_type="constant">
48         <actors> <actor-rule name="_actor" name_type="variable" /> </actors>
49         <attributes>
50           <attribute-rule type="String" variable_type="variable" name="title">
51             *
52           </attribute-rule>
53           <attribute-rule type="String" variable_type="variable" name="artist">
54             _artist2
55           </attribute-rule>
56           <attribute-rule type="String" variable_type="variable" name="album">
57             *
58           </attribute-rule>
59           <attribute-rule type="String" variable_type="variable" name="genre">
60             _genre
61           </attribute-rule>
62         </attributes>
63       </action-rule>
64       <scene-rule />
65     </action-predicate>
66   </consequents>
67 </rule>
```

Listato 3.8: terza regola della teoria

Come si nota, ogni regola è separata in due grossi blocchi rappresentati dagli elementi <*antecedents*> e <*consequents*>. Questi blocchi corrispondono rispettivamente agli "antecedenti" e ai "conseguenti" di ciascuna regola definita nella relazione di vincolo culturale ovverosia, rispettivamente, alla prima e alla seconda parte. Per esempio, nel caso della prima regola, l'elemento <*antecedents*> corrisponde a *listen(u,t)*, mentre l'elemento <*consequent*> corrisponde a *accept(u,t)*.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <observations xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="sics.xsd">
3    <observation identifier="webRadio">
4      <action name="listen" timestamp="1213961982302">
5        <actors>
6          <actor name="john" />
7        </actors>
8        <attributes>
9          <attribute name="album" type="String">Gladiator</attribute>
10         <attribute name="artist" type="String">Hans Zimmer</attribute>
11         <attribute name="genre" type="String">Soundtrack</attribute>
12         <attribute name="title" type="String">Now We Are Free</attribute>
13       </attributes>
14     </action>
15     <scene>
16       <actions>
17         <action name="listen">
18           <attributes>
19             <attribute name="album" type="String">Gladiator</attribute>
20             <attribute name="artist" type="String">Hans Zimmer</attribute>
21             <attribute name="genre" type="String">Soundtrack</attribute>
22             <attribute name="title" type="String">Now We Are Free</attribute>
23           </attributes>
24         </action>
25       </actions>
26     </scene>
27   </observation>
28   <observation identifier="webRadio">
29     <action name="accept" timestamp="1213961985546">
30       <actors>
31         <actor name="john" />
32       </actors>
33       <attributes>
34         <attribute name="album" type="String">Gladiator</attribute>
35         <attribute name="artist" type="String">Hans Zimmer</attribute>
36         <attribute name="genre" type="String">Soundtrack</attribute>
37         <attribute name="title" type="String">Now We Are Free</attribute>
38       </attributes>
39     </action>
40     <scene>
41       <actions>
42         <action name="accept">
43           <attributes>
44             <attribute name="album" type="String">Gladiator</attribute>
45             <attribute name="artist" type="String">Hans Zimmer</attribute>
46             <attribute name="genre" type="String">Soundtrack</attribute>
47             <attribute name="title" type="String">Now We Are Free</attribute>
48           </attributes>
49         </action>
50       </actions>
51     </scene>
52   </observation>
53 </observations>
```

Listato 3.9: esempio di osservazioni memorizzate nel file sics_store.xml

Come è già stato spiegato, invece di rappresentare le tracce musicali come oggetti, la modellazione prevede che a tale scopo vengano utilizzati gli attributi delle singole azioni.

Si nota infatti che le regole definiscono gli attributi *title*, *artist*, *album* e *genre* all'interno dell'elemento <*action-rule*> che rappresenta un'azione. Le regole prevedono l'esistenza di costanti e variabili. Queste ultime, espresse nella forma *_nomeVariabile*, vengono sostituite dai valori attuali nelle osservazioni e creano il legame tra gli antecedenti e i conseguenti della teoria. Il Listato 3.9, a titolo esemplificativo, riporta parte del contenuto del file *sics_store.xml* dove le osservazioni effettuate dell'Observer vengono memorizzate. Si noti la corrispondenza tra le osservazioni e le regole della teoria.

Le regole così definite vengono utilizzate dal modulo Composer di SICS (in particolare da CAF e SP) per generare delle scene da proporre agli utenti nelle quali l'azione attesa sia una "azione culturale".

Similarità

La valutazione di similarità tra le osservazioni, è il processo con il quale il modulo SICS decide quali tra le scene prodotte dal modulo Composer siano le più adatte ad essere proposte agli attori.

Come la teoria, anche la similarità tra le osservazione dipende dal dominio e viene espressa mediante delle regole specificate nel file *similarity_config.xml*. Le regole di similarità permettono di specificare quale "peso" attribuire a ciascuno degli elementi modellati (attori, azioni, regole ed attributi) al fine di computare il valore totale di similarità tra due osservazioni. L'algoritmo di valutazione della similarità implementato da SICS rende possibile, inoltre, definire eccezioni e priorità in modo da permettere la creazione di regole anche molto complesse. I dettagli algoritmici per il calcolo della similarità totale non verranno qui trattati; si rimanda alla documentazione di IC-Service [32] per una esaustiva presentazione a riguardo. Per una generale comprensione, invece, ciò che risulta fondamentale è sapere che la similarità totale tra due osservazioni può assumere un valore compreso tra *0* e *1*, dove un valore di similarità pari a *0* significa che le due osservazioni vengono considerate totalmente differenti, mentre un valore di similarità pari a *1* sta a significare che le due osservazioni saranno considerate equivalenti. Il valore totale viene computato sulla base dei "pesi" assegnati ai singoli elementi modellati.

Per lo specifico problema di dominio è stato deciso di effettuare il calcolo della similarità soltanto sulla base delle azioni eseguite e dei loro attributi (che identificano i singoli brani musicali). Il *50%* del valore totale di similarità dipenderà dall'azione eseguita, mentre il restante *50%* sarà calcolato in base alla similarità dei brani musicali. Tale similarità si baserà unicamente sugli attributi titolo, artista, album e genere che sono estratti direttamente dalle proprietà dei file musicali (i cosiddetti *tags*); è perciò fondamentale che tutti i brani che compongono la libreria musicale siano stati correttamente commentati.

Si noti che il fatto di computare, oltre alla similarità tra le azioni eseguite, anche quella tra i commenti dei brani musicali ascoltati, equivale ad effettuare una basilare analisi del contenuto. Il *50%* del valore totale di similarità, assegnato agli attributi musicali, viene così ripartito: il *30%* dipenderà dal genere, il *19%* dall'artista, il *0%* dall'album ed il restante *1%* dal titolo del brano musicale.

Per quanto riguarda la similarità tra le azioni, rispetto all'azione *accept* le altre saranno

valutate come segue. L'azione *reject* verrà considerata totalmente differente quindi la similarità con *accept* sarà pari a *0*. L'azione *listen*, invece, avrà un valore di similarità con *accept* pari a *0.3* (*30%*) al fine di tentare di contemplare nel modello il caso in cui l'utente non esprima alcun giudizio esplicito, sia esso positivo o negativo, sul gradimento delle tracce ascoltate.

E' stato precedentemente anticipato che un importante parametro da configurare nel file *sicsInstanceConfiguration.xml*, è il valore dell'elemento <*threshold*>. Questo valore rappresenta la soglia minima di similarità richiesta affinché le scene prodotte dal modulo Composer di SICS possano rientrare tra quelle raccomandabili agli attori del sistema. E' dunque di fondamentale importanza impostare un valore adeguato in base anche ai singoli "pesi" assegnati a ciascun elemento del modello descritti poc'anzi.

Per la configurazione di SICS utilizzata è stato impostato un valore di soglia pari a *0.75*. Ciò significa che un attore riceverà solo quelle raccomandazioni in cui il valore di similarità tra la scena proposta e le scene presenti nelle osservazioni passate risulterà pari o maggiore a tale valore.

```
<instances>
      <instance-actions>
        <instance-action name="accept" valueForName="0.50" valueForAttributes="0.50"
          valueForObjects="0.00" valueForActors="0.00" valueForTimestamp="0.00"
          caseInsensitiveMatching="true" useOnlyRegularExpression="false"
          processRegularExpression="false"
          processRegularExpressionAsOr="true">
          <regExp-ForName>^*</regExp-ForName>
          <name-exceptions>
            <name-exception nameToMatch="listen" valueToSubtract="0.20"
              caseInsensitiveMatching="true" matchingAsRegularExpression="false" />
            <name-exception nameToMatch="reject" valueToSubtract="0.50"
              caseInsensitiveMatching="true" matchingAsRegularExpression="false" />
          </name-exceptions>
          <attributes>
            <instance-attribute name="title" value="0.01"
              performMatchingAsString="true" caseInsensitiveMatching="true"
              useOnlyRegularExpression="false" processRegularExpression="false"
              processRegularExpressionAsOr="false">
              <regExp-ForValue>*</regExp-ForValue>
            </instance-attribute>
            <instance-attribute name="album" value="0.00"
              performMatchingAsString="true" caseInsensitiveMatching="true"
              useOnlyRegularExpression="false" processRegularExpression="false"
              processRegularExpressionAsOr="false">
              <regExp-ForValue>*</regExp-ForValue>
            </instance-attribute>
            <instance-attribute name="artist" value="0.19"
              performMatchingAsString="true" caseInsensitiveMatching="true"
              useOnlyRegularExpression="false" processRegularExpression="false"
              processRegularExpressionAsOr="false">
              <regExp-ForValue>*</regExp-ForValue>
            </instance-attribute>
            <instance-attribute name="genre" value="0.30"
              performMatchingAsString="true" caseInsensitiveMatching="true"
              useOnlyRegularExpression="false" processRegularExpression="false"
              processRegularExpressionAsOr="false">
              <regExp-ForValue>*</regExp-ForValue>
            </instance-attribute>
          </attributes>
        </instance-action>
      </instance-actions>
    </instances>
```

Listato 3.10: Regole di similarità per il problema di dominio

Per comprendere meglio come la similarità tra le osservazioni venga computata nella specifica configurazione utilizzata dal sistema, si consideri il seguente sommario.

Con il valore di soglia di *0.75*, due osservazioni vengono considerate simili nei casi in cui:

a) i nomi delle azioni siano uguali tra loro, così come il generi musicali. In questo caso il valore totale di similarità sarà pari a *0.50 + 0.30 = 0.8*.

b) le azioni a confronto siano *accept* e *listen* e i due attributi genere e artista siano entrambi uguali tra le due osservazioni determinando quindi una similarità totale pari a *0.3 + 0.30 + 0.19 = 0.79*.

Alla luce di quanto sopra descritto, si nota che in entrambi e casi il peso assegnato al genere musicale si rivela determinante al fine di stabilire una relazione di similarità. Si tratta di una scelta voluta, in quanto l'analisi del contenuto dei brani musicali utilizzata dal sistema, considera unicamente i semplici attributi musicali e, in questo contesto, il genere gioca un ruolo discriminante di fondamentale rilevanza.

Il Listato 3.10 il illustra le principali regole di similarità configurate, mentre due esempi di valutazione della similarità tra osservazioni sono riportati nei listati 3.11 e 3.12.

```
1  Similarity between
   listen(actor=u0,title=Il ricordo delle tue mani,album=Occidente,artist=Rein, genre=Folk)
   and
   listen(actor=u2,title=Beginning of the End,album=Rust,artist=No, Really, genre=Folk)
   = sim(listen,listen) * 0.5
   + sim(u0,u2) * 0
   + sim('Il ricordo delle tue mani','Beginning of the End') * 0.01
   + sim(Occidente,Rust) * 0
   + sim(Rein,'No, Really') * 0.19
   + sim(Folk,Folk) * 0.3
   = 0.8
```

Listato 3.11: Valutazione di similarità tra due elementi simili

```
1  Similarity between
   listen(actor=u0,title=Il ricordo delle tue mani,album=Occidente,artist=Rein, genre=Folk)
   and
   accept(actor=u0,title=Tia Hody,album=Libre,artist=Abdou Day, genre=Reggae-Ska)
   = sim(listen,accept)*0.3
   + sim(u0,u0) * 0
   + sim('Il ricordo delle tue mani','Tia Hody') * 0.01
   + sim(Occidente,Libre) * 0
   + sim(Rein,'Abdou Day') * 0.19
   + sim(Folk,Reggae-Ska) * 0.3
   = 0.3
```

Listato 3.12: Valutazione di similarità tra due elementi dissimili

3.2.4 La piattaforma di streaming

La piattaforma di streaming è la componente del sistema che, avendo il compito di creare e trasmettere il flusso audio all'applicazione client a partire dai brani musicali raccomandati dal server, realizza effettivamente la funzione di una radio, andando quindi a soddisfare il

terzo requisito richiesto all'applicazione server.

Questa piattaforma, come già anticipato, è costituita da due applicazioni (il source client ed il server di streaming) che non sono incluse nell'applicazione server, ma necessitano di una installazione propria. La corretta installazione di queste due applicazioni sulla macchina in cui l'applicazione server sarà eseguita costituisce un requisito fondamentale per il corretto funzionamento dell'intero sistema. Source client e streaming server vengono di seguito presentati con maggior dettaglio.

Il source client

La creazione del flusso audio a partire da una qualche sorgente e l'invio di questo al server di streaming sono i due compiti assolti da questa componente. Per la generazione del flusso audio il source client può seguire due modalità differenti. La prima consiste nell'utilizzare, come sorgente audio, dei file musicali presenti sulla macchina dove il source client è in esecuzione. In questo caso viene generalmente creata una lista di esecuzione (denominata playlist) in cui sono sequenzialmente elencati i percorsi nel file system dei brani che si desidera trasmettere. Il source client, poi, creerà il flusso audio a partire da questa lista e lo invierà al server di streaming. La seconda modalità, invece, prevede che come flusso audio sia utilizzato direttamente quello in output dalla scheda audio della macchina. Questa modalità consente di eseguire il cosiddetto "live streaming" (streaming in diretta) trasmettendo per esempio il segnale del microfono missato con un sottofondo musicale piuttosto che il segnale di uno strumento musicale collegato alla scheda audio. Nel caso del sistema realizzato sarà utilizzata la prima modalità descritta.

Un'altra funzione che è generalmente assolta dal source client è quella di codificare, se necessario, il flusso audio in un formato adeguato ad essere trasmesso dal server di streaming ed eventualmente di ricampionarlo con una differente frequenza.

Ices2 (in versione 2.0.1) è il software che si è scelto adottare come source client. Si tratta di una semplice applicazione che è in grado di generare flussi audio codificati in formato *Ogg Vorbis* con entrambe le modalità descritte precedentemente, ed è compatibile con il server di streaming *Icecast2*. Poiché *ices2* non prevede la funzionalità di "transcoding" i file musicali utilizzati come sorgente audio devono anch'essi essere codificati in formato *Ogg Vorbis*.

Ices2 è configurabile attraverso un semplice file in XML nel quale, tra le altre impostazioni, vengono specificati l'indirizzo del server di streaming, la porta sulla quale sarà in esecuzione, il nome del "mountpoint" sul quale sarà trasmesso il flusso e il percorso della playlist da utilizzare. Il "mountpoint" costituisce una sorta di identificatore per il flusso audio.

Poiché *ices2* genera un unico flusso audio, ciascun utente del sistema è associato ad una specifica istanza (processo) di *ices2* che viene avviata dal modulo Core dell'applicazione server. Per ciascun utente, quindi, è necessario creare uno specifico file di configurazione di *ices2* ed una playlist. Questa procedura è assolta dal modulo Core nella prima fase di configurazione, dove, a partire da una copia "master" di un file di configurazione di *ices2*, vengono creati quelli specifici per ciascun utente sostituendo agli elementi <*input*> e

<*mount*> i valori corretti per la playlist (*john.pls*) e il “mountpoint” (*/john.ogg*) come illustrato nel Listato 3.13.

E' inoltre importante verificare che i valori degli elementi <*hostname*>, <*port*> e <*password*> corrispondano esattamente a quelli configurati nel server di streaming. Si rimanda alla documentazione ufficiale di *ices2* [33] per ogni ulteriore dettaglio.

Una istanza del processo *ices2* viene lanciata ogni qual volta un utente accede al sistema eseguendo l'azione *connect*, viene invece terminata nel momento in cui l'utente chiuderà la propria applicazione client generando quindi l'azione *kill*.

Ices2 è software libero, distribuito con licenza GPLv2 e reperibile sul sito web *http://www.icecast.org/ices.php*. *Ices2* è compatibile unicamente con i sistemi operativi *Unix-like* e questo implica che l'intera applicazione server debba essere eseguita su tale piattaforma, sebbene effettuando qualche modifica al codice sorgente dell'applicazione server sarà possibile renderla compatibile con altri source clients.

```
1  <?xml version="1.0"?>
2  <ices>
3    ...
4    <stream>
5      ...
6      <input>
7        <!-- playlist module settings -->
8        <module>playlist</module>
9        <param name="type">basic</param>
10       <param name="file">john.pls</param>
11       <param name="random">0</param>
12       <param name="restart-after-reread">0</param>
13       <param name="once">0</param>
14     </input>
15     ...
16     <instance>
17       <!-- Icecast2 server details -->
18       <hostname>localhost</hostname>
19       <port>8000</port>
20       <password>hackme</password>
21       <mount>/john.ogg</mount>
22       ...
23     </instance>
24   </stream>
25 </ices>
```

Listato 3.13: Configurazione del source client ices2

Il server di streaming

Icecast2 (in versione 2.3.1) è il software utilizzato dal sistema per realizzare il server di streaming. Un server di streaming ha il compito di mettere a disposizione di utenti remoti contenuti multimediali in forma di “flusso” attraverso una rete.

Icecast2 supporta la trasmissione di flussi audio codificati nei formati *Ogg Vorbis* su protocollo HTTP e di *MPEG audio layer 3* (*mp3*) e *AAC* su protocollo *SHOUTcast*; anche la trasmissione di flussi video nei formati *Ogg Theora* e *NSV* è supportata.

Poiché per il sistema realizzato si è scelto di fare uso di formati aperti e liberi come *Ogg*

Vorbis per i file musicali, anche i*cecast2* trasmetterà in questo formato su protocollo HTTP.

Come *ices2*, anche *icecast2* è configurabile mediante un semplice file XML. Sebbene i parametri di configurazione siano molti (per la cui comprensione si rimanda alla documentazione ufficiale di *icecast2* [34]), quelli su cui è necessario porre attenzione ai fini dell'utilizzo che ne fa il sistema realizzato sono una parte ristretta. Nella sezione <*limits*> i valori dei due elementi <*clients*> e <*sources*> rappresentano rispettivamente il numero massimo di clients (rappresentati principalmente dagli ascoltatori) supportati concorrentemente dal server di streaming e il numero massimo di source clients. Dato che il sistema realizzato prevede che ad ogni utente venga associata una radio personale, è stato attribuito il valore *100* ad entrambi gli elementi. Gli altri parametri da impostare riguardano la locazione del server e la porta su cui *icecast2* trasmetterà i flussi audio e le impostazioni di autenticazione dei source client. E' importante che i valori impostati corrispondano in modo esatto con quelli configurati nei source client.

L'indirizzo di uno specifico flusso audio trasmesso da *icecast2* sarà del tipo *http://hostname:port/mountpoint*. Nel sistema realizzato il flusso *http://localhost:8000/john.ogg* sarà quello rappresentante la radio personale dell'utente *john*.

Contrariamente a quanto avviene per il source client, *icecast2* viene avviato dal modulo Core dell'applicazione server una sola volta non appena il processo di configurazione di SICS viene concluso e, comunque, prima di avviare il thread per l'ascolto delle connessioni dei client.

Icecast2, infine, rende disponibili alcune informazioni di carattere sia statistico che tecnico accessibili alla pagina web: *http://hostname:port/*.

Come *ices2* anche *icecast2* è software libero distribuito con licenza GPLv2 e reperibile sul sito web *http://www.icecast.org/download.php*.

3.3 Applicazione Client

3.3.1 Requisiti

L'applicazione client rappresenta la componente mediante la quale ogni utente interagisce con il sistema. Il primo requisito che dovrà soddisfare è la capacità riprodurre i brani musicali che saranno raccomandati dal server e che verranno ricevuti come flusso audio remoto. Il secondo requisito previsto consiste nel rendere possibile all'utente di esprimere un feedback circa il gradimento del brano ascoltato e di visualizzare le informazioni contestuali dello stesso quali titolo, artista, album e genere musicale.

JOrbisPlayer, un software per la riproduzione di file audio (o flussi audio remoti) codificati con il codec *Ogg Vorbis*, è stato scelto come base per lo sviluppo del client. Questo software permetterà di soddisfare il primo requisito richiesto.

Le funzionalità di comunicazione con il server per l'invio del feedback e di altre informazioni di servizio sono state realizzate apportando modifiche alla base posta da

JOrbisPlayer. In questo modo anche il secondo requisito viene soddisfatto.

Per la decodifica vera e propria, *JOrbisPlayer* si serve delle librerie *JOrbis* [35] assieme alle quali viene distribuito. *JOrbis* è sviluppato totalmente in Java ed è distribuito come "software libero" (ed opensource) sotto licenza LGPL.

JOrbisPlayer può essere eseguito sia come applicazione java standard che come applet java. A causa delle limitate capacità di comunicazione che le applet java, per ragioni di sicurezza offrono, è stato scelto di utilizzare *JOrbisPlayer* nella sua versione di applicazione java standard. Per questo motivo l'intera applicazione client viene eseguita anch'essa come applicazione java standard.

```
1  java -jar ThesisPlayer.jar <username> <password> <serverAddress:port>
```

Listato 3.14: Avvio dell'applicazione client

Al momento dell'avvio dell'applicazione client devono essere specificati i seguenti tre parametri: nome utente, password e indirizzo del server (Listato 3.14). I primi due saranno utilizzati per effettuare l'autenticazione sul server. Il terzo, invece, specifica l'indirizzo della macchina sulla quale il server è in esecuzione e la porta che sarà utilizzata per la comunicazione. La correttezza dei parametri specificati è un requisito fondamentale per il funzionamento dell'applicazione client.

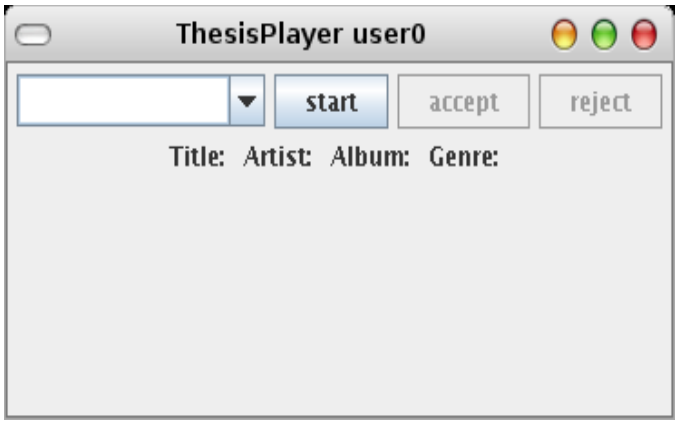

Fig. 3.2: Interfaccia utente all'avvio dell'applicazione Client

La Fig. 3.2 illustra l'interfaccia utente che compare una volta avviata l'applicazione client. L'utente controlla il client agendo sui pulsanti messi a disposizione dall'interfaccia mentre le informazioni contestuali del brano in ascolto saranno visualizzate nell'area centrale. Esse saranno costituite dal titolo del brano, il nome dell'artista, il titolo dell'album e il genere musicale. In questo momento nessuna connessione con il server è stata ancora stabilita e nessun brano è quindi in esecuzione. Per stabilire la connessione con il server è necessario agire una prima volta sul pulsante *start*. Il server, una volta effettuata l'autenticazione del client, risponderà inizializzando il flusso audio remoto e rendendolo disponibile

all'indirizzo che comparirà nel menù a tendina.

Fig. 3.3: Interfaccia utente durante l'ascolto dei brani musicali proposti

Agendo nuovamente sul pulsante *start* la riproduzione della radio personale avrà inizio e le informazioni contestuali del brano corrente saranno visualizzate (come illustrato in Fig. 3.3). Sarà possibile, in qualsiasi momento, arrestare la riproduzione agendo sul pulsante *stop* e quindi ristabilirla agendo nuovamente su *start*. Una volta avviata la radio personale, l'utente potrà esprimere il proprio giudizio riguardo il brano musicale in esecuzione agendo sui pulsanti *accept* o *reject*. Nel primo caso l'utente esprime un giudizio positivo "accettando" il brano proposto. Questo significa che tale proposta musicale rispecchia le preferenze dell'utente stesso. In caso contrario, agendo sul pulsante *reject*, l'utente esprime un giudizio negativo poiché il brano propostogli non rispecchia le sue preferenze musicali. Una volta espresso il proprio giudizio i pulsanti resteranno disabilitati fino alla conclusione del brano corrente; saranno, quindi, riabilitati nel momento in cui il brano successivo avrà inizio.

3.3.2 Implementazione

L'applicazione client è stata interamente implementata in linguaggio Java. La scelta è ricaduta su questo linguaggio di programmazione poiché anche le altre componenti del sistema utilizzano lo stesso linguaggio ed, inoltre, per la portabilità del codice, in quanto l'applicazione client viene potenzialmente eseguita su macchine con sistemi operativi differenti.

I dettagli implementativi strettamente riguardanti il processo di decodifica e riproduzione del flusso audio esulano dall'obiettivo della trattazione e pertanto non saranno analizzati. Verranno invece presentati i dettagli più significativi della procedura adottata per effettuare la comunicazione con l'applicazione server.

Come accennato in precedenza, uno dei requisiti richiesti all'applicazione client è quello di rendere possibile l'invio al server delle informazioni di feedback circa il gradimento del brano ascoltato, oltre che ad altre informazioni di servizio. Il feedback è composto dalle

seguenti possibili azioni: *accept*, *reject* e *listen*. La scelta tra *accept* e *reject* è a totale discrezione dell'utente, come spiegato nella sezione precedente. L'azione *listen*, al contrario, viene generata in mono automatico dall'applicazione client non appena la riproduzione di un nuovo brano ha inizio. Queste tre azioni sono strettamente legate al processo di raccomandazione effettuato dal modulo SICS dell'applicazione server descritto in sezione 3.2.3. Le informazioni di servizio sono invece composte da nome utente e password per effettuare l'autenticazione, dalle informazioni contestuali dei brani musicali (titolo, artista, album e genere) e dalle azioni *connect* e *kill* utilizzate rispettivamente per inizializzare e terminare il processo che sulla macchina server rappresenta la radio personale.

```
2  private void sendObservation(String action){
3       this.observation = new ClientObservation(this.getUsername(), this.getPassword());
4       this.observation.setAction(action);
5       this.observation.setTitle(this.title_value.getText());
6       this.observation.setArtist(this.artist_value.getText());
7       this.observation.setAlbum(this.album_value.getText());
8       this.observation.setGenre(this.genre_value.getText());
9       ...
10      try{
11              this.socket = new Socket(this.serverAddress, this.serverPort);
12              //retrieve I/O streams
13              ObjectInputStream in = new ObjectInputStream(this.socket.getInputStream());
14              ObjectOutputStream out = new
   ObjectOutputStream(this.socket.getOutputStream());
15              String msg = (String)in.readObject();
16              System.out.println("PLAYER - Server says: \""+msg+"\"");
17              //send client's observation to the server
18              out.writeObject(this.observation);
19              out.flush();
20              ...
21              if(action.equalsIgnoreCase("connect")){
22                      String[] mountpoints = (String[])in.readObject();
23                      for(String mp : mountpoints){
24                              if(mp.endsWith(".ogg") && !playlist.contains(mp)){
25                                      playlist.add(mp);
26                              }
27                              ...
28                      }
29              }
30                      //close the connection
31              this.socket.close();
32      }catch(Exception e){
33              ...
34              e.printStackTrace();
35      }
36 }
```

Listato 3.15: Metodo sendObservation() della classe ThesisPlayer

Nel momento in cui è necessario effettuare la comunicazione con il server, il metodo *sendObservation()*, viene invocato. Il Listato 3.15 contiene i passaggi principali di tale metodo. Come si nota il metodo richiede come unico parametro l'azione da inviare al server. In primo luogo viene creato l'oggetto *observation* di tipo *ClientObservation* specificando nome utente e password come parametri. Vengono quindi aggiunte l'azione eseguita e le informazioni contestuali del brano ascoltato. A questo punto viene istanziata la socket tramite la quale l'oggeto *observation* sarà inviato al server. E' necessario

specificare l'indirizzo del server e il numero della porta sulla quale il server è in ascolto. I valori di tali parametri saranno quelli specificati dall'utente al momento dell'avvio dell'applicazione client. Una volta stabilita con successo la connessione, il server invia un messaggio di benvenuto dopodiché il client invia l'oggetto *observation* invocando il metodo *out.writeObject(this.observation)*. Infine la connessione con il server viene chiusa. Questo è il procedimento standard per l'invio di osservazioni nel caso in cui la radio personale sia già stata precedentemente inizializzata dall'applicazione server. Se però si tratta della prima connessione che il client effettua, l'azione da inviare al server sarà *connect*. In questo caso, come evidenziato nel Listato 3.15, dopo aver inviato l'oggetto *observation* il client attende una risposta dal server. La risposta conterrà l'indirizzo remoto tramite il quale il server trasmetterà i brani musicali raccomandati al client in forma di flusso audio. Tale indirizzo viene quindi reso visualizzabile nel menù a tendina dell'interfaccia grafica.

Si fa notare che la connessione con il server viene mantenuta soltanto per il tempo strettamente necessario ad inviare l'azione dopodiché essa viene chiusa. In questo modo si evita che l'applicazione server sprechi inutilmente risorse nel mantenere aperta la connessione quando essa, di fatto, non viene utilizzata.

Le azioni *start* e *stop* non vengono inviate al server, poiché esse sono utilizzate internamente all'applicazione client per avviare e terminare la riproduzione audio. L'azione *kill* invece viene generata automaticamente ed inviata al server nel momento in cui l'utente termina (chiude) l'applicazione client. Una volta ricevuta l'azione *kill* il server terminerà il processo associato alla radio personale dell'utente.

4 Test e valutazioni

In questo capitolo vengono presentati i test effettuati e i risultati ottenuti. Nella prima sezione verrà descritta la metodologia che è stata adottata per eseguire i test, mentre nella seconda sezione saranno illustrati i risultati ottenuti e verrà svolta l'analisi e la valutazione degli stessi.

4.1 Metodologia

Il sistema sviluppato è stato sottoposto a testing da parte di utenti reali secondo due differenti modalità, ed è poi stato messo a confronto con i risultati ottenuti da un terzo test, anch'esso svolto da utenti reali, del sistema commerciale di raccomandazione musicale Last.fm. I nove utenti sono stati suddivisi in tre gruppi (di tre componenti ciascuno) ognuno dei quali con il compito di effettuare uno dei tre suddetti test.

Ogni test prevede che ogni utente del gruppo ascolti 15 brani musicali e, per ciascuno, ne esprima un giudizio circa il gradimento su una scala di valori interi variabili da *0* a *5*. Non è prevista la possibilità di esprimere una valutazione con valori diversi o non interi. La scala di valutazione, di seguito riportata, risponde alla domanda:

"ti piace questo brano musicale?".

5 – molto
4 – abbastanza
3 – più si che no
2 – più non che si
1 – poco
0 – per nulla

Una volta collezionati i dati per ciascuno dei tre test previsti verrà analizzato l'andamento del voto assegnato dagli utenti nel corso dei 15 ascolti effettuati.

Modalità "Standard"

In questa modalità ciascun utente riceve le raccomandazioni elaborate dal sistema sia in base al feedback ricevuto sugli ascolti precedenti espresso mediante l'utilizzo dell'applicazione client, che in base alle azioni compiute dagli altri utenti. Poiché al momento dell'avvio il sistema presenta il problema del "cold-start" non possedendo alcuna informazione circa i gusti degli utenti, è stato deciso di inizializzare ciascuna stazione radio personale con un brano musicale per ognuno dei sette "macro-generi" (vedasi capitolo 3.2) in cui sono stati suddivisi. Questi brani iniziali vengono selezionati in modo casuale al momento della creazione delle singole playlist e, di conseguenza, sono diversi per ciascun utente. Al contrario i brani a partire dall'ottavo in poi, saranno quelli raccomandati dal sistema in base alle osservazioni ricevute.

Poiché le azioni che l'utente può compiere su ciascun brano ascoltato sono due (*accept* o *reject*), è stato scelto di associare ai brani rifiutati i voti della scala di valutazione da *0* a *2*, e a quelli accettati, invece, i voti da *3* a *5*.

Modalità "Random"

Come nella modalità standard, anche quella random prevede che gli utenti utilizzino l'applicazione client per l'ascolto dei brani musicali. Al contrario, però, i brani che saranno loro proposti non verranno elaborati dal modulo SICS in base alle azioni compiute sugli stessi, ma saranno scelti in maniera casuale dalla libreria musicale. Per ragioni di uniformità, comunque, è stato scelto di inizializzare le stazioni radio personali allo stesso modo della modalità standard, ovvero con un brano per ciascuno dei "macro-generi".

Anche in questo caso l'associazione tra i voti della scala di valutazione e le azioni *accept* e *reject* avviene come nella modalità standard, con la fondamentale differenza che tali azioni non influenzano in alcun modo le raccomandazione successive.

Last.fm

A differenza di entrambi i precedenti, quest'ultimo test prevede l'utilizzo da parte di ciascun utente, del servizio di raccomandazione musicale messo a disposizione da Last.fm attraverso la pagina web *http://www.last.fm/listen*. Agli utenti non è stato richiesto di possedere un account per tale servizio, bensì di utilizzare il player online come utente non registrato. In questa modalità non è possibile esprimere un giudizio esplicito sulle tracce proposte. Tale scelta è stata effettuata poiché nel caso in cui gli utenti fossero stati autenticati, i brani proposti loro dal sistema sarebbero stati basati sulle preferenze musicali da esso già precedentemente collezionate. Ciò avrebbe costituito una importante differenza rispetto a quanto avviene, al momento dell'avvio, nel il sistema realizzato il quale non possiede alcuna informazione a priori riguardo le preferenze degli utenti.

Il player online prevede che l'utente immetta il nome di un artista come punto di partenza per le raccomandazioni future. E' stata lasciata piena libertà agli utenti del gruppo di scegliere un artista di loro gradimento, in quanto il dato effettivamente importante al fine dell'analisi che verrà effettuata è il voto espresso sui brani consigliati ed il suo andamento nel corso dei 15 ascolti e non lo specifico brano musicale in sé. Inoltre, data la totale differenza tra la libreria musicale a disposizione di Last.fm e quella utilizzata nel sistema realizzato, sarebbe stato impossibile effettuare una significativa valutazione sugli specifici brani musicali raccomandati dai due sistemi.

4.2 Risultati e valutazione

Di seguito vengono riportati i risultati ottenuti per ciascuno dei tre test svolti. I dati raccolti sono stati elaborati in forma grafica in modo da renderli facilmente leggibili e confrontabili l'uno con l'altro.

Saranno proposti sia i grafici relativi al singolo utente, sia uno complessivo ricavato a partire dalla media dei giudizi assegnati da ciascun utente ai brani musicali. Infine viene presentato un confronto diretto tra i risultati ottenuti per ciascuno dei test svolti al fine di permetterne una immediata analisi comparativa.

Modalità "Standard"

Per una appropriata interpretazione dei risultati ottenuti in questo test è importante tener presente che i primi sette brani proposti sono stati selezionati dalla libreria musicale uno per ogni "macro-genere" in modalità casuale secondo il seguente ordine: *Pop*, *Rock*, *Classical-Soundtrack*, *Reggae-Ska*, *Jazz*, *Folk* e *HipHop-Electro*. Questa scelta, come già è stato detto, si è resa necessaria per affrontare il problema del "cold-start". Questo implica che le valutazioni circa la bontà del sistema nell'adattarsi al gusto del singolo utente dovranno essere fatte a partire dall'ottavo brano in poi.

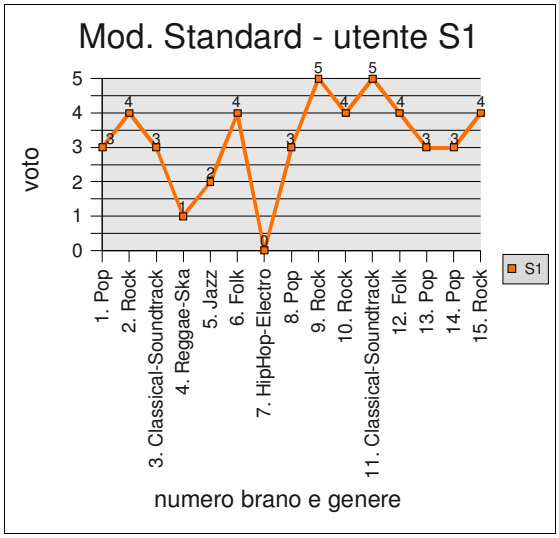

Fig. 4.1: Valutazione di gradimento dell'utente S1 in modalità Standard

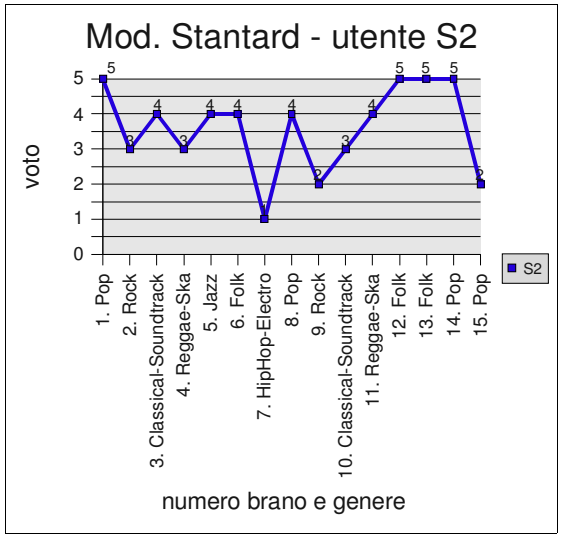

Fig. 4.2: Valutazione di gradimento dell'utente S2 in modalità Standard

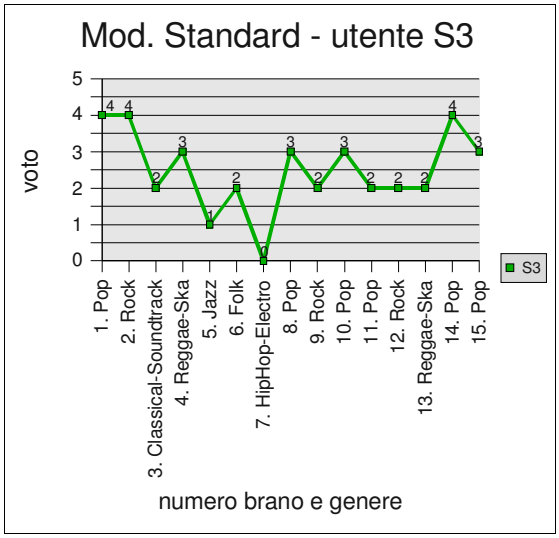

Fig. 4.3: Valutazione di gradimento dell'utente S3 in modalità Standard

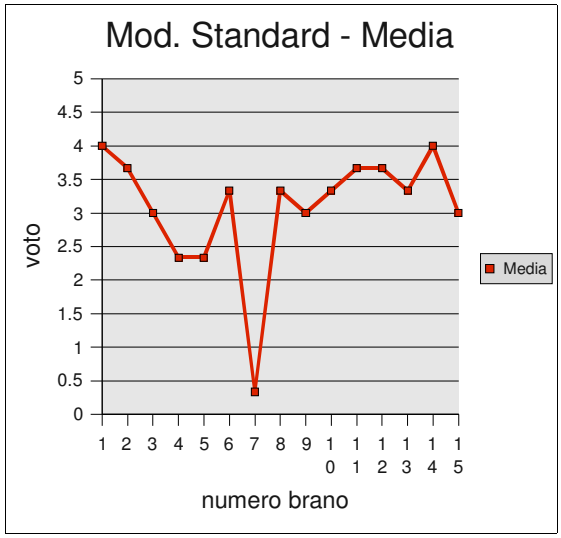

Fig. 4.4: Valutazione di gradimento media in modalità Standard

Analizzando i grafici, si nota che a partire dall'ottavo brano in poi, i valori assegnati alle tracce proposte, tendono, seppur con qualche eccezione (in particolare per l'utente S3), a stabilizzarsi su valori compresi tra *3* e *5* della scala di valutazione. Ciò significa che le tracce raccomandate sono state per la maggior parte accettate dagli utenti, in quanto rientranti nelle loro preferenze musicali. Il grafico della media (Fig. 4.4) conferma questa affermazione assumendo un valore medio globale pari a *3.09* sulla scala di valutazione.

Il fatto, poi, che tutti gli utenti abbiano assegnato una valutazione molto negativa al settimo brano proposto, è indicativo in quanto significa che essi, probabilmente, non hanno gradito il "macro-genere" musicale *HipHop-Electro*.

In base ai risultati ottenuti è possibile affermare che il sistema realizzato tende, effettivamente, ad adattarsi ai gusti musicali di ciascun utente. E' però fondamentale delineare i limiti che esso presenta.

I principali limiti del sistema sono dovuti a due fattori. Il primo fattore è la teoria utilizzata per esprimere il vincolo culturale (Relazione 3.1). In base alle tre regole definite, infatti, nel caso in cui un utente rifiuti per la prima volta un brano del genere X, tale utente non riceverà più raccomandazioni di brani musicali appartenenti al genere X. Questo fatto è riscontrabile, per esempio, nel grafico dell'utente S1 (Fig. 4.1). S1, infatti, rifiuta il terzo brano appartenente al genere *Reggae-Ska*, e, come si nota, nessuna raccomandazione successivamente ricevuta appartiene a tale genere. La stessa cosa accade, sempre per l'utente S1 con il genere *Jazz*.

Si noti, inoltre, che nel caso in cui un utente rifiutasse tutte le prime sette tracce musicali, egli non riceverebbe alcuna raccomandazione. Questo, comunque, è considerabile come un corretto comportamento del sistema.

La terza regola della teoria, contempla il fatto che se un utente accetta un brano del genere X e, in un secondo momento, rifiuta un altro brano dello stesso genere, allora egli potrà ancora ricevere raccomandazioni musicali appartenenti al genere X. E' stato scelto di adottare questa regola poiché l'utente potrebbe, ad esempio, gradire i brani del genere X di un determinato artista e non quelli dello stesso genere, ma di un altro artista.

Il secondo fattore che limita le capacità del sistema di adattarsi in modo ottimale alle preferenze musicali di un utente è dovuto alla categorizzazione nei sette "macro-generi", precedentemente presentati, dei brani musicali. Sebbene tale categorizzazione sia stata di fondamentale necessità per poter effettuare i confronti di similarità tra i brani ascoltati dagli utenti, essa ha comportato una elevata semplificazione della varietà dei generi musicali. Questo è significato, per esempio, associare al "macro-genere" *Rock* sia brani musicali di genere *Progressive Metal* che altri di *Rock melodico*. Di conseguenza, un utente a cui aggrada il primo e non il secondo, si trova ad accettare e rifiutare tracce appartenenti allo stesso "macro-genere". Ciò motiva, ad esempio, l'andamento del grafico illustrato in Fig. 4.3 relativo all'utente S3 il quale ha assegnato voti anche molto diversi tra loro a brani dello stesso genere musicale.

Entrambi i fattori problematici appena descritti sono legati alla parte di analisi del contenuto svolta al fine di valutare la similarità tra le tracce musicali. Essi conducono il

sistema a presentare, in parte, il tipico problema della “sovra-specializzazione” caratteristico dei sistemi di raccomandazione basati sul content-based filtering come, ad esempio, Pandora.

Sarebbe stato possibile categorizzare le tracce musicali in una quantità maggiore di generi musicali, ma questo avrebbe comportato la necessità di utilizzare un numero maggiore di brani per la fase iniziale nell'affrontare il problema del “cold-start”.

Poiché il sistema non prevede la creazione di profili persistenti associati agli utenti, l'applicazione del collaborative filtering è limitata alla valutazione di similarità tra le azioni compiute da ciascun utente su i brani ascoltati. Se, infatti, due utenti accettano entrambi i brani musicali del genere X, allora il sistema raccomanderà ad un utente il brano musicale accettato dall'altro utente purché esso non sia già presente nella sua playlist.

Modalità “Random”

Anche in questa modalità i primi sette brani musicali sono stati selezionati secondo il criterio utilizzato per la modalità “Standard”. A partire dunque, dall'ottavo brano in poi, le proposte ricevute da ciascun utente sono avvenute in modo del tutto casuale.

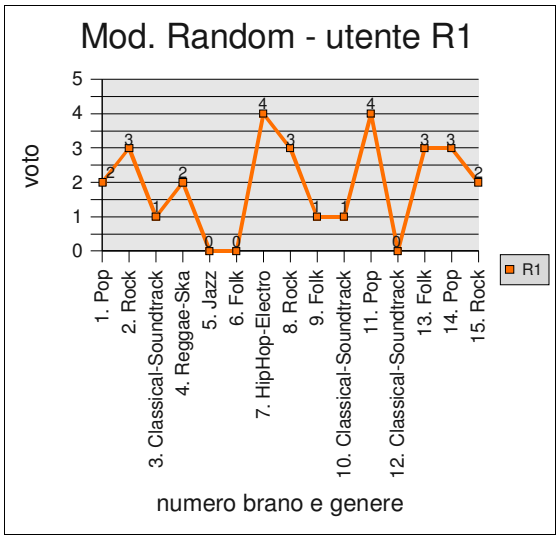

Fig. 4.5: Valutazione di gradimento dell'utente R1 in modalità Random

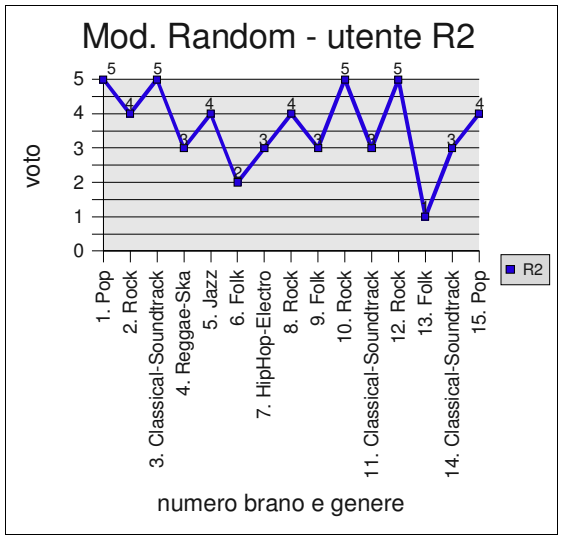

Fig. 4.6: Valutazione di gradimento dell'utente R2 in modalità Random

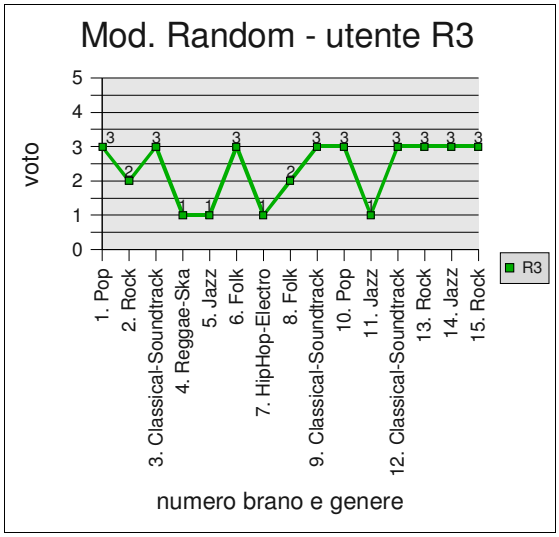

Fig. 4.7: Valutazione di gradimento dell'utente R3 in modalità Random

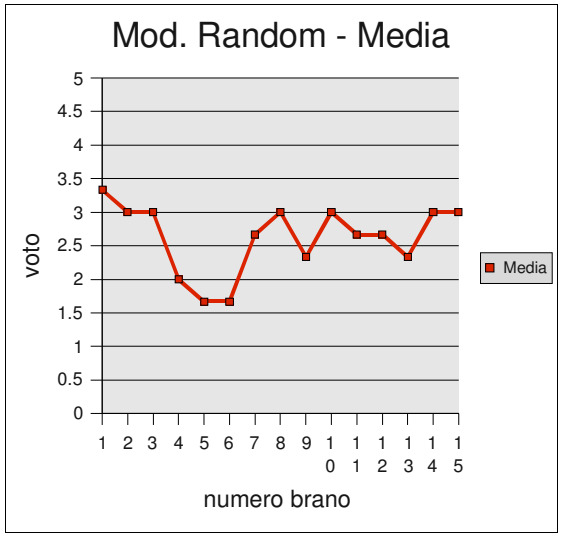

Fig. 4.8: Valutazione di gradimento media in modalità Random

Come era stato previsto, i risultati ottenuti in modalità Random sono, effettivamente, casuali. Il grafico relativo all'utente R1 (Fig. 4.5) presenta una grande variazione dei giudizi assegnati ai brani proposti anche se la maggior parte sono ti tipo negativo. Al contrario, invece, il grafico in Fig. 4.6 relativo all'utente R2 presenta per la maggior parte giudizi positivi, così come il grafico dell'utente R3, illustrato in Fig. 4.7, sebbene essi raggiungano come massimo il valore *3*.

Il grafico della media (Fig. 4.8) riassume i giudizi espressi dai tre utenti. Il giudizio medio globale assume un valore pari a *2.62* che si rivela essere di poco superiore alla sufficienza sulla scala di valutazione.

Last.fm

I grafici riportati in seguito illustrano i risultati ottenuti nel test del sistema commerciale di raccomandazione musicale Last.fm.

A differenza dei precedenti test, in questo caso la valutazione sulla bontà delle raccomandazioni ricevute potrà essere svolta a partire dal secondo brano proposto.

Poiché il test è stato svolto da utenti non autenticati, le raccomandazioni proposte da Last.fm si basano unicamente sulle correlazioni tra gli artisti computate a partire dai dati presenti nel database del sistema, senza tenere presente, quindi, eventuali precedenti preferenze degli utenti soggetto del test.

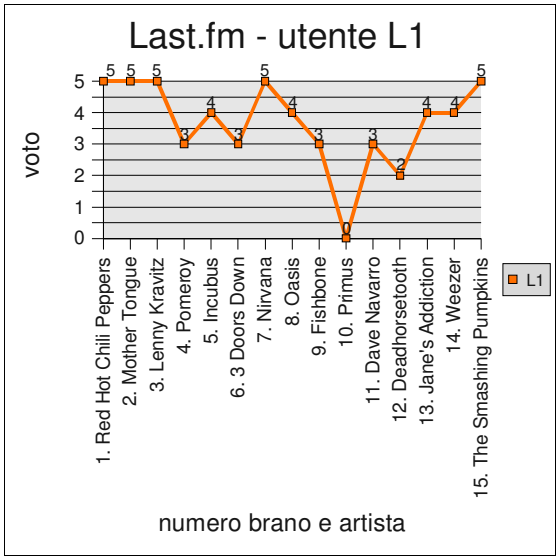

Fig. 4.9: Last.fm – valutazione di gradimento dell'utente L1

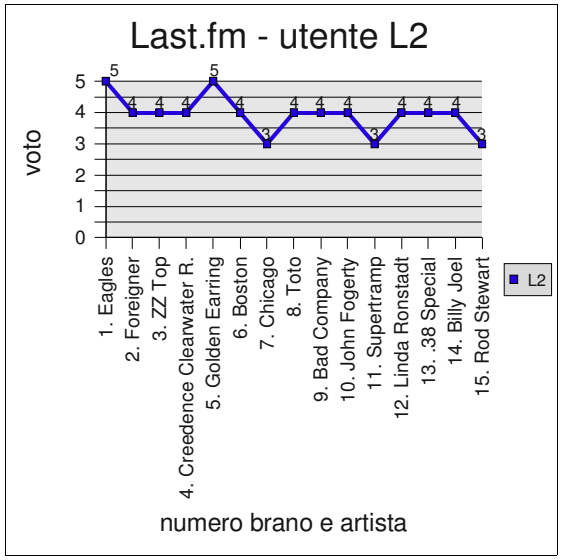

Fig. 4.10: Last.fm – valutazione di gradimento dell'utente L2

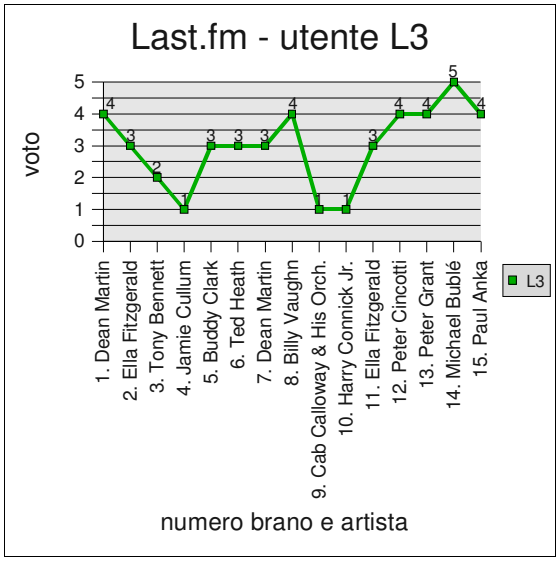

Fig. 4.11: Last.fm – valutazione di gradimento dell'utente L3

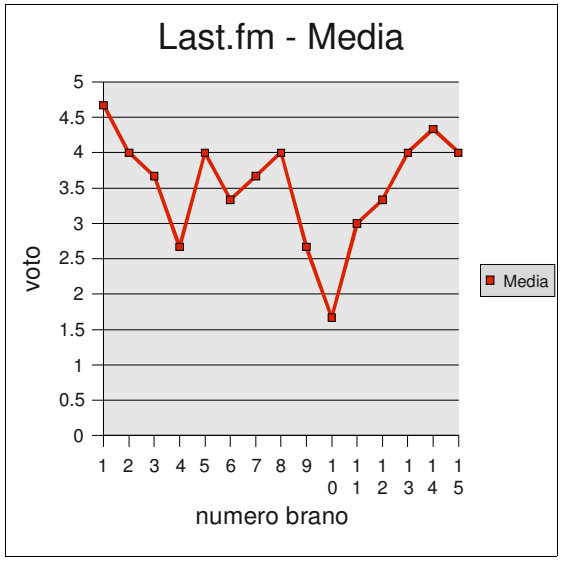

Fig. 4.12: Last.fm – valutazione di gradimento media

Al primo brano proposto, tutti gli utenti hanno espresso una valutazione molto positiva. Questo è dovuto al fatto che il sistema come prima raccomandazione suggerisce, nella maggior parte dei casi, un brano dell'artista stesso scelto dall'utente.

Inaspettatamente, dopo i primissimi brani proposti e dopo, circa, la metà degli stessi, due utenti su tre hanno espresso valutazioni molto negative. Nel complesso, comunque, i valori assegnati ai brani sono stati superiori alla sufficienza sulla scala di valutazione. Questo è evidenziato dal grafico in Fig. 4.12 il quale presenta un valore globale medio pari a *3.53*.

Confronto

A conclusione delle valutazioni sui test svolti, il grafico in Fig. 4.13 propone un confronto tra le medie dei risultati ottenuti.

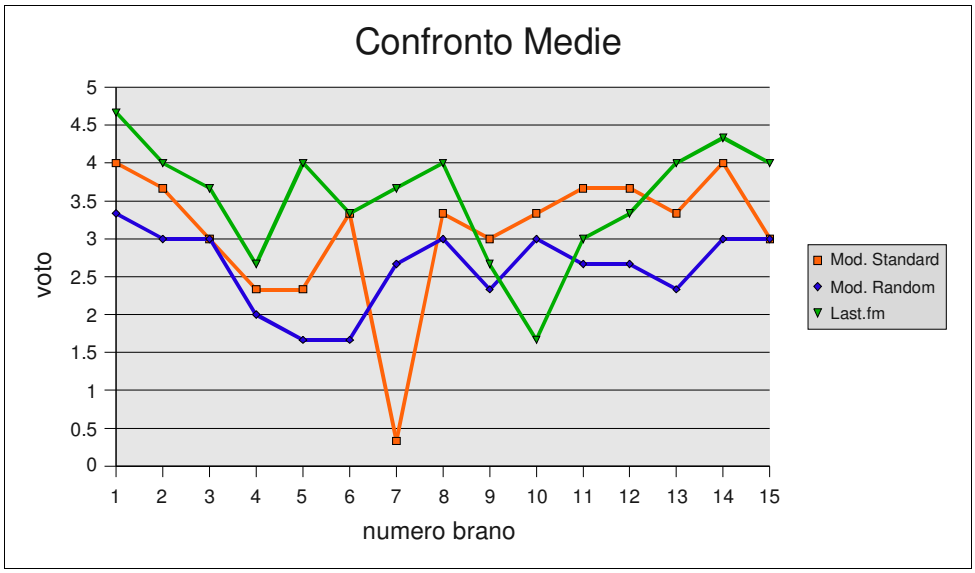

Fig. 4.13: Andamento delle valutazioni medie nei tre test svolti

Come era stato previsto, il sistema commerciale Last.fm dimostra un grado superiore di adattamento alle preferenze musicali degli utenti. Il valore medio sulla scala di valutazione ad esso associato è pari a *3.53*; decisamente al di sopra della soglia di sufficienza sulla medesima scala.

L'andamento delle valutazioni attribuite dagli utenti ai brani musicali proposti nella modalità Random è quello peggiore dei tre test effettuati, sebbene esso presenti un valore medio pari a *2.62*, che risulta essere di poco al di sopra della soglia di sufficienza.

Infine, il sistema realizzato, utilizzato in modalità Standard, presenta un buon andamento, soprattutto a partire dall'ottavo brano in poi, ovvero dal momento in cui gli utenti ricevono le effettive raccomandazioni musicali elaborate dal sistema. In questo caso il valore medio ottenuto, pari a *3.09* sulla scala di valutazione, si colloca a metà rispetto agli altri due test, con una differenza di *+0.47* punti sulla modalità Random e di *-0.42* sul risultato del test di Last.fm.

Si rivela, di particolare interesse analizzare l'andamento delle valutazioni ottenute nei tre test svolti a partire dall'ottavo brano, in quanto è da questo momento che i brani proposti agli utenti sono il risultato delle effettive elaborazioni compiute dal sistema.

In questo caso il valore della media delle valutazioni attribuite dagli utenti alle raccomandazioni ricevute è pari a *3.42* per il sistema proposto in modalità Standard, a *2.75* in modalità Random e a *3.38* per Last.fm. Questo dimostra che, nello specifico caso, il sistema realizzato risulta essere paragonabile a quello commerciale per quanto riguarda le valutazioni assegnate dagli utenti ai brani proposti.

In conclusione si vuole nuovamente ribadire che, per una corretta interpretazione dei risultati ottenuti, è necessario tenere in considerazione le importanti differenze sussistenti tra il sistema sperimentale realizzato e quello commerciale.

5 Conclusioni e sviluppi futuri

E' stato realizzato un sistema adattivo di raccomandazione musicale per internet radio personalizzate basato su Cultura Implicita.

Il sistema si è dimostrato essere di tipo ibrido in quanto utilizza entrambi gli approcci di content-based filtering e di collaborative filtering per generare suggerimenti musicali in base al feedback ricevuto dagli utenti sui brani ascoltati.

In base ai risultati ottenuti è possibile affermare che il sistema riesce in maniera discreta ad adattarsi alle preferenze musicali di ciascun utente, e quindi raggiunge lo scopo che era stato prefissato.

Cultura Implicita, si è rivelata essere una valida e funzionale tecnologia per la generazione di raccomandazioni. L'implementazione di SICS fornita da IC-Service, grazie alla sua generalità, è stata efficacemente adattata al caso specifico delle raccomandazioni musicali.

Poiché il lavoro svolto nella presente tesi è il primo tentativo di utilizzare Cultura Implicita in questo specifico dominio, la modellazione del problema è stata effettuata in modo semplice e in via sperimentale. Sviluppi futuri di questo sistema possono prevedere una modellazione più complessa utilizzando, oltre ad azioni, attori e attributi, anche elementi di tipo oggetto. Altre, e più complesse, teorie possono essere studiate, sulla base dei risultati ottenuti, per migliorare il comportamento generale del sistema. A questo proposito il modulo induttivo potrebbe essere impiegato per modificare dinamicamente le regole della teoria in modo da contemplare il fatto che, per esempio, un utente può non gradire, in linea generale, un determinato genere musicale, ma può, al contempo, gradire un artista specifico di tale genere.

Inoltre, la creazione e l'utilizzo di profili utente potrebbe aumentare il peso della componente di collaborative filtering utilizzato dal sistema e, allo stesso tempo, possono essere adottati approcci più complessi della semplice comparazione di attributi musicali di un brano (titolo, artista, album e genere) per l'analisi del contenuto e la valutazione di similarità.

L'affiancamento al concetto di radio personale di quello di radio globale fruita da una pluralità di utenti e generata a partire dalle singole preferenze musicali potrebbe rappresentare un'interessante estensione del sistema su cui concentrare gli sviluppi futuri.

6 Referenze

[1] Amazon.com: *http://www.amazon.com/*, 2008.

[2] Celma, O., Ramírez, M. and Herrera, P.: Getting music recommendations and filtering newsfeeds from FOAF descriptions. In *Proceedings of the 1st Workshop on Scripting for the Semantic Web, 2nd European Semantic Web Conference (ESWC2005)*, 2005.

[3] Fox, A.E.: Battle of the Music Recommender Systems: User-Centered Evaluation of Collaborative Filtering, Content-Based Analysis and Hybrid Systems. *Master's paper for the M.S. in I.S. degree, University of North Carolina*, 2007.

[4] Goldberg, D., Nichols, D., Oki, B.M. and Terry, D.: Using Collaborative Filtering to Weave an Information Tapestry. In *Communications of the ACM*, 61-70, 1992.

[5] Resnick P., Iacovou N., Suchak M., Bergstrom P. and Riedl J.: GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, 175-186, 1994.

[6] MovieLens: *http://movielens.org/*, 2008.

[7] Herlocker, J. L., Konstan, J. A., and Riedl, J.: Explaining Collaborative Filtering Recommendations. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW)*, 241-250, 2000.

[8] Last.fm: *http://www.last.fm/*, 2008.

[9] Pandora: *http://www.pandora.com/*, 2008.

[10] Jones, N. and Pu, P.: User Technology Adoption Issuesin Recommender Systems, 2007.

[11] Cano, P., Koppenberger, M. and Wack, N.: Content-based Music Audio Recommendation. In *Proceedings of the 13th annual ACM international conference on Multimedia*, 211-212, 2005.

[12] Mortensen, M.: Design and Evaluation of a Recommender System. *INF-3981 Master's Thesisin Computer Science Magnus Mortensen Faculty of Science Department of ComputerScience University of Tromsø*, 2007.

[13] Rashid, A. M., Albert, I., Cosley, D., Lam, S. K., Mcnee, S. M., Konstan, J. A. and Riedl, J.: Getting to Know You: Learning New User Preferences in Recommender Systems. In *Proc. of ACM IUI 2002, ACM Press*, 127-134, 2002.

[14] Burke, Robin: Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction* 12 no. 4 (November 1, 2002)

[15] Claypool, M, Gokhale, A., Miranda, T., Murnikov, P., Netes, D. and Sartin, M.: Combining content-based and collaborative filters in an online newspaper. In *Proceedings of ACM SIGIR Workshop on Recommender Systems*, 1999.

[16] Adomavicius, G. and Tuzhilin, A: Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* 17 no. 6 (, 2005)

[17] Basu, C., Hirsh, H., and Cohen, W.: Recommendation as classification: Using social and content-based information in recommendation. In *In Proceedings of the Fifteenth*

National Conference on Artificial Intelligence, 714-720, 1998.

[18] Burke, R.: Knowledge-based recommender systems, 2000.

[19] Park, Han-Saem, Ji-Oh Yoo, and Sung-Bae Cho: A Context-Aware Music Recommendation System Using Fuzzy Bayesian Networks with Utility Theory. In *Fuzzy Systems and Knowledge Discovery*, 970-979, 2006.

[20] Last.fm widgets boost user numbers: *http://www.guardian.co.uk/media/2008/feb/28/web20.digitalmedia*, 2008.

[21] Pandora: Music Genome Project: *http://www.pandora.com/mgp.shtml*, 2008.

[22] Blanzieri, E. and Giorgini, P.: From collaborative filtering to implicit culture: a general agent-based framework. In *In Proceedings of the Workshop on Agents and Recommender Systems*, 2000.

[23] Blanzieri, E., Giorgini, P., Massa, P. and Recla, S.: Implicit culture for multi-agent interaction support. In *CooplS '01: Proceedings of the 9th International Conference on Cooperative Information Systems*, 27-39, 2001.

[24] Birukou, A., Blanzieri, E., and Giorgini, P.: Implicit: An agent-based recommendation system for web search. In *Proceedings of the 4th International Conference on Autonomous Agents and Multi-Agent Systems*, 618-624, 2005.

[25] Birukou, A., Blanzieri, E., D'Andrea, V., Giorgini, P., Kokash, N. and Modena, A.: IC-Service: A service-oriented approach to the development of recommendation systems. In *Proceedings of ACM Symposium on Applied Computing. ACM Press*, 1683-1688, 2007.

[26] Birukou, A., Blanzieri, E. and Giorgini, P.: A multi-agent system that facilitates scientific publications search. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems. ACM*, 265-272, 2006.

[27] Kokash, N., Birukou, A. and D'Andrea, V.: Web Service Discovery Based on Past User Experience. In *Proc. Int'l Conf. Business Information Systems (BIS 07), LNCS 4439*, 95-107, 2007.

[28] Creative Commons: *http://www.creativecommons.org/*, 2008.

[29] Jamendo: *http://www.jamendo.com/*, 2008.

[30] Ogg Vorbis: *http://www.vorbis.com/*, 2008.

[31] Modena, A., Birukou, A. and Kokash, N.: Configuring SICS Core Step-By-Step. (version 1.5.1) *Internal document*, 2007.

[32] Modena, A., Birukou, A. and Kokash, N.: Configuring SICS COre Step-By-Step. (version 1.5.1) *Internal document*, 2007.

[33] Ices2 documentation: *http://www.icecast.org/docs/ices-2.0.0/*, 2008.

[34] Icecast2 documentation: *http://www.icecast.org/docs/icecast-2.3.1/*, 2008.

[35] JOrbis: *http://www.jcraft.com/jorbis*, 2008.