НОВ БЪЛГАРСКИ УНИВЕРСИТЕТ

МАГИСТЪРСКИ ФАКУЛТЕТ

ДЕПАРТАМЕНТ “ИНФОРМАТИКА”

ПРОГРАМА “СОФТУЕРНИ ТЕХНОЛОГИИ В ИНТЕРНЕТ”

МАГИСТРЪРСКА ТЕЗА

на тема

“Разпределена система за автоматизирано генериране на музикални мелодии, чрез взаимодействие между човек и компютър”

Разработил:

Тодор Димитров Балабанов

ФN№ F19952

Ръководител:

доц. д-р Симо Лазаров

София 2008

Благодарности

Авторът би искал да изкаже своите искрени благодарности към преподавателите в департамент “Информатика” на НБУ-София, за времето и вниманието, което отделиха в процеса на обучение и запознаването с всички технологии използвани в разработената магистърска теза.

Специални благодарности на доц. д-р Симо Лазаров, който като ръководител на разработката допринесе с изключителния си професионализъм, както и ценните идеи за начина по който работата да придобие значителна научна стойност.

Искрени благодарности на Стела Атанасова, по настояще докторант в НБУ-София, която изключително много помогна с музикалната част от работата. Без помощта на Стела настоящата магистърска теза би била суха програмистка теория и малко практически работен код.

Не на последно място, благодарности на всички близки и приятели, които по един или друг начин бяха част от моето професионално развитие и усъвършенстване като студент.

Тодор Балабанов

СЪДЪРЖАНИЕ

Благодарности....2
СЪДЪРЖАНИЕ....3
1 Въведение....7
2 Java Remote Method Invocation технология....9
2.1 Основни понятия....9
2.2 Технически подробности....9
2.3 Стъпки за създаването на RMI приложение....11
2.4 Java RMI за нуждите на разработката....11
3 Musical Instrument Digital Interface файлов стандарт 1.0....13
3.1 Парчета....14
3.2 Парчето MThd....15
3.3 Парчето MTrk....17
3.4 Количества с променлива дължина - време на събития....18
3.5 Събития....20
3.5.1 Sequence Number....23
3.5.2 Text....23
3.5.3 Copyright....24
3.5.4 Sequence/Track Name....24
3.5.5 Instrument....24
3.5.6 Lyric....25
3.5.7 Marker....25
3.5.8 Cue Point....26
3.5.9 MIDI Channel....26
3.5.10 MIDI Port....27
3.5.11 End of Track....28
3.5.12 Tempo....28
3.5.13 SMPTE Offset....33
3.5.14 Time Signature....34

3.5.15 Key Signature....34
3.6 Някои особености....35
4 Диференциална еволюция....36
4.1 Исторически бележки....36
4.2 Основна идея....37
4.3 Модификации за нуждите на разработката....38
5 Релационна система за управление на бази данни PostgreSQL....40
5.1 Общи понятия....40
5.2 Технически характеристики....40
5.3 Избор на СУБД....41
5.4 Съхранението на данни в приложението....42
6 Java аплети....44
6.1 Основни понятия....44
6.2 Концепцията на пясъчната кутия....44
6.3 Аплетът от страна на крайния потребител....45
7 Софтуер с отворен код....46
7.1 Основни понятия....46
7.2 Разлика между отворен код и комерсиален софтуер....46
7.3 Разликата между програмен код и компилиран код....47
7.4 Как действа отворения код....47
7.5 Разликата между свободен софтуер и отворен код....48
7.6 Лицензи за отворен код....48
7.7 Определяне на продуктите с отворен код....49
7.8 Разликата в различните лицензи....49
7.9 Предимства на софтуера с отворен код....50
7.10 Недостатъци на софтуера с отворен код....51
7.11 Настоящата разработка....52
8 Същност на разработеното приложение....53
8.1 Дефиниция на проблема....53
8.2 Мотивация за избраното решение на поставения проблем....53
8.3 Концептуално решение на проблема....55

8.4 Архитектура на системата....57
8.5 Комуникация в системата....62
8.6 Изграждане на приложението от програмния код....62
8.7 Поддръжка и отстраняване на проблеми....63
8.8 Инструкции за инсталация....64
8.9 Инструкции за употреба....64
8.9.1 Стартиране на сървъра....65
8.9.1 Стартиране на клиента....65
8.10 Параметри за стартиране на сървъра от командния ред....65
8.11 Управление на проекта....67
9 Експериментални данни и резултати....69
10 Изводи, заключения и препоръки за бъдеща разработка....70
11 Информационни източници....72
Приложение А - Компакт диск....73
Приложение Б - Структура на базата данни....73
Приложение В - Програмен код....74
build.bat....74
wideopen.policy....76
client.html....76
database.properties....76
Melody.java....76
Note.java....88
Population.java....93
MIDIDERMIClient.java....100
MIDIDERMIInterface.java....104
MIDIDERMITask.java....105
DatabaseMediator.java....108
DatabaseSetMelodiesProvider.java....111
FileMelodyProvider.java....112
FileSetMelodiesProvider.java....115
FractalMelodyProvider.java....116

FractalSetMelodiesProvider.java....118
NotValidDescriptorFileException.java....119
RandomMelodyProvider.java....121
RandomSetMelodiesProvider.java....122
MelodyPool.java....123
MIDIDERMIImplement.java....131
MIDIDERMIServer.java....134
MidiFileProducer.java....136

1 Въведение

Настоящата магистърска теза изцяло е насочена в областта за автоматизирано композиране на музикални мелодии с помощта на компютър и взаимодействието на човека с машината. За целите на разработката е създадена софтуерна система, работеща в разпределена среда на принципа “клиент-сървър”. Проблемът, който се разрешава е разработването на системата, а не създаването на набор от мелодии с помощта на системата, въпреки че това е вторичен продукт от работата. Постигането на резултати, чрез използването на системата излиза извън проблемите, които са разрешени в настоящата разработка. Създаването на естетически издържани музикални мелодии е обект на правилно заложени фрактални формули, начален набор от мелодии композирани от човек, и многократно експериментиране с голям брой крайни потребители (слушатели/оценители). За създаването на подобни системи се работи усилено през последното десетилетие. [1]

По своята същност системата представлява RMI базиран сървър който разпраща на RMI базирани клиенти набор от музикални мелодии. От страната на клиента мелодиите се рекомбинират с помощта на оптимизационния алгоритъм за диференциална еволюция. Всички мелодии подлежат на прослушване от крайния потребител и на база оценките му се управлява метода на еволюция. Оценените мелодии се връщат на сървъра, където се преразпределят за прослушване от други потребители. При клиента съществува функционалност за визуално представяне на музикалната информация и механизъм за оценка на прослушаното. От страна на сървъра е добавена възможност за съхраняване на информацията в база данни, както и в бинарни MIDI файлове.

В предстоящото изложение са разгледани всички аспекти, оказали влияние в процеса на разработка. След съдържанието и въвеждащата секция следват следните секции: Представяне на Java RMI технологията; Представяне на стандарта

MIDI файлов формат 1.0; Представяне на евристичния оптимизационен алгоритъм Диференциална еволюция; Представяне на релационната система за управление на бази от данни PostgreSQL; Представяне на Java аплет технологията; Дискусия за предимствата и недостатъците на приложенията с отворен код; Изложение на разработката и подробно описание на разработеното приложение; Кратка експериментална част с обобщение на получените резултати; Изводи, заключения и препоръки; Използвани информационни източници; Приложение А - посветено на компакт диска, към разработката; Приложение Б - структура на базата данни; Приложение В - пълен програмен текст;

В по-голяма част настоящото изложение се представя общоизвестни технологии и алгоритми. Всеки, който се чувства запознат с общоизвестните неща спокойно може да се насочи към последните няколко секции от този документ, който представят същността на разработката.

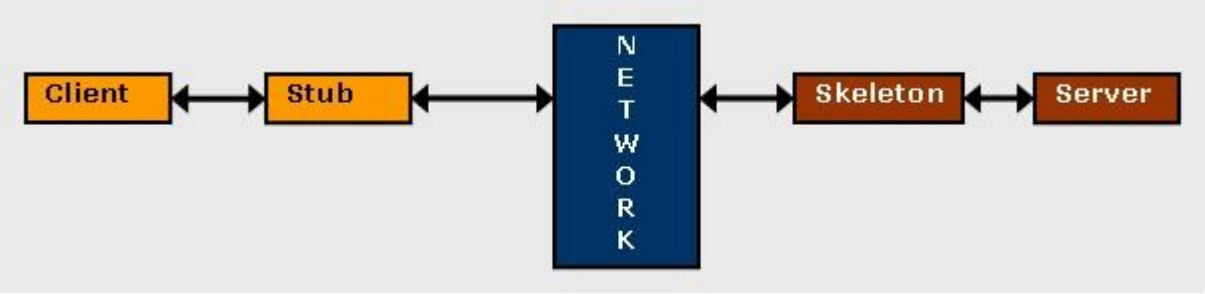
2 Java Remote Method Invocation технология

В настоящата секция ще бъде представена технологията за отдалечено изпълнение на код, която стандартно се поддържа от езика Java на компанията Sun Microsystems, Inc.

2.1 Основни понятия

Java RMI API е приложен програмен интерфейс, обектен вариант за извършването на отдалечено извикване на процедури.

Има две основни реализации на това API. Оригиналната реализация зависи от представянето на класовете в Java виртуалната машина и поради тази причина единствено поддържа обръщения от една виртуална машина към друга. Протоколът стоящ под този механизъм е известен под названието JRMP (Java Remote Method Protocol). За да се поддържа код изпълняван в среда без Java виртуална машина в последствие е добавена и втората реализация на основата на CORBA. [2]



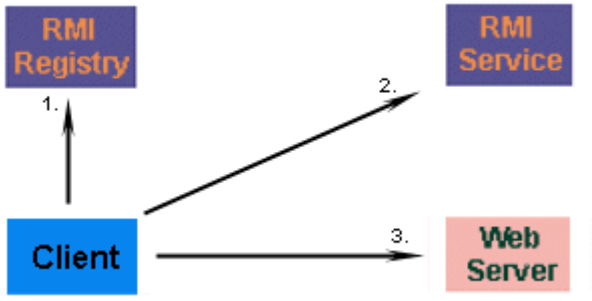
Фиг. 2.1 Модел за осъществяването на Java-RMI връзка с използването на скелети и гнезда.

2.2 Технически подробности

RMI позволява извикването на методи на обекти между различни Java виртуални машини. Java виртуалните машини могат да се изпълняват на различни компютри и въпреки това е възможно извикването на отдалечени методи, които принадлежат на отдалечената Java виртуална машина. Методите дори могат да си предават обекти, които отдалечената виртуална машина никога не е срещала преди това, позволявайки динамичното зареждане на нови класове при необходимост. Това е една от най-мощните възможности на Java RMI технологията.

Ако се вземе като пример следния сценарии: Програмист А написва услуга, която предлага някаква полезна функционалност. Този програмист редовно обновява тази услуга, добавяйки нова функционалност и подобрявайки вече съществуващата функционалност. Програмист В иска да използва услугата предоставена от програмист А. При тази схема е неудобно А винаги да обновява кода който В използва.

Java RMI предлага много лесно решение на този проблем. Тъй като RMI може динамично да зарежда нови класове, то програмист В може да остави RMI механизъма автоматично да обновява класовете които програмист А публикува в определена уеб директория, от където RMI може да изтегли обновените класове при необходимост.



Фиг. 2.2 Ред на връзките при използване на RMI от клиента.

Връзките, които се правят когато клиентът използва RMI са както следва: Първо, клиентът трябва да се свърже с RMI регистъра и трябва да поиска услугата

по име. Клиент приложение няма как да знае точното местоположение на услугата, но знае достатъчно за да се свърже с RMI регистъра на сървъра. Тази връзка ще го насочи към услугата, която клиента се опитва да извика.

От страна на сървъра кодът на услугата се променя често, така че клиентът не винаги разполага с актуално копие на класовете, променени от страна на сървъра. Този проблем се разрешава като клиента изтегля новите класове от уеб сървър, където двете приложения си споделят класове. Новите класове се зареждат в паметта на клиента и са готови за използване. Всичко това се случва напълно прозрачно за програмистите от страна на сървъра и от страна на клиента, не е нужно да се пише допълнителен код за извличане на класовете. [3]

2.3 Стъпки за създаването на RMI приложение

Първоначално се създава интерфейсът с методи, които сървърът ще предлага в публичното пространство. След това се имплементира отдалеченият интерфейс и се създава приложение, което да зарежда и регистрира съответната RMI услуга в локалния RMI регистър. Последно се създава клентското приложение. Стартирането на системата протича през следните стъпки: 1. Стартиране на RMI регистъра; 2. Компилиране и стартиране на сървъра; 3. Компилиране и стартиране на клиента.

2.4 Java RMI за нуждите на разработката

Езикът Java бе избран за нуждите на разработката, тъй като това е един от най-съвременните конвенционални програмни езици. Със своята платформена независимост той гарантира възможност за достигане на крайния продукт до максимален брой крайни потребители. В същото време езикът и платформата за разработка са налични под формата на отворен код, което е изключително важно за оцеляването на един софтуерен продукт в рамките на години експлоатация.

Обмяната на данни чрез Java RMI изключително улеснява проектирането и реализацията на системата от гледна точка на канала за комуникация, между сървъра и множеството клиенти, които се прикачат към него. С помощта на Java RMI мрежовата комуникация се свежда до класическо извикване на методи от обекти, които почти не се различават от обектите налични в паметта на локалната Java виртуална машина.

3 Musical Instrument Digital Interface файлов стандарт 1.0

В настоящата секция на кратко ще бъде представен стандартът за MIDI файловия формат. Без да претендира за изчерпателност тази секция ще акцентира върху основните фрагменти от стандарта използвани в разработката. Секцията представя буквален превод на спецификацията за стандарта. [4]

Стандартен MIDI файл (SMF) е файлов формат специално проектиран за съхранение и просвирване на данни използвани от музикалните секуенсъри (секуенсърите могат да са софтуерни или хардуерни).

Файловият формат съхранява MIDI съобщения (статус байтове с подходящи байтове за данни) плюс времеви маркер за всяко съобщение (последователност от байтове които показват колко времеви интервала трябва да изтекат преди определено събитие да бъде изпълнено). Форматът позволява да се съхрани информация за темпото, интервал за четвърт нота (или разделителна способност в брой тактове за секунда), времеви и ключови маркери, имена на пътеки и шаблони. Може да се съхраняват множество шаблони и пътеки, така че всяко приложение да може да възпроизвежда тези структури при зареждането на файла.

Забележка: Пътека обикновено съдържа информацията за един музикален инструмент, примерно партитурата на тромпета. Шаблон представлява аналог на множество музикални партитури (примерно тромпет, барабани, пиано и т.н.) за една песен или фрагмент от песен.

Файловият формат е така проектиран, че да бъде базов, така че всеки секуенсър да може да чете и записва такъв файл без загуба на най-важните данни и да бъде достатъчно гъвкав, така че конкретно приложение да съхранява свои допълнителни данни по такъв начин, че друго приложение да не бъде объркано при

зареждането на файла и безопасно да игнорира допълнителната информация, която така или иначе му е непотребна. За MIDI файловият формат може да се мисли като за музикална версия на ASCII текстов файл (с изключение на, това че MIDI съдържа също и двоични данни) и различни секуенсър програми като текстови редактори са способни да разчитат този файл. За разлика от ASCII, MIDI файловият формат запазва данните под формата на парчета (група от байтове предхождани от идентификатор и размер на парчето), които могат да бъда парсвани, зареждани или пропускани. Поради тази причина файловият формат може лесно да бъде разширяван, така че да включва специфична за производителя информация. Примерно програма може да иска да запазва байт с флагове за това дали потребителят е включил чуваеми цъкания за метронома. Програмата може да сложи този флагов байт в MIDI файла по такъв начин, че другите приложения да пропускат този байт без да имат нужда да разбират за какво точно служи. В бъдеще MIDI файловият формат може да бъде разширен и да включва нови официални парчета, които всички програми ще могат да изберат и да заредят. Всичко това може да бъде направено без да се обявяват старите файлове за несъвместими (форматът е така проектиран, че да бъде разширяем чрез запазване на обратната съвместимост).

3.1 Парчета

Данните винаги се записват като част от парче. Може да има много парчета в един единствен MIDI файл. Всяко парче може да бъде с различен размер (размерът показва колко байта се съдържат в съответното парче). Байтовете с данни са свързани по някакъв начин в самото парче. Парче е просто група от свързани по някакъв начин байтове.

Всяко парче започва с 4 символа (4 ASCII байта) идентификатор, който показва какъв тип е парчето. Следващите 4 байта (всеки байт се състои от 8 бита) формират 32 бита с информация за дължината на парчето (размер). Всички парчета

трябва да започват с тези две полета (8 байта), които се определят като заглавна част на парчето.

Забележка: Полето за дължина на парчето не включва 8 байта за заглавната част. Това поле просто казва колко байта следват след заглавната част.

Пример за заглавна част на парче (байтовете са представени в шестнадесетичен формат):

4D 54 68 64 00 00 00 06

Забелязва се, че първите 4 байта съставят ASCII идентификатора MThd (първите 4 байта са ASCII стойностите за буквите 'M', 'T', 'h' и 'd'). Следващите 4 байта показват, че трябва да последват още 6 байта с данни за това парче (след което трябва да последва заглавната част на друго парче или краят на файла).

Всъщност, всички MIDI файлове започват с тази MThd заглавна част (по този начин се разбира, че това е точно MIDI файл).

Забележка: Четирите байта за размера на парчето се съхраняват в последователността използвана в Motorola 68000, а не в Intel обратна последователност (06 е четвъртият байт, а не първият от четирите). Всички полета съхранени в повече от един байт в MIDI файла следват този стандарт, познат най-вече като "Big Endian" формат.

3.2 Парчето MThd

Заглавната част на MThd парчето има идентификатор MThd и дължина 6 байта.

По-подробно изследване на шестте байта, следващи заглавната част показва следното: Първите два байта с данни показват формата (или типа на файла). Съществуват три различни типа (формата) MIDI файлове. Тип 0 означава, че файлът се състои от една пътека, съдържаща MIDI данни за всичките 16 възможни канала. Ако секуенсърът сортира/записва всичките MIDI данни в един единствен блок памет с последователност в която трябва да бъдат просвирени то този формат е най-подходящия. Тип 1 означава, че файлът се състои от една или повече едновременни пътеки (всичките започващи от приет нулев момент във времето), най-често всяка пътека е асоциирана с отделен канал. Заедно всичките пътеки се смятат за една последователност или шаблон. Ако секуенсърът разделя MIDI данните (пътека) в различни блокове памет, но ги просвирва едновременно (като едно произведение), то този тип файл е най-подходящ. Тип 2 означава, че файлът се състои от един или повече последователни и независими шаблона, изградени от една пътека. Ако секуенсърът разделя MIDI данните в различни блокове памет, но изпълнява само един блок в определен момент от времето (всеки блок се смята за различно произведение или песен), то този тип файл е най-подходящ.

Следват два байта, които показват колко пътечки са записани във файла. Разбира се за файловете от тип 0 винаги е записана стойност 1. За типове 1 и 2 може да има различен брой пътечки.

Последните два байта показват колко такта са за четвъртина нота (PPQN Pulses Per Quarter Note), като резолюция на която са базирани времевите маркери (деление). Примерно, ако секуенсърът има 96 PPQN, то това поле ще съдържа (в шестнадесетичен формат):

00 06

Алтернативно, ако първият байт за деление е отрицателен, то тогава делението е части от секундата на които са базирани времевите маркери. Първият байт ще бъде -24, -25, -29 или - 30, отговаряйки на 4 SMPTE стандарта за

представяне на кадри в секунда. Вторият байт (положително число) е резолюцията в самия кадър (или подкадър). Обичайни стойности са 4 (MIDI времеви код), 8, 10, 80 (SMPTE резолюция на бит) или 100.

За да се зададат времеви маркери в милисекунди то се използват стойностите -25 и 40 подкадъра.

Примерно MThd парче (със заглавна част):

4D 54 68 64	MThd идентификатор.
00 00 00 06	Дължината на парчето MThd е винаги 6 байта.
00 01	Файлов формат от тип 1.
00 02	Файлът съдържа 2 музикални пътеки.
E7 28	Всяко увеличаване на делта времето е с размер в милисекунди.

3.3 Парчето MTrk

След парчето MThd трябва да следва MTrk парче, което е единственото дефинирано до момента парче (ако бъде открито парче с друг идентификатор то просто трябва да бъде пропуснато, като се игнорират такъв брой байтове какъвто е указан в полето за размер).

MTrk парчето съдържа всички MIDI данни (с времеви маркери) плюс не-MIDI данни в една от пътечките. Очевидно трябва да бъдат налични толкова пътечки, колкото са оказани в заглавната част на MThd.

Заглавната част на MTrk започва с идентификатора MTrk, непосредствено последван от стойност за размера (брой байтове, които трябва да бъдат прочетени за тази пътечка). Размерът на всяка пътечка се дефинира по отделно за всяка

пътечка (Все пак, пътека съдържаща партитурата за цигулка в концерт от Бах най-вероятно съдържа повече данни, отколкото ритъм на барабан от два удара).

3.4 Количества с променлива дължина - време на събития

Пътечката в MIDI файла се представя по начин аналогичен на представянето им в секуенсър. Пътеката в секуенсъра представлява последователност от събития. Примерно, първото събитие в пътеката може да е засвирване на средна нота C. Второто събитие може да е засвирване на нота E над средната нота C. Тези две събития може да се случат в един и същи момент от времето. Третото събитие може да бъде спиране на просвирването на средната нота C. Това събитие може да се случи няколко музикални такта след първите две събития (средна нота C се просвирва за няколко музикални такта). Всяко събитие има време в което трябва да се появи, като събитията са подредени в парчето памет по хронологичен ред за тяхната поява.

В MIDI файла времето на събитието предшества байтовете с данни, които представляват самото събитие. С други думи, байтовете които съставляват времевия маркер предхождат останалите. За някои събития времевия маркер е относителен спрямо предходното събитие. Примерно, ако първото събитие се появи 4 такта след началото на просвирването тогава делта времето е 04. Ако следващото събитие се появява едновременно с първото, то делта времето е 00. Тоест делта времето е продължителността (в брой тактове) между събитието и предходното събитие.

Забележка: Тъй като всички пътеки започват с време 0 по подразбиране, то първото относително време винаги се смята спрямо нулата.

Делта времената се записват в серия от байтове, които се наричат количества с променлива дължина. Значими са само първите 7 бита от всеки байт (дясно

подравняване, подобно на ASCII байт). Тоест, ако има 32 битово делта време трябва да се пакетира в 7 битови серии (примерно ако трябва да се предава като SYSEX съобщение). Разбира се, ще има променлив брой байтове в зависимост от стойностите за делта време. За да се определи кой е последният байт в серията, то този байт има 0 в бит номер 7. Всички предхождащи байтове имат 1 в бит номер 7. Тоест, ако делта времето е между 0 и 127 то може да се представи в един байт. Най-голямото позволено делта време е 0FFFFFFF, което се трансформира в 4 байта с променлива дължина. Следва пример за делта времена в 32 бита и техния еквивалент в количества с променлива дължина:

Число	Променливо количество
00000000	00
00000040	40
0000007F	7F
00000080	81 00
00002000	C0 00
00003FFF	FF 7F
00004000	81 80 00
00100000	C0 80 00
001FFFFF	FF FF 7F
00200000	81 80 80 00
08000000	C0 80 80 00
0FFFFFFF	FF FF FF 7F

Представени са функции, написани на C, за четене и запис на количества с променлива дължина, каквито са делта времената. На WriteVarLen() се подава 32 битова стойност (unsigned long), която се разделя на съответстващата серия от байтове, записани във файл. ReadVarLen() чете серия байтове от файл докато достигне последния байт от количеството с променлива дължина и връща 32 битова стойност.

```
void WriteVarLen(register unsigned long value)
{
   register unsigned long buffer;
   buffer = value & 0x7F;

   while ( (value >>= 7) )
   {
     buffer <<= 8;
     buffer |= ((value & 0x7F) | 0x80);
   }

   while (TRUE)
   {
      putc(buffer,outfile);
      if (buffer & 0x80)
          buffer >>= 8;
      else
          break;
   }
}


unsigned long ReadVarLen()
{
    register unsigned long value;
    register unsigned char c;

    if ( (value = getc(infile)) & 0x80 )
    {
       value &= 0x7F;
       do
       {
         value = (value << 7) + ((c = getc(infile)) & 0x7F);
       } while (c & 0x80);
    }

    return(value);
}
```

Забележка: Концепцията за количества с променлива дължина (разделяне на големи стойности в серия от байтове) се използва и в други полета на MIDI файла, а не само за делта времената както ще стане ясно в последствие.

3.5 Събития

Първите байтове (от 1 до 4) в MTrk са делта времето на първото събитие, представени като количество с променлива дължина. След това следва първият байт на съответното събитие. Тези данни могат да се смятат като статус на събитието. За MIDI събитията това е актуалният статус байт на събитието (първият статус байт е за това дали събитието се изпълнява). Примерно, ако байтът е шестнадесетично 90 то това е събитието просвирване на нота на MIDI канал 0. Ако

байтът е шестнадесетично 23 трябва да се използва статусът на предишното събитие (верижен MIDI статус на изпълнение). Очевидно първото събитие в парчето MTrk трябва да има статус байт. След MIDI статус байта следват 1 или 2 байта данни (в зависимост от статуса - някои MIDI съобщения имат 1 последователен байт данни). След това следва следващото делта време за събитие (като променливо количество) с което започва и процеса за прочитане на следващото събитие.

SYSEX (специални служебни) съобщения (статус = F0) са особен случай, тъй като могат да имат всякаква дължина. След статуса F0 (който винаги се записва - няма статус при изпълнение) следва серия от байтове с променлива дължина. С помощта на ReadVarLen() се получава 32 битова стойност, която показва колко байта още следват и са част от това SYSEX събитие. Дължината не включва статуса F0.

Примерно следното SYSEX MIDI съобщение:

F0 7F 7F 04 01 7F 7F F7

То ще бъде съхранено в MIDI файла като следната последователност от байтове (не се показват делта време байтовете, които го предхождат):

F0 07 7F 7F 04 01 7F 7F F7

Някои MIDI модули изпращат изключителни системни съобщения като серия от малки пакети (със закъснение по време между предаването на всеки пакет). Първият пакет започва с F0, но не завършва с F7. Последващите пакети не започват с F0 нито завършват с F7. Последният пакет не започва с F0, но завършва с F7. По този начин между отварянето в първия пакет с F0 и затварянето в последния пакет с F7 има само едно SYSEX съобщение (Забележка: Използването на тази възможност за предаване на пакети е признак за много лошо проектиране на системата). Разбира се, тъй като е нужно закъснение при изпращането на всеки пакет, то всеки пакет трябва да бъде съхранен като отделно събитие със свое специфично време в MTrk.

Освен това е нужно да се знае по някакъв начин кои събития не бива да започват с F0 (всички, осен първия пакет). За тази цел спецификацията на MIDI файловете предефинира статуса F7 (обичайно използван като краен маркер за SYSEX пакети) по такъв начин, че да обозначава пакети, които не започват с F0. Ако такова събитие последва F0 събитие, то се приема, че F7 събитието е втори пакет в последователност от пакети. В този контекст събитието се отнася като SYSEX CONTINUATION събитие. Точно както F0 събитие то има променлива стойност за обозначаване на дължината и данни, които следват след това. От друга стана F7 събитието може да се ползва за съхраняване на MIDI REALTIME или MIDI COMMON съобщения. В този случай след количеството с променлива дължина, обозначаващо размера, трябва да се очакват следните статус байтове: F1, F2, F3, F6, F8, FA, FB, FC или FE (трябва да се отбележи, че такива статус байтове не може да се открият в SYSEX CONTINUATION събитието). При това използване F7 се отнася като ESCAPE събитие.

Статусът FF е резервиран да обозначава специални не-MIDI събития (трябва да се отбележи, че FF се използва в MIDI да означи ресет, така че този статус няма да бъде съхраняван във файл, за това MIDI спецификацията за файлове предефинира този статус). След статуса FF следва друг байт, който показва какъв тип не-MIDI събитие е това. Може да се смята за вид втори статус байт. След което следват байтове (или само един) във формата за променливи количества, които показват колко още байта са част от това събитие. Тази дължина не включва FF, байта за тип, нито байтовете за самата дължина. Тези специални не-MIDI събития се наричат мета-събития и повечето са незадължителни, освен ако не е оказано противното. По-надолу следва дефиницията на някои мета събития (включително FF статуса и размера). Трябва да се отбележи, че ако изрично не е споменато то тези събития може да се съдържат повече от един път в MTrk (дори същото мета-съобщение) на различни делта времена. Точно както MIDI събитията, така и мета-събитията имат делта време, спрямо предходното събитие, независимо какво точно е предходното събитие, така че свободно може да се смесват MIDI събития и мета-събития.

3.5.1 Sequence Number

FF 00 02 ss ss

Това е незадължително събитие, което трябва да се появи в началото на MTrk (преди всички събития с нулево делта време и преди всички MIDI събития) и определя броя последователности. Двата байта за данни ss ss са числото което отговаря на MIDI Cue съобщението. Във файл от тип 2 това число определя всеки шаблон (тоест всяко MTrk парче), така че последователността на мелодията може да използва съобщението MIDI Cue за да се обръща към шаблоните. Ако числата ss ss са пропуснати (примерно размер 0 вместо 2 за събитието), то тогава се използва местоположението на MTrk парчето във файла (примерно първото MTrk парче е и първия шаблон). При файлове от тип 0 или 1 има само един шаблон (въпреки че при тип 1 има няколко MTrk парчета) това събитие се поставя само в първото MTrk парче. Така група от файлове тип 1 с различни номера на последователности могат да се сравнят с колекция от песни.

Може да има само по едно такова събитие във всяко MTrk парче при файлове от тип 2. Може да има само едно такова събитие при файлове от тип 0 и 1 и то трябва да е в първото MTrk парче.

3.5.2 Text

FF 01 len text

Всякакво количество текст (брой байтове = len) за каквито и да е цели. Най-добре е това събитие да се намира в началото на MTrk. Въпреки че този текст може да се използва за всякакви цели има и други текстово базирани събития като: оркестрация, лирика, име на пътека и т.н. Това събитие основно се използва за

добавяне на коментари към MIDI файла, като се очаква програмите, които зареждат файла да го игнорират.

Трябва да се отбележи, че len може да бъде серия от байтове, тъй като се изразява под формата на количество с променлива дължина.

3.5.3 Copyright

FF 02 len text

Съобщение за авторски права (текст). Най-добре е това събитие да се намира в началото на MTrk.

Трябва да се отбележи, че len може да бъде серия от байтове, тъй като се изразява под формата на количество с променлива дължина.

3.5.4 Sequence/Track Name

FF 03 len text

Име на последователност или пътека (текст). Най-добре е това събитие да се намира в началото на MTrk.

Трябва да се отбележи, че len може да бъде серия от байтове, тъй като се изразява под формата на количество с променлива дължина.

3.5.5 Instrument

FF 04 len text

Името на инструмента, който пътеката изпълнява (текст). Може да се различава от името на последователността или пътеката. Примерно ако името на последователността е "Пеперуда", но пътеката се свири на пиано, то може да се включи име на инструмент "Пиано".

Най-добре е едно (или няколко) събитие да се намира в началото на MTrk за да предоставя информация на потребителя какъв инструменти изпълнява пътеката. Обикновено инструментите (пачове, тонове, банки и т.н.) са заложени от звуковите устройства чрез събития MIDI Program Change в MTrk, а в частност това се осъществява от General MIDI Sound Modules. Това събитие основно съществува за да предостави на потребителя визуална обратна връзка за инструмента използван в пътеката.

Трябва да се отбележи, че len може да бъде серия от байтове, тъй като се изразява под формата на количество с променлива дължина.

3.5.6 Lyric

FF 05 len text

Лириката на песента (текст) който се показва на конкретен такт. Единично лирично мета-събитие трябва да съдържа само една сричка.

Трябва да се отбележи, че len може да бъде серия от байтове, тъй като се изразява под формата на количество с променлива дължина.

3.5.7 Marker

FF 06 len text

Маркер (текст) който се появява на определен такт. Събитията за маркери могат да се ползват за отбелязване начало и край на цикли (къде системата се връща назад, към предходно събитие).

Трябва да се отбележи, че len може да бъде серия от байтове, тъй като се изразява под формата на количество с променлива дължина.

3.5.8 Cue Point

FF 07 len text

Точка на стартер (текст) която се появява на даден такт. Точката на стартер може да се използва за определяне къде започва изпълнението на WAVE файл (семплиран звук), където текстът е името на файла.

Трябва да се отбележи, че len може да бъде серия от байтове, тъй като се изразява под формата на количество с променлива дължина.

3.5.9 MIDI Channel

FF 20 01 cc

Това незадължително събитие обикновено се появява в началото на MTrk (преди всички събития с ненулево време и всички мета-събития, с изключение на събитието за брой последователности) и определя към кой MIDI канал всички последователни мета-събития или системни събития се асоциират. Байтът данни cc е номерът на MIDI канала, където 0 се взема първия канал.

MIDI спецификацията не дава MIDI канал за изключителните системни събития. Мета-събитията също нямат предназначен канал. Когато се създава MIDI файл от тип 0 всички изключителни системни събития и мета-събития отиват в една пътека, така че е трудно да се асоциират тези събития със съответно MIDI звуково съобщение (примерно ако се именува музикална партитурата на MIDI канал 1 “Соло флейта” и партитурата на MIDI канал 2 “Соло тромпет” трябва да се използват 2 мета-събития по имена на пътеки, докато и двата вида събития ще са в една пътека при файл от тип 0 за да се различават кое име на пътека е асоциирано в кой MIDI канал то се поставя MIDI мета-събитие за канал с номер на канал 0 пред събитието за името на пътеката “Соло флейта” и се поставя друго MIDI канал мета-събитие с номер на канал 1 преди мета събитието за име на пътека “Соло тромпет”).

Приемливо е да има повече от едно събитие за MIDI канал в дадена пътека, ако тази пътека трябва да асоциира различни събития с различни канали.

3.5.10 MIDI Port

FF 21 01 pp

Това незадължително събитие обикновено се появява в началото на MTrk (преди всички събития с ненулево време и всички MIDI събития) определя на кой MIDI порт (гнездо) отиват MIDI събитията от MTrk. Байтът с данни pp е номерът на порта, като 0 се смята за първи порт в системата.

MIDI спецификацията има лимит от 16 канала за всеки MIDI вход/изход (порт, гнездо, жак или какъвто и термин да се използва за описание на за единичен MIDI вход/изход). Номерът на MIDI канала за дадено събитие е кодиран в младшите 4 бита в статус байта. Поради тази причина номерът на канала е винаги между 0 и 15. Много MIDI интерфейси имат множество MIDI входове/изходи за да заобиколят ограниченията на честотната лента (позволява MIDI данните да бъдат

изпращани/получавани по-ефективно към/от няколко външни модула) и да дадат на музикантите повече от 16 MIDI канала. Също някои секуенсъри поддържат повече от един MIDI интерфейс използван за едновременен вход/изход. За съжаление няма начин да се кодират повече от 16 MIDI канала в MIDI статус байта, така че има нужда от метод за идентифициране на събитията които трябва да се изпратят, примерно на канал 1 на втория MIDI порт, а не на канал 1 на първия MIDI порт. Това мета-събитие позволява на секуенсъра да определи кои MTrk събития трябва да се изпратят на кой MIDI порт. MIDI събитията, следващи мета събитието за смяна на порт биват изпращани на оказания в това събитие порт.

Приемливо е да има повече от едно събитие за смяна на порт в дадена пътека, ако пътеката трябва да изпраща данни на различни портове в различни моменти от времето.

3.5.11 End of Track

FF 2F 00

Това събитие е задължително. Трябва да се появи като последно събитие във всяко MTrk парче. Използва се като дефиниран маркер за край на MTrk. Може да има само едно такова събитие за всяко MTrk.

3.5.12 Tempo

FF 51 03 tt tt tt

Обозначава промяна в темпото. Трите байта с данни tt tt tt са темпото в микросекунди за четвъртина нота. С други думи стойността на темпото в микросекунди показва колко дълго трябва да просвирва секуенсърът ноти

четвъртини. Примерно, ако са представени три байта за темп 07 A1 20, то тогава всяка нота четвъртина трябва да бъде 0x07A120 (или 500 000) микросекунди дълга.

И така, MIDI файловият формат представя темпото като количество време (микросекунди) за четвъртина нота.

3.5.12.1 BPM

Обичайно музикантите представят темпото като брой ноти четвъртини за една минута (период време). Това е противоположно на начинът по който MIDI файловият формат представя темпото.

Когато музикантите говорят за “такт” в термините на темпо, то те имат предвид четвъртина нота (четвъртина нота е винаги един такт, когато се говори за темп, без значение от времевата сигнатура). Може да бъде малко объркващо за хора, които не са музиканти, това че такт от времевата сигнатура може да не е същото като такт за темпото, но може и да бъдат еквиваленти, ако тактът във времевата сигнатура се окаже с дължина четвърт нота. Все пак, това е традиционната дефиниция за BPM темпо. За музикантите, темпо винаги означава колко четвъртини ноти могат да се изсвирят в рамките на една минута. Музикантите обозначават това темпо с BPM (Beats Per Minute). И така, темпо от 100 BPM означава, че музикант трябва да може да изсвири 100 равномерни ноти четвъртини, една след друга за една минута. Това показва колко бързо е музикалното темпо при 100 BPM. Много е важно да се разбере концепцията по която музикантите изразяват музикалното темпо (BPM), така че правилно да се представят настройките за темпото на музикантите и също за да се направи връзката кака MIDI файловия формат представя темпото.

За да се конвертира темпото от MIDI файловия формат (трите байта указват броя микросекунди за четвъртина нота) към BPM:

BPM = 60 000 000 / (tt tt tt)

Примерно, темпо от 120 BPM = 07 A1 20 микросекунди за четвъртина нота.

И така, защо MIDI файловият формат използва темпо “време за четвъртина нота” вместо “брой четвъртини ноти за период от време”? Отговорът е, че по-лесно може да се специфицира времето в този формат. С BPM понякога се налага да се работи с разделяне на темпото (примерно 100.3 BPM) ако е необходима по-фина резолюция за темпото. Използването на микросекунди за изразяване на темпото предлага изключително големи възможности за резолюция.

Също така SMPTE е времеви базиран протокол (базира се на секунди, минути, часове, а не на музикалното темпо). Поради тази причина е по-лесно да се свърже темпото от MIDI файловия формат към SMPTE времето, ако се представя в микросекунди. Много музикални устройства използват SMPTE за синхронизиране на плейбека.

3.5.12.2 PPQN Clock

Секуенсърът обикновено използва някакъв първичен хардуерен брояч за устойчиво време (микросекунди в повечето случаи) за да генерира софтуерен PPQN Clock (Pulses Per Quarter Note), който брои времевата база (деление) за тактове. По този начин времето в което едно събитие трябва да се появи може да се представи на музиканта в термините на музикални такта, вместо колко микросекунди са изминали от старта на плейбека. Трябва да се има предвид, че музикантите винаги разсъждават в термините на тактове, а не период секунди, минути и т.н.

Както бе споменато, стойността за темпо в микросекунди показва колко дълга трябва да е всяка нота четвъртина за секуенсъра. От тази позиция може да се определи колко дълги трябва да са PPQN тактовете, чрез разделяне стойността на микросекундите на делението прието в MIDI файла. Примерно, Ако делението в MIDI файла е 96 PPQN, то това означава, че всички PPQN тактове на секуенсъра при

горното темпо трябва да бъдат 500 000 / 96 (или 5208.3) микросекунди дълги (трябва да има 5208.3 микросекунди между всеки PPQN такта за да се постигне темпо от 120 BPM при 96 PPQN), също така трябва винаги да бъдат 96 такта на този часовник във вскяа нота четвъртина, 48 такта във всяка нота осмина, 24 такта във всяка нота шестнадесетина и т.н.

Трябва да се отбележи, че може да има всяка времева база за всяко темпо. Примерно може да има 96 PPQN файл, който се изпълнява на 100 BPM, точно както може да има файл 192 PPQN, който се изпълнява на 100 BPM. Също така може да има файл на 96 PPQN, който се изпълнява на 100 BPM или 120 BPM. Времева база и темпо са две напълно отделни количествени мерки. Разбира се, и двете мерки са необходими, когато се настройва таймерът на хардуера (това става с настройване от колко микросекунди се състои един PPQN такт). И разбира се, на по-бавно темпо PPQN тактовете ще са по дълги в сравнение с тези на по-бързо темпо.

3.5.12.3 MIDI Clock

Данни MIDI часовника се изпращат през MIDI, за да се синхронизира работата на 2 устройства (едното устройство генерира MIDI тактове от часовника на неговото текущо темп, което се брои вътрешно, а второто устройство синхронизира плейбека според получените байтове за тактуването на часовника). За разлика от SMPTE кадрите, байтовете от MIDI тактовия часовник се изпращат на норма определена от музикалното темпо.

Докато има 24 MIDI часовникови такта във всяка нота четвъртина дължината на MIDI часовниковите тактове (времето между всяко MIDI съобщение за тактовете на часовника) е темпото в микросекунди разделено на 24. В горния пример това ще бъде 500 000 / 24 или 20833.3 микросекунди за всеки MIDI такт на часовника. Като алтернатива може да се отнесе към времевата база (PPQN тактове). Ако е зададено 96 PPQN, то това означава, че байтовете за MIDI тактовете на часовника трябва да се появяват на всеки 96 / 24 (или 4) PPQN такта.

3.5.12.4 SMPTE

SMPTE (Society of Motion Picture and Television Engineers) времеви код преброява фрагмент от време в термините на секунди, минути и часове (начинът по който не-музикантите броят времето). Също така се прави разбиване на секундите в по малки единици наречени “кадри”. Филмовата индустрия създава SMPTE, при което се възприемат 4 различни кадрови мерки. Секундата може да се раздели на 24, 25, 29 или 30 кадъра. По-късно дори по-фина резолюция е била нужна за музикалните устройства, така че всеки кадър се разделя на “под-кадри”.

И така, SMPTE не е директно свързан с музикалното темпо. SMPTE не се мени с музикалното темпо.

Много устройства използват SMPTE за да синхронизират плейбека. Ако трябва да се синхронизира с такова устройство, то най-вероятно ще трябва да се ползва SMPTE време. Разбира се, най-вероятно ще трябва паралелно да се поддържа някакъв вид PPQN брояч, базиран на преминаващите SMPTE под-кадри, така че потребителя да може да настройва темпото на плейбека в термините на BPM и да разглежда времето за всяко събитие в термините на музикални тактове. Въпреки това SMPTE не се свързва директно с музикалното темпо, то трябва да се интерполира (пресметне) съответстващото PPQN тактуване от преминаващите под-кадри/кадри/секунди/минути/часове (точно както предходно бе калкулирано PPQN тактуването от хардуерния брояч към микросекунди).

Един опростен пример с 25 кадъра и 40 под-кадъра. Както бе споменато във фрагмента за деленето този избор е еквивалентен на време-отчитане в милисекунди защото има 1 000 SMPTE под-кадъра в секунда. Тоест 25 кадъра в секунда, при 40 под-кадъра в кадър, което прави 25 * 40 под-кадъра в секунда. Също така 1 000 милисекунди се равняват на една секунда. В тази ситуация всяка милисекунда

означава, че е изминал един под-кадър (и обратното). Също така, при преброяването на 40 под-кадъра отминава един SMPTE кадър (и обратното).

При предположение, че са желани 96 PPQN и темпо от 500 000 микросекунди. Вземайки се предвид, че 25-40 (кадри - под-кадри) SMPTE време-броене 1 милисекунда = 1 под-кадър (също така 1 милисекунда = 1 000 микросекунди), то трябва да има 500 000 / 1 000 (или 500) под-кадъра за всяка нота четвъртина. При положение, че са налични 96 PPQN за всяка нота четвъртина, то се оказва, че всеки PPQN е 500 / 96 под-кадъра дълъг или 5.2083 милисекунди (по този начин се получават 5208.3 микросекунди за PPQN такт както се вижда във фрагмента обсъждащ PPQN тактуването). Докато 1 милисекунда = 1 под-кадър, то всеки PPQN такт ще е еквивалентен на 5.2083 под-кадъра при горното темпо и времева база.

3.5.12.5 Пресмятания

BPM = 60 000 000 / MicroTempo
MicrosPerPPQN = MicroTempo / TimeBase
MicrosPerMIDIClock = MicroTempo / 24
PPQNPerMIDIClock = TimeBase / 24
MicrosPerSubFrame = 1000 000 * Frames * SubFrames
SubFramesPerQuarterNote = MicroTempo / (Frames * SubFrames)
SubFramesPerPPQN = SubFramesPerQuarterNote / TimeBase
MicrosPerPPQN = SubFramesPerPPQN * Frames * SubFrames

3.5.13 SMPTE Offset

FF 54 05 hr mn se fr ff

Обозначава SMPTE начално време (часове, минути, секунди, кадри, под-кадри) на MTrk. Трябва да бъде в началото на MTrk. Часът не бива да се кодира в

SMPTE формат както е в MIDI времевия код. При файлове от тип 1 SMPTE отместването трябва да се съхранява с карта на темпото (намира се в първото MTrk парче) и няма значение в никое от останалите MTrk парчета. Полето ff съдържа кадрово деление в стотици от кадъра, дори в SMPTE базираните MTrk парчета които определят различно деление на кадрите за делта времена (различни от настройките определени в MThd парчето).

3.5.14 Time Signature

FF 58 04 nn dd cc bb

Времевата сигнатура се представя от 4 числа nn и dd представляват числител и знаменател на сигнатурата както се записва в музиката на листи. Делителят е отрицателна степен на 2: 2 = четвърт нота , 3 = осмина нота и т.н. Числото cc обозначава броят на MIDI тактовете за един такт на метронома. Числото bb определя броя на нотираните 32 ноти в MIDI четвърт нота (24 MIDI такта). Това събитие позволява на софтуера да определи какво MIDI възприема като четвъртина или нещо съвсем различно. Примерно, 6/8 време с такт на метронома 3 на всяка осма нота и 24 такта за четвърт нота ще бъде представено по следния начин:

FF 58 04 06 03 18 08

3.5.15 Key Signature

FF 59 02 sf mi

sf = -7 за 7 бемол, -1 за 1 бемол и т.н., 0 за клавиш с, 1 за 1 диез и т.н.

mi = 0 за мажор, 1 за минор

3.5.16 Proprietary Event

FF 7F len data...

Това събитие може да се ползва от софтуера за съхранение на собствени данни. Първият байт (байтове) трябва да бъде уникален идентификатор (ID) от някакъв вид, който програмата може да определи, че събитието и принадлежи или е на някоя друга програма. Препоръчителни са 4 символа (ASCII), като идентификатор.

Трябва да се отбележи, че len може да бъде серия от байтове, тъй като се изразява под формата на количество с променлива дължина.

3.6 Някои особености

При файлове от тип 0 промените в темпото и времевата сигнатура са разпръснати в едно MTrk парче. При файлове от тип 1 най-първото MTrk парче трябва да се състои само от събитията за темпо и времева сигнатура, така че да може да бъде четено от някои устройства способни да генерират карта на темпото. При файлове от тип 2 всяко MTrk парче трябва да започва с най-малко едно събитие за темпо и времева сигнатура, като инициализация.

Забележка: Ако не е посочено темпо и времева сигнатура по подразбиране се приема 120BPM и 4/4.

4 Диференциална еволюция

Диференциална еволюция (DE) е метод за математическа оптимизация при многомерни функции и принадлежи към класа на оптимизаторите с еволюционна стратегия. Диференциалната еволюция се ползва за намирането на глобални оптимуми при многомерни и многомодални (с повече от един оптимум) функции с добра степен на вероятност.

Основната идея зад диференциалната еволюция е схемата за получаване на параметрите за вектора с който се извършва модификацията. При диференциалната еволюция вектор с претеглената разлика (умножаване с подходящо избран тегловен коефициент) между два други вектора от популацията се добавя към трети вектор. По този начин не е нужно да се използва специално вероятностно разпределение, което прави тази схема напълно самоорганизираща се. [5]

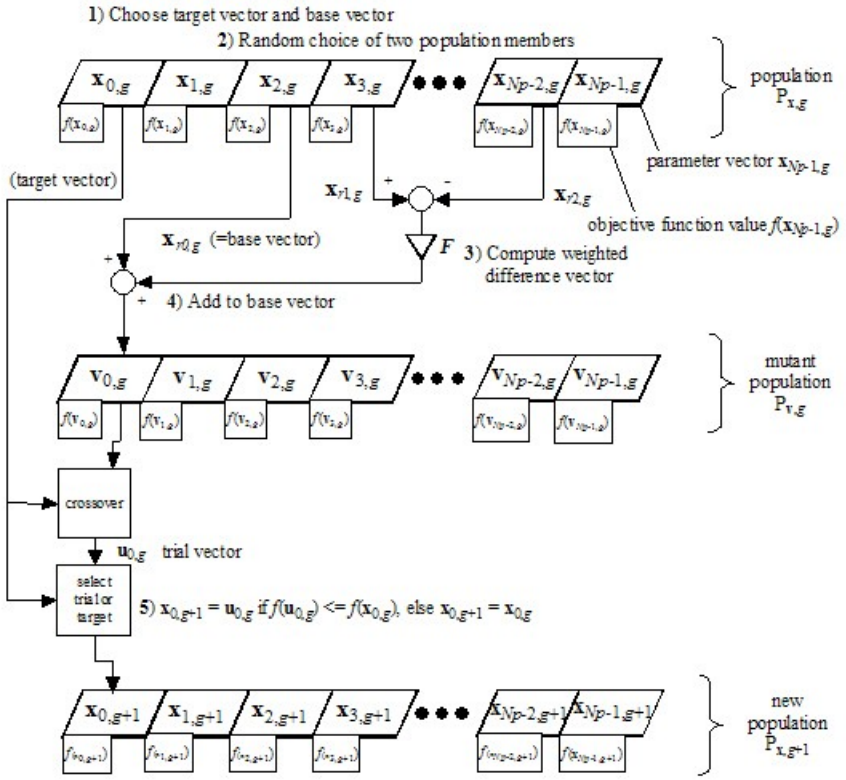
Настоящата секция представлява буквален превод на идеите заложени в диференциалната еволюция от официалната страница, посветена на този оптимизационен метод.

4.1 Исторически бележки

Диференциалната еволюция е породена от опитите на Ken Price да разреши проблема за напасване на полином на Чебишев, който му е бил предложен от Rainer Storn. Пробивът се е получил когато Ken стигнал до идеята, че може да използва диференциален вектор (вектор разлика) за малка промяна на конкретен вектор от популацията. Методът в съвременния му вариант възниква след интензивна комуникация между двамата автори.

4.2 Основна идея

Диференциалната еволюция е прост стохастичен минимизатор базиран на популация, но в същото време е много мощно средство за оптимизация. Основната идея в метода е начинът по който се образува резултатния вектор за формирането на новата популация.



Фиг. 4.1 Диференциална еволюция - концептуална схема.

Основно, при диференциалната еволюция се добавя претеглен (умножение с тегловен коефициент) диференциален вектор към трети вектор. По този начин не е нужно да се знае каквото и да било за вероятностното разпределение на проблема и съответно не е нужно да се използва конкретно вероятностно разпределение, което прави тази схема напълно самоорганизираща се. [6]

На първа стъпка се избира вектор цел (векторът, който ще бъде променен). На втора стъпка, по абсолютно случаен принцип, се избират два други вектора от популацията. На трета стъпка се изчислява претеглен диференциален вектор (разликата между двата вектора умножена с тегловен коефициент). На четвърта стъпка избран базов вектор се сумира с претегления диференциален вектор. На пета стъпка се кръстосва целевия вектор и модифицирания базов вектор. На шеста стъпка се извършва селекцията, като се изчислява целевата функция за новополучения вектор след кръстосването и целевия вектор за промяна. От двата вектора за влизане в новата популация се избира този, който даде по-оптимален резултат на целевата функция. Алгоритъмът се изпълнява в цикъл от епохи, като една епоха включва обновяването на всички елементи в популацията.

4.3 Модификации за нуждите на разработката

В настоящата разработка са направени серия от модификации на алгоритъма за диференциална еволюция с цел максимално подобряване на метода за нуждите на решавания проблем.

Една от основните модификации е използването на диференциална еволюция като алгоритъм в разпределена среда. Тази модификация е продиктувана от нуждата множество потребители/слушатели да прослушват различните мелодии в един и същи момент от времето.

Втората много важна модификация е замяната на функцията за сравнение от етапа на селекцията с оценъчна функция базирана на човешкия субективен критерии за естетика на прослушаната мелодия.

Третата изключително съществена модификация е премахването на класическото кръстосване като операция над векторите в популацията. Ако бъде приложено класическото кръстосване над вектор от музикални ноти, то това би

разрушило хармонията в музикалната мелодия и би влошило многократно постигнатите резултати, като обезсмисли използването на алгоритъма като цяло. Поради тези причини класическото кръстосване не се прилага.

Последната четвърта, и не толкова съществена модификация, е използването на чист диференциален вектор (тоест не се използва тегловен коефициент за умножение). Тъй като разработката се стреми към целочислена оптимизация (информацията за музикалните ноти е представена изцяло от цели положителни числа), то въвеждането на теглови коефициенти би усложнило първоначалния прототип. Разбира се тегловите коефициенти са изключително важен параметър за сходимостта на алгоритъма, но тъй като целевата функция представлява субективна човешка оценка, то наличието на фино регулирани тегловни коефициенти не би подобрило драстично работата на системата. Въпреки всичко на бъдещ етап от разработката е добре да бъдат добавени коефициенти за претегляне на диференциалния вектор и по този начин регулиране на стъпката с която се обхожда многомерното пространство.

5 Релационна система за управление на бази данни PostgreSQL

В настоящата секция ще бъде разгледана на кратко системата за управление на бази от данни PostgreSQL.

5.1 Общи понятия

PostgreSQL е обектно-релационна система за управление на бази от данни. PostgreSQL се разпространява под разновидност на BSD лиценз и по същество представлява софтуер на свободно разпространение. Както много други програми с отворен код, PostgreSQL не се контролира от никоя конкретна компания, но се осланя на глобална общност от разработчици и голям брой компании, които я развиват. [7]

5.2 Технически характеристики

PostgreSQL се изпълнява под всички популярни операционни системи, включително Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64) и Windows. Системата е напълно ACID (Atomicity, Consistency, Isolation, Durability) съвместима, като има пълна поддръжка на чужди ключове, добавяния (joins), изгледи, тригери и споделени процедури (на много различни езици). Включва повечето SQL92 и SQL99 типове данни, включително INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL и TIMESTAMP. Също се поддържа съхраняването на големи бинарни обекти, включително картини, звуци или видео. Има базов програмен интерфейс за C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC.

PostgreSQL е СУБД от корпоративен клас и включва сложна функционалност, като MVCC (Multi-Version Concurrency Control), точки за възстановяване във времето, пространства за таблици, асинхронна репликация, вложени транзакции (точки на запазване), он-лайн/топли бакъпи, усложнен планировчик/оптимизатор на заявките и система за логове при поява на грешки. СУБД-то поддържа международни таблици със символи, кодиране на символите в множество байтове на символ, Unicode. Съобразява се с локалното сортиране, чувствителност към малки и големи букви и форматиране. Системата е с висока степен на разширяемост както на данни в явен вид, които може да управлява, така и в броя потребители, които може да обслужва едновременно. [8]

5.3 Избор на СУБД

При избор на СУБД с отворен код алтернативата е между две особено популярни системи, а именно MySQL и PostgreSQL. За целите на сравнението може да се използва таблица от характеристиките за PostgreSQL 8.0 и MySQL 5.0 (beta). [9]

	PostgreSQL 8.0	MySQL 5.0 (beta)
Операционна система	Windows, Linux, BSDs, HP-UX, AIX, OS X, Unixware, Netware и др.	Linux, Windows, FreeBSD, MacOS X, Solaris, HP UX, AIX и др.
ANSI SQL съвместимост	ANSI-SQL 92/99	ANSI съместим (ANSI mode)
Производителност	Бавна	Несигурна
Sub-selects	Да	Да
Транзакции	Да	Да
Репликация на базата данни	Да	Да

Външни ключове	Да	Да
Изгледи	Да	Да
Съхрнанени процедури	Да	Да
Тригери	Да	Да
Обединения	Да	Да
Пълни добавяния (joins)	Да	Не
Ограничения	Да	Не
Курсори	Да	Частично
Процедурни езици	Да	Да
Почистване	Да	Да
Различни формати файлове	Не	Да
ODBC	Да	Да
JDBC	Да	Да
Други API-та	Повечето програмни езици	Повечето програмни езици
IPv6 Поддържка	Да	Не
Инсталация	PostgreSQL инсталация	MySQL инсталация

5.4 Съхранението на данни в приложението

За нуждите на разработката в настоящия етап от развитието на програмата се използва единствено пълно съхранение на данните в системата (при спиране на приложението сървър) и прочитането им от базата данни при зареждане на системата (приложението сървър). Въпреки че тази изключително лека задача за боравене с данни може да се реализира както със стандартни бинарни файлове, на операционната система, така и с някоя по-опростена СУБД, изборът бе направен на база бъдещите възможности за развитие на системата, при което всички данни се

съхраняват в PosgreSQL база данни. При разработката изключително полезни се оказаха възможностите за работа със съхранени процедури.

6 Java аплети

В настоящата секция ще бъде разяснено какво е Java аплет и каква е ползата от използването на аплети в разработеното приложение.

6.1 Основни понятия

Аплет е програма написана на програмния език Java, която може да бъде включвана в HTML страница, почти по същия начин както се включва изображение в страницата. Когато се използва уеб брузър, който поддържа Java вградената технология за разглеждане на страница съдържаща аплет. Аплетът се прехвърля на локалната система и браузърът стартира изпълнимия код с помощта на Java виртуалната машина (JVM). [10]

6.2 Концепцията на пясъчната кутия

Оригиналният модел за сигурност, предоставен от Java, е познат под названието модел на пясъчната кутия, който съществува с идеята да предложи много ограничена среда в която да се изпълнява несигурен код, придобит от отворената мрежа. Есенцията на модела пясъчна кутия е в това, че кодът на локалната машина е доверен и може да има пълен достъп до всички системни ресурси (примерно файловата система), докато кодът свален от отдалечен сървър (примерно аплет) не е доверен и може да има достъп само до ограничени системни ресурси, които са разположени вътре в пясъчната кутия.

В JDK 1.1 (Java Development Kid) е представена концепцията за подписани аплети. При публикуване, дигитално подписаните аплети се смятат за доверен код, точно както се смята за доверен кодът на локалната машина, ако ключът на

цифровия подпис се разпознае като доверен от крайната система, която получава аплета. Подписаните аплети, заедно с техния подпис се получават в JAR (Java Archive) формат и се изпълняват извън пясъчната кутия. В JDK 1.1 неподписаните аплети продължават да се изпълняват само в рамките на пясъчната кутия. [11]

6.3 Аплетът от страна на крайния потребител

На професионален жаргон аплетите се определят като “лек” клиент. Лек клиент е клиентска програма, която не се нужда е от инсталация на потребителската машина и в повечето случаи се изпълнява в друга програма контейнер (в случая уеб браузър). Изключителното предимство на леките клиенти е в липсата на нужда за инсталация, при което дори не особено технически грамотен потребител може да зареди програмата и да борави с нея по класически начин, към който е привикнал работейки с услугата WWW.

В разработеното приложение е необходимо аплетът да бъде дигитално подписан, тъй като клиентската програма комуникира с програмата на сървъра през механизма RMI, а той от своя страна е базиран на комуникация през стандартни TCP сокети. Моделът на пясъчната кутия ограничава неподписаните аплети и не им позволява да установяват TCP сокет връзки извън рамките на пясъчната кутия. Чрез дигитално подписване това ограничение се заобикаля и се постига пълният комфорт на програмен код изпълняващ се в рамките на стандартен Java съвместим уеб браузър.

В настоящата разработка аплетите не са дигитално подписани, а за целите на тестването се използва подходящо избран Java Policy файл, който дава широки права на достъп за аплетите изпълнявани в стандартното JDK приложение Applet Viewer.

7 Софтуер с отворен код

Настоящата секция е посветена на софтуерните продукти с отворен код и е буквален превод от материали публикувани в Интернет. [12] В процеса на разработката са използвани единствено развойни средства и среди базирани на отворен код. По своята същност разработеното приложение също е публикувано с отворен код, под GPL лиценз, в едно от най-големите хранилища за проекти с отворен код SourceForge Net. Повече информация е налична в страницата на проекта: http://www.sourceforge.net/projects/mididermi/

7.1 Основни понятия

Базовата идея за отворения код е много проста. Когато програмистите могат да четат, разпространяват и модифицират програмен код в части от софтуера, то софтуерът се развива. Хората го подобряват, хората го адаптират, хората поправят дефекти открити в него. Разработването на отворен код идеално съвпада с инфраструктурата на глобалната мрежа Интернет и все повече се разпространява. Има потенциала да се движи със скорост, която може да засрами комерсиалните разработки.

7.2 Разлика между отворен код и комерсиален софтуер

Софтуерът с отворен код е такъв софтуер, чиито програмен код е напълно достъпен безплатно. Потребителите са свободни да правят подобрения и да разпространяват, докато спазват определени условия (повече подробности в раздела за лицензи на отворен код). Най-популярният, в настоящия момент, софтуер с отворен код е операционната система Linux.

Обратно, програмния код на комерсиалния софтуер основно се пази в тайна. Потребителите си поръчват само компилираната версия на комерсиалния софтуер и няма друг избор, освен да използва софтуерът такъв какъвто му е доставен.

7.3 Разликата между програмен код и компилиран код

Програмен код е текст написан на програмен език от високо ниво, който хората програмисти използват за изграждането на компютърни програми. Всеки обучен в конкретен програмен език (примерно Java или C/C++) може да разбира и да редактира програмния код на този език.

Компилиран код е програмен код транслиран на език, който се разбира от компютрите (компилирания код също се нарича двоичен код). Хората не могат да разбират или редактират компилирания код. Дори специализирани програми, проектирани за възстановяване на програмен текст от компилиран код, не могат да възстановят идеален програмен текст от компилиран код.

7.4 Как действа отворения код

Отвореният код е социално центриран модел за разработка (общност от хора, обединени около обща идея), който окуражава свободното разпространение на знания умения между всички членове. Моделът на отворения код не се свързва с организации и централен контрол, като ги заменя с отворена мрежа от индивиди. Всеки индивид може да надгражда върху работата свършена от други хора в мрежата, като не се губи време за преоткриване на вече открити неща.

През последните години, свързването между хората се разширява, чрез огромния капацитет на Интернет, като скорост за предаване на данни. Поради ефективната мрежова инфраструктура моделът за сътрудничество при отворения

код има неограничен потенциал. Всъщност, през последните няколко десетилетия лицензите за отворен код обхващат целия свят и вече се използват за хиляди компютърни програми.

7.5 Разликата между свободен софтуер и отворен код

Разработката на свободен софтуер е мотивирана от алтруистично желание за подобряване на обществото като цяло - обществото е на първо място, а индивидуалната икономическа изгода на второ място (най-меко казано). Разработката на отворен код е мотивирано от вярата, че този модел на разработка е по-съвършен от комерсиалния модел. Отворения код е разклонение на свободния софтуер. При отворения код се залага на повече прагматичност, отколкото догматичност при модела за разработка, което го прави много по привлекателен за повечето разработчици.

7.6 Лицензи за отворен код

Традиционно авторските права са предназначени за запазване на правата за продажба (всички права запазени) във връзката с вид оригинална работа. Противоположно, лицензите за отворен код са юридически инструмент използван да направи една разработка свободно достъпна. Терминът “свободно” не е еквивалентен на термина “безплатно”. Този термин се отнася до свободата на потребителя да стартира, копира, разпространява, изучава, променя и подобрява софтуерът без да се нуждае от изрично разрешително.

Лицензите за отворен код използват приложението на авторските права върху компютърните програми за да осигурят няколко стандартни условия. Всеки може да копира, разпространява и модифицира софтуер с отворен код, докато се съобразява с условията. Условията осигуряват, че успешната разработка на програмния код ще

остане достъпна за бъдещо подобряване. Всеки който нарушава условията на лицензa може да бъде обект на съдебно преследване, според законите за авторски права.

7.7 Определяне на продуктите с отворен код

Няма стандартен лиценз който софтуерът трябва да използва за да бъде определен като отворен код. OSI (Open Source Initiative) служи като определящ дефиницията за софтуер с отворен код. По настояще съществуват над 40 различни лицензи за отворен код, които покриват дефиницията на OSI. Десетте критерия за софтуер с отворен код са обяснени в официалния сайт на OSI. За да бъде един софтуер определен като софтуер с отворен код то разработчиците трябва покрият и десетте критерия.

7.8 Разликата в различните лицензи

Широкото разпространение на лицензите за отворен код предлага голямо разнообразие под формата на ограничения и специфики. В най-либералният край на спектъра е BSD (Berkeley Software Distribution) лицензът, който позволява създаването на частни затворени разработки (включително комерсиален софтуер с непубликуван програмен код) и не изисква промени в публичната версия да бъдат публикувани в каквато и да е форма.

В другия консервативен край на спектърът е GNU GPL (General Public Licence) лицензът, който задължава свободно разпространение, без заплащане или допълнителни лицензни клаузи на програмния код за всяка производна разработка.

Между BSD и GPL, в спектъра от ограничения, се намира MPL (Mozilla Public Licence) лицензът. Промени в програмен текст под MPL трябва да бъдат направени

публично достъпни в Интернет. Лицензът MPL не е вирусен (да поражда нуждата производната работа да бъде публикувана под същия лиценз), позволява лицензиране под друг лиценз и възможност работата да не бъде публикувана изобщо.

7.9 Предимства на софтуера с отворен код

Разработчиците на отворен код имат възможността да надграждат работата на други разработчици. Наличието на софтуер с отворен код позволява на световната общност от разработчици на отворен код да участва наравно в разпространението, преглеждането и производството. Една програма може да бъде подобрявана и разпространявана до безкрайност, носейки полза за цялата общност. Колкото повече се разширява отвореността на модела за отворен код, токова повече ще се подобрява качеството на продуктите с отворен код.

Проблемът с цялостното качество, за който софтуерът с отворен код има четири наследствени предимства, пред комерсиалния софтуер. Първи, софтуерът с отворен код е значително по евтин в сравнение с комерсиалните алтернативи. Второ, наличието на програмния код позволява на потребителите да откриват и отстраняват дефекти, също софтуерът може да бъде модифициран за специфичните нужди на конкретен потребител, като надграждането се случват по избор на потребителя, а не по избор на производителя. Трето, прозрачността в софтуера с отворен код подобрява сигурността като цяло, тъй като слаби места в сигурността може бързо да се откриват и отстраняват. Четвърто, отворения код позволява на потребителите да бъдат гъвкави при избора на производители. Ако потребителите не са доволни от версията на продукт получен от един доставчик с лекота могат да преминат към еквивалентен продукт от друг доставчик. Това предпазва потребителите от изпадане в зависимост към конкретен доставчик или договор за поддръжка.

7.10 Недостатъци на софтуера с отворен код

Първият основен недостатък е възможността за многократно нарушаване на авторските права. Един обичаен проект се състои от сътрудничество между много хора. Почти невъзможно е да се проследи целия програмен текст за нарушаване на предходни лицензионни ограничения. Това създава много възможности за участници в проекта да включат програмен код в нарушение. Този риск в процеса на разработка до голяма степен е създаден от съществуващите лицензи. Участниците не отговарят за нарушенията в кода, който те включват към проекта. Всъщност, стандартните лицензи за отворен код са проектирани, така че максимално да защитават участниците в проекта. Стандартните лицензионни споразумения не включват никакви изявления за интелектуалната собственост, гранация или обезщетения, но вместо това съдържат широк списък с отказване от права гарантиращи изгода за ползвателите/сътрудниците.

Вторият голям недостатък е липсата на гаранции за качество или работоспособност. Големите проекти с отворен код имат някаква изградена йерархия, която се грижи за качеството на кода, проследява процеса на разработка, следи за дефекти и тяхното отстраняване. Други по-малки проекти често са продукт на любители, не се радват на същото качество на програмния код и стриктен протокол за тестване. Без договорно обвързване за качество и работоспособност ползвателят трябва да приеме рисковете, че софтуерът може да съдържа фатални дефекти, вируси или други проблеми, които могат да доведат до сериозни финансови загуби.

И не на последно място, третият недостатък на софтуера с отворен код се отнася до лицензите с отказване от права (Copyleft Licencing). Някои лицензи, като GPL, задължават всички възползващи се от продукта да предоставят копия без печалба на тяхната производна работа, под формата на програмен текст, за използване, модифициране и разпространение от широката публика, съобразно с клаузите в първичното лицензионно споразумение. Тези лицензионни условия

създават много трудности за фирмите в производството на комерсиален софтуер да използват подобни решения базирани на отворен код, като основа за бизнеса си. В резултат на което, фирмите се отнасят към употребата на такива продукти с отворен код (или свързването към такива продукти на техния комерсиален софтуер), като потенциална заплаха, че може да бъде предизвикано преобразуване на целия програмен текст в производна работа от продукта с отворен код, което може да предизвика преобразуването на комерсиалния продукт в продукт без печалба.

7.11 Настоящата разработка

За целите на настоящата разработка бе избран GNU GPL лицензa, тъй като разработката представлява своеобразен вид научно изследване, чиито новаторски идеи, методи и похвати не могат ефективно и икономически изгодно да бъдат интелектуално защитени по никакъв друг начин. Изборът е направен с идеята авторът/авторите да запазят всички права над своя интелектуален труд, но в същото време да се даде възможност за бъдещо развитие на системата и разпространение на натрупаните знания и умения.

8 Същност на разработеното приложение

8.1 Дефиниция на проблема

Учени, музиканти и експериментатори от векове се опитват да автоматизират процесът по който се създава музика. Тъй като музиката е свързана със субективните човешки критерии, изцяло автоматизиране на процеса по създаването й се оказва почти непосилна задача.

Най-опростената дефиниция на разглеждания проблем е система, която без никакъв вход или с оскъдни входни данни на изхода си предоставя музикална творба, която да покрива високите естетически критерии, които хората налагат при този вид творчество.

Проблемът който се поставя в настоящата разработка е създаването на разпределена система, която да композира мелодии, на база оскъдните данни, предоставени от крайните потребители (слушатели/оценители) и предварително заложен механизъм за еволюция.

8.2 Мотивация за избраното решение на поставения проблем

Навлизайки в света на компютрите всеки човек остава изумен от възможностите на които са способни тези машини. Съвсем уверено можем да твърдим, че появата на компютъра коренно променя динамиката на човешкия живот. Изчислителните машини и комуникационната техника полагат основите на това което сега наричаме глобално общество. Чрез компютъра всеки можем да комуникира с хора от цял свят. Може да се слуша любимата радиостанция или да се гледа

предпочитан телевизионен канал. Може да се слуша музика, да се гледа филми или да се играе компютърна игра, която понякога удивява със своята реалистичност.

Въпреки невероятните възможности на съвременната изчислителна техника, все още има дейности, които могат да се вършат само и единствено от човека. Компютърът може да показва картини нарисувани от човек, но не може да нарисува нова картина. Компютърът може да изсвири музикално произведение, но не може да напише нова мелодия. За добро или зло, границите на съвременните компютри опират точно до творчеството. Тъй като творческите възможности на човека са най-висшата нервна дейност, най-вероятно компютрите още дълги години няма да достигнат това ниво. Въпреки всичко създателите на нов хардуер и софтуер не се отказват от надеждата, че компютърът някой ден ще може да прави всичко на което е способен човекът.

Един често използван подход за създаване на музика, генерирана автоматично от компютър, е базиран на идеята за генетични алгоритми. Може би една от най-интересните разработки е направена от Joy Schoenberger [13], която изцяло добронамерено е споделила своите програми с потребителите на Световната мрежа. Разработката е основана на MIDI стандарта за представяне на музикална информация. Прослушвайки няколкото примерни мелодии в страницата на Joy се забелязва колко неестествено звучат те. Този ефект се получава заради начина по който Joy оценява новото поколение хромозоми. За целите на оценката се използват таблици за хармонизиране, добре познати на повечето композитори. В тези таблици се съдържа информация кои тонове вървят в добро съчетание един след друг. Използвайки този подход постигаме хармонизиране в обкръжението от няколкото най-близки тона, но нямаме възможност да получим хармония за цялостната мелодия.

Друг много интересен проект, под заглавието Electric Sheep, е представен от Scott Draves [14]. Въпреки че Scott е насочил своето внимание в автоматизираното генериране на филмчета, то неговият опит дава изключително добра насока как да

се доразвие и подобри идеята на Joy, така че компютърно генерираната музика наистина да носи естетическите свойства, които хората възлагат на този вид произведения.

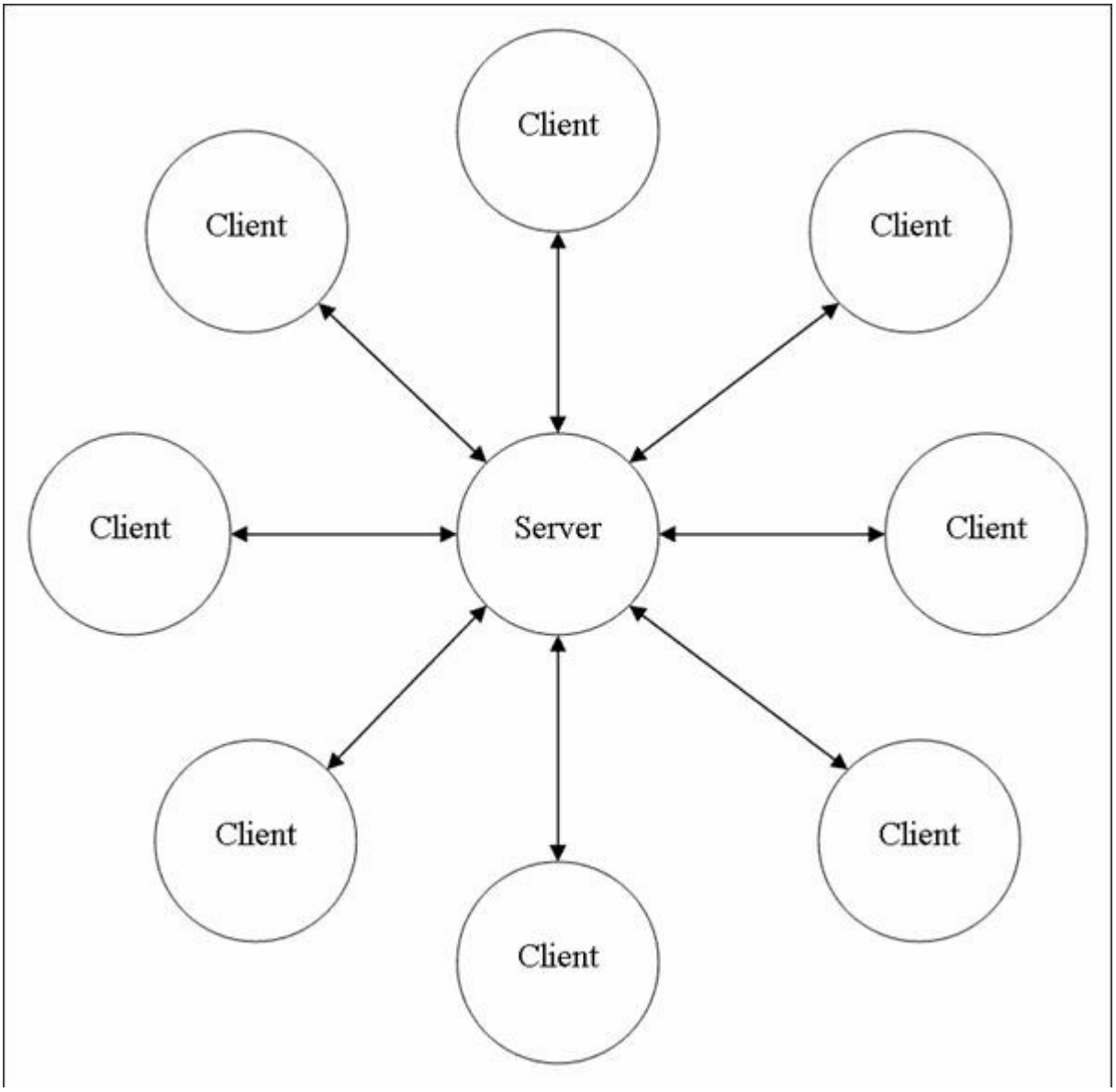
Вместо да се ползват таблици за хармонизиране, като оценъчна функция, оценката се възлага на хора, които прослушват автоматично генерираните мелодии. По този начин, всеки слушател би вложил индивидуалните си естетически критерии в оценката на мелодиите, създадени от алгоритъма за диференциална еволюция. За да бъде композирането по-бързо и обективно, изчисленията трябва да се извършват в разпределена среда, която да обединява оценките на различните слушатели при генерирането на нови мелодии.

Благодарение високата степен на паралелност, характерна за алгоритъма диференциална еволюция, има възможност да се създаде изключително ефективна разпределена система, която да реализира достатъчно бързо и достатъчно адекватно поставените задачи за хармонизиране на мелодиите генерирани по случаен принцип.

Въпреки че настоящата разработка дори не се доближава до идеите за напълно действащ изкуствен интелект може да бъде изключително полезен инструмент в ръцете на опитните композитори и музикалните студия.

8.3 Концептуално решение на проблема

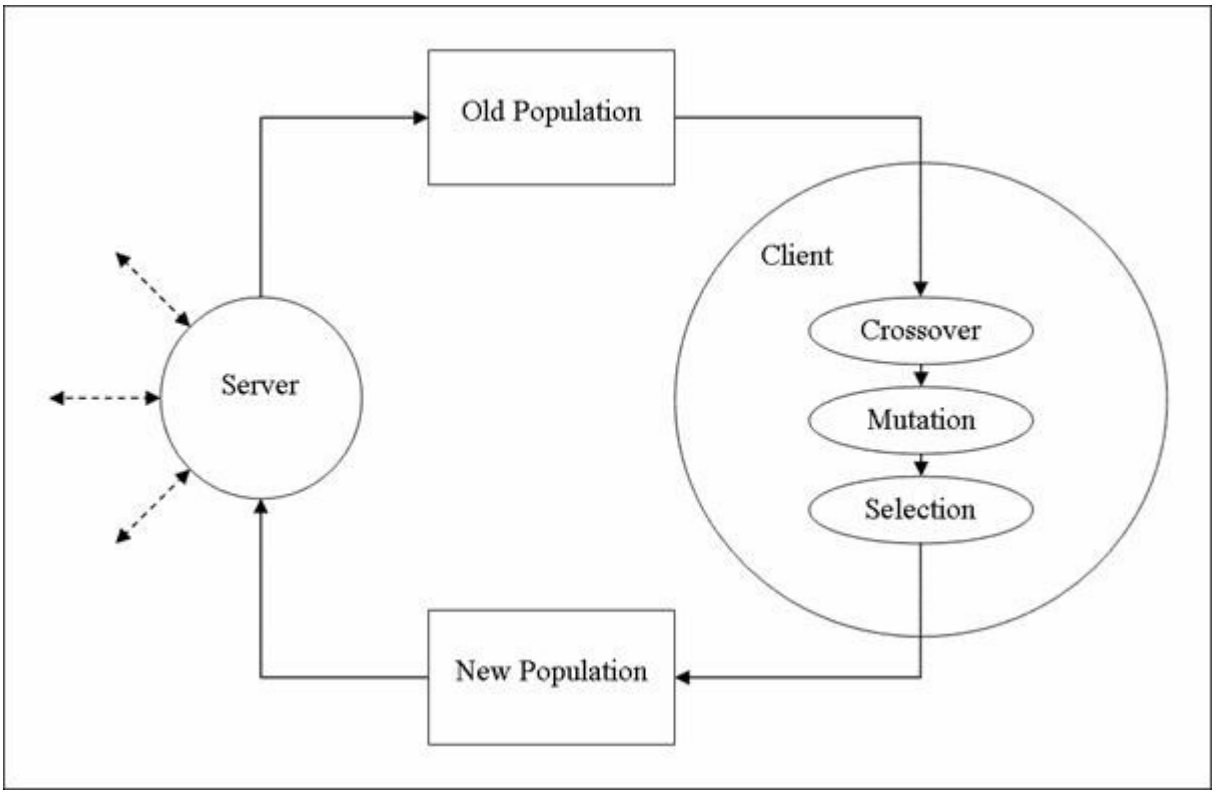
За решаването на поставения проблем е избрана архитектура от тип “клиент-сървър”. От страна на сървъра е разположен набор от мелодии (пул с мелодии). На случаен принцип се избират подмножества от мелодии, които биват изпращани на клиентската страна. За комуникация между сървъра и клиента е избран комуникационен протокол базиран а Java RMI. По същество сървърът и клиентът си разменят отдалечени Java обекти.



Фиг. 8.1 Архитектура “клиент-сървър”.

Подмножеството мелодии, получени от страна на клиента, се рекомбинират с помощта на оптимизационния алгоритъм за диференциална еволюция. За да бъде възможна рекомбинацията мелодиите са едноканални, изпълняват се само от един инструмент и се просвирва само една нота в определен период от време. Тези ограничения за наложени от гледна точка опростяване на диференциалния вектор, който се изчислява при диференциалната еволюция и служи за мутацията на мелодиите, при получаван на новата популация от старата популация.

Старата популация и новата популация последователно се предоставят за прослушване и оценка от страна на крайния потребител (слушател/оценител). Оценката, получена от страна на потребителите служи за жизнена функция и определя кои мелодии ще влязат като част от новото поколение.



Фиг. 8.2 Стъпки за рекомбинация и синхронизация със сървъра.

В процеса си на работа, клиентското приложение синхронизира своята информация с информацията на сървера, след изтичането на определен брой епохи, които се задават от сървъра при заявка на подмножество от мелодии. След като бъдат изчислени заложения брой епохи, клиентът изпраща резултатите на сърра и прави заявка за ново подмножество.

8.4 Архитектура на системата

Тъй като за разработката е използвана гъвкава методология за създаване на софтуер, то самият процес на работа пропуска фазата на софтуерен дизайн, в класическият смисъл на думата, а залага на малки итерации, в рамките на които се оформят отделните класове, групиране на методи и групиране на пакети. Поради тази причина, не са представени диаграми или структурни схеми на модулите в системата, а е описана структурата получила се в итеративния процес: задача - анализ - проектиране - кодиране - интегриране - тестване.

Стъпките по които е получена описаната архитектура могат да бъдат проследени в системата за контрол на версиите, намиращата се в официалната SourceForge Net страница на проекта.

Системата е реализирана в множество модули под формата на отделни класове. Различните модули са групирани в пакети, както следва:

ru.mail.teodorgig.mididermi.base

В базовия пакет се съдържат класовете, които са в основата на решавания проблем. По своето същество това са клас за представяне на обекта нота, клас за представяне на обекта мелодия (съвкупност от ноти) и клас за представяне на обекта популация (съвкупност от мелодии).

Класът за представяне на ноти съдържа четири основни характеристики с които една нота се описва, според стандарта MIDI за описване характеристиките на нотите. Към класа е добавена и функционалност за генерирането на ноти със случайни параметри.

Класът описващ мелодии по същество съдържа последователност от ноти, под формата на динамична структура от данни, както и свойства за отделните мелодии, според нуждите за описване на музикалния инструмент, който ще се ползва за просвирване в стандартната Java MIDI система, оценката получена от

потребителите за мелодията, уникален номер и жанр (за момента не се използва). Класът има множество вътрешни обслужващи методи, като метод за сортиране на нотите по време на просвирване, метод за генериране на уникални идентификатори и спомагателен метод за преобразуване в 7 битово представяне на байтовете, според стандарта MIDI. Добавени са методи за превръщането на данните за мелодията в последователност от числа (използва се за съхраняване в базата данни) и един от най-важните методи, който преобразува мелодията в байтова последователност, според изискванията на стандарта MIDI. Методът за визуална синхронизация се използва за напасване на просвирваната мелодия, спрямо визуалните ефекти, които се изпълняват. Някои от методите, реализиращи алгоритъма на диференциална еволюция също са описани в този клас, предимно изчисляването на диференциалния вектор и обновяването на мелодия с външно подаден диференциален вектор.

Класът описващ популация основно отговаря за съхраняването на множество от мелодии в динамична структура от данни. В него е реализиран вътрешен клас, който има задачата да извършва визуалното представяне на мелодиите по време на тяхното просвирване и оценка. Точно този клас е отговорен за просвирването на мелодиите и събирането на оценката от страна на слушателите/оценителите. Друга част от алгоритъма за диференциална еволюция е реализиран в методи на този клас, предимно рекомбинацията, оценката и селекцията.

ru.mail.teodorgig.mididermi.client

Пакетът предназначен за клиента съдържа само един модул, реализиращ Java аплет, който осъществява връзка с RMI сървър, управлява процеса на еволюция, оценка и визуализация на мелодиите.

ru.mail.teodorgig.mididermi.common

Пакетът с общи модули съдържа основно класове, които се използват както от страната на сървъра, така и от страната на клиента. Това е класът за RMI интерфейс и класът представляващ задачата за отдалечено пресмятане.

Тъй като е реализиран принцип на разпределени изчисления от тип в който клиента се свързва към сървъра, иска задача за изпълнение и след това връща изчислените резултати, то RMI интерфейсът предлага само два метода за отдалечено извикване, които са заявяване на задача и отчитане на резултата.

Класът описващ задачата за пресмятане съдържа една популация и брой епохи за които да бъде извършвана еволюцията. Графичния контекст е част от този клас и се получава от страната на клиента, след като задачата е била получена от страната на сървъра. Също така механизмът за оценка на мелодиите преминава през методи на този клас.

ru.mail.teodorgig.mididermi.database

Пакетът предназначен за връзка с базата данни съдържа само един модул, който реализира посреднически функции, между модулите на приложението и функционалността предоставена от драйвера на избраната база от данни. Класът в този пакет е създаден с цел разделяне на програмната логика за приложението и програмната логика нужна за управлението на базата от данни. По този начин, при бъдещи промени в използваното СУБД с лекота може да се модифицира само този клас, без това по никакъв начин да наруши работата на останалите модули в системата.

Параметрите за връзка към базата данни се четат от стандартен Java файл за описване на свойства. Двата основни метода, които предлага този посреднически клас е изчитане на всички мелодии, записани в базата данни, и записване на всички налични мелодии, от приложението в базата данни. В самата база данни са взети

мерки да не съществуват повтарящи се мелодии с един и същи уникален, за рамките на приложението, идентификатор.

ru.mail.teodorgig.mididermi.providers

Пакетът с доставчиците реализира модули, които доставят различни множества обекти. Основно това са доставчици на мелодии от базата данни, от текстово описани файлове с мелодии, фрактално генерирани мелодии и случайно генерирани мелодии. Част от доставчиците представляват доставчици на една мелодия, а другата част от доставчиците доставят множество от мелодии, използвайки функционалността на единичните доставчици. При четенето на мелодии от текстови файлове се извършва анализ на структурата и поради тази причина е добавен един клас за възникване на изключителни ситуации.

ru.mail.teodorgig.mididermi.server

Пакетът предназначен за сървъра реализира функционалността на конзолен RMI сървър. Един от класовете се стартира в конзолата и с помощта на множество параметри настройва една инстанция, която по същество е имплементация на работната логика, която сървърът е натоварен да изпълнява.

Всички мелодии, които се намират на сървър се съхраняват в пул от мелодии. Когато клиентите поискат задачи за изпълнение от пула се избира случайно подмножество мелодии и се изпращат на клиента. Също така, когато клиентът върне своя отговор точно пулът с мелодии е отговорен да обедини резултатните мелодии с вече съществуващите в пула. Съхраняването и зареждането на данните от страна на сървъра се осъществява точно в този клас.

От страна на сървъра е реализиран и един доставчик на двоични файлове, който има за цел да съхрани в бинарни MIDI файлове всички мелодии налични от страна приложението на сървъра.

8.5 Комуникация в системата

Цялата комуникация в системата се извършва на базата на RMI извиквания и обмен на отдалечени Java обекти. Приложението сървър предоставя два метода за отдалечено извикване, чрез своя RMI интерфейс. Чрез тези методи всеки клиент може да заяви отдалечен RMI обект (обекта реализира задачата за пресмятане), който да му бъде изпратен, да бъде изчислен и след това пак чрез RMI извикване резултатния обект да бъде върнат от страната на сървъра.

Така реализираната комуникация максимално разтоварва изчислителните ресурси на сървъра и прехвърля всички необходими изчисления от страна на клиента. Чрез подходящо настройване на изчислителните епохи интервалите за комуникация между сървъра и клиента могат да се разширят оптимално, което допълнително да разтовари сървъра от множеството синхронизиращи операции, които се налагат при всеки отговор от страна на клиентите.

8.6 Изграждане на приложението от програмния код

За цялостното изграждане на системата от програмния код се използва билд скрипт “build.bat”. Скриптът представлява MS-DOS базиран файл за пакетна обработка (batch file). По настояще процесът на компилация и стартиране е достъпен само в средата на Win32 SP2, но при минимални промени изграждането на системата не би трябвало да бъде проблем под която и да е платформа, поддържаща Java.

За да бъде успешно изграждането на системата трябва в променливата на средата да бъде зададен пътя до bin директорията на Java JDK, така че изпълнимите

файлове javac.exe, java.exe и registry.exe да се извикват безпроблемно от всяка друга текуща директория за операционната система.

За правилното функциониране на системата е нужно да съществува инсталиран PostgreSQL сървър и настройките във файла за свойствата на базата данни (database.properties) да отговарят на реално заложените от страна на PostgreSQL системата за управление на базата от данни.

След процеса на компилация, ако компилацията е преминала без грешки, билд скриптът стартира RMI регистъра, регистрира един екземпляр от RMI сървъра и след натискане на произволен клавиш стартира един RMI клиент. Изчакването за натискане на клавиш се налага от факта, че често клиента се стартира по-бързо, отколкото се стартира RMI регистърът и се прикача RMI сървърът.

8.7 Поддръжка и отстраняване на проблеми

Поддръжката и отстраняването на проблеми в системата изцяло се извършва от автора, но тъй като проектът е с отворен код и не е с комерсиална насоченост не може да се очаква каквото и да било задължение от страна на автора към потребителите на системата.

За контакт с автора:	Todor Dimitrov Balabanov
Email:	tdb@tbsoft-bg.com
ICQ:	17000829
Skype:	teodorgig

Все пак, проектът е достъпен под GPL лиценз за софтуер с отворен код и всеки желаещ самостоятелно може да внесе нужните корекции и поправки, изтегляйки проекта от системата за контрол на версиите, предоставена от официалната страница на проекта в SourceForge Net.

8.8 Инструкции за инсталация

Тъй като проектът е с отворен код, то инсталационния пакет се състои от пълния проект, част от който е и пълния програмен текст. Целият проект официално е публикуван в Subversion хранилището на SourceForge Net, предназначено за целите на проекта.

За целите на инсталацията първо трябва да се изградят приложенията съставящи системата (приложение сървър и приложение клиент). Компютърната система трябва да е снабдена с актуална версия на Java виртуалната машина, както и актуална версия на JDK. От страна на клиента трябва да са налични всички class файлове съставящи клиентското приложение, както и всички общи за клиента и сървъра class файлове. От страната на сървъра са необходими всички class файлове, съставящи приложението сървър, както и общите class файлове.

Тъй като по своята същност системата е уеб базирана, то е необходимо всички спомагателни файлове да се намират в правилните файлови директории и да са с правилните имена, това включва: client.html, database.properties и wideopen.policy.

За да работи системата при пълната си функционалност трябва да е инсталиран PostgreSQL сървър от страна на машината сървър и да бъде изградена базата данни проектирана специално за нуждите на проекта (ползва се SQL скрипт, част от проекта).

8.9 Инструкции за употреба

Всички стъпки за стартиране на системата са изписани под формата на точни команди като част от скритпът за изграждане на системата.

8.9.1 Стартиране на сървъра

От страна на сървъра първоначално се стартира PostgreSQL сървър приложение. На втора стъпка се стартира RMI регистърът. На последната трета стъпка се стартира приложението сървър.

8.9.1 Стартиране на клиента

Тъй като клиента по същество представлява Java аплет, то неговото стартиране в експериментален режим става от командния ред в приложението Applet Viewer, част от стандартния пакет за разработка JDK.

За да бъде стартиран клиентът в стандартен Java съвместим браузър е необходимо клиентското приложение да бъде оформено в JAR архив, който да бъде цифрово подписан. Поради експерименталния характер на системата тази възможност все още не е реализирана, но е основно подобрение при бъдещо доработване.

8.10 Параметри за стартиране на сървъра от командния ред

Приложението сървър се изпълнява в конзолен режим, поради своята относителна простота. Въпреки това, сървърът има набор от параметри с които се управлява поведението му:

-MINPOOL n

Цяло положително число, което указва минимален размер на пула с мелодии от страна на сървъра.

-MAXPOOL n

Цяло положително число, което указва максимален размер на пула с мелодии от страна на сървъра.

-MINEPOCHS n

Цяло положително число, което указва минимален брой еволюционни епохи, които да бъдат изпълнени от страна на клиента, преди осъществяване на синхронизация с данните на сървъра.

-MAXEPOCHS n

Цяло положително число, което указва максимален брой еволюционни епохи, които да бъдат изпълнени от страна на клиента, преди осъществяване на синхронизация с данните на сървъра.

-LR n

Цяло положително число, което указва колко на брой случайно генерирани мелодии да бъдат генерирани в пула, от страна на сървъра.

-LT n

Цяло положително число, което указва колко на брой фрактално генерирани мелодии да бъдат генерирани в пула, от страна на сървъра.

-LD

Параметър, който указва дали да бъдат заредени всички мелодии от базата данни в общия пул, от страна на сървъра.

-LF

Параметър, който указва дали да бъдат заредени всички мелодии от текстови файлове в общия пул, от страна на сървъра.

-SD

Параметър, който указва дали да бъдат съхранени всички мелодии в базата данни от общия пул, от страна на сървъра.

-SF

Параметър, който указва дали да бъдат съхранени всички мелодии в бинарни MIDI файлове от общия пул, от страна на сървъра.

8.11 Управление на проекта

Цялото управление на проекта се извършва с помощта на публичната система SourceForge Net. Системата разполага с модули за форум, система за проследяване на задачи/дефекти, система за контрол на версиите, система за изграждане на уебстраница и множество други подсистеми. Платформата на SourceForge Net се оказва изключително полезна за реализиране принципа на единоначалието при управлението на проекти.

Проектът се администрира от автора на разработката, а в процеса на работа могат да се включат желаещи програмисти от цял свят. Цялата документация (текстови описания и коментари в кода) е написана на английски език.

9 Експериментални данни и резултати

Тъй като целта на разработката не е постигане на резултати под формата на автоматизирано композирани мелодии, а е постигането на работещ прототип за система, която в бъдеще би могла да се използва за научни изследвания в областта на автоматизираното компютърно музициране, то основните експерименти са проведени над функционалността на системата.

В множество тестове е проверена функционалността на системата, а наличието на пълния програмен текст, под формата на проект с отворен код, позволява възможности за допълнително тестване от широк кръг потребители.

При проведените тестове се наблюдава процесът на еволюция през който преминават мелодиите. Също така се наблюдава възможността за оценка на мелодиите, както и визуалното представяне на музикалната информация, която аудио системата изпълнява.

Тъй като разработеният прототип не е напълно завършен, то голяма част от функционалността в системата трябва да бъде доразработена с експерти по музициране, който да проведат и серия от тестове с множество слушатели/оценители. Подобно проучване може да бъде обект на бъдещи изследвания и разработки.

10 Изводи, заключения и препоръки за бъдеща разработка

Разработената система представлява уникален софтуерен продукт, който има за цел да представи новаторски подход за автоматизирано композиране на музикални мелодии. Въпреки безспорните предимства, които системата предлага, финансова изгода може да бъде получена единствено от мелодиите, които приложението може да предложи на професионални композитори изпаднали в творчески застой.

Системата е така разработена, че да позволи на слушателите максимално да се концентрират върху прослушването на генерираните мелодии, които по същество представляват рекомбинация на вече съществуващите мелодии в системата. Тъй като системата поставя слушателите в условия, в които всеки допринася със своята субективна оценка, то води до обобщен и консолидиран вот за мелодиите, които се предлагат.

В съвременните условия на динамично развиващи се пазари иновациите са основен двигател на човешкия прогрес, а от там и на успешния бизнес. Разработената система дава възможност да се провокира креативното мислене у множество професионални композитори, чиито идеи в последствие биха намерили приложение при комерсиалното създаване на музикални продукти с голяма слушаемост, като същевременно води до позициониране в предния ешелон на творческия прогрес.

Композирането на мелодии с помощта на съвременни средства, каквото е представеното в това изложение, позволява значително подобрение и ускорение на творческият процес както при млади композитори, така и при доказани композитори с дългогодишен опит. Като страничен ефект от такива композиции може да се добави неизбежният положителен PR, който носят творбите на изкуството, създадени по иновативен и непознат до момента, за широката публика, подход.

Не на последно място, системата позволява да се осъществят множество обстойни проучвания на потребителските нагласи и индиректно достигане до хармонията, която най-добре би се възприела в конкретния момент от голямо множество слушатели. В бъдещото развитие на предложения софтуер се предвижда визуално оформление на музикалните ефекти (по аналогия с Microsoft Media Player), както и добавяне на графичен потребителски интерфейс, който значително ще улесни крайния потребител.

Разработената система може да бъде адаптирана за множество музикални програми, а на определен етап да се превърне в стандартизирана платформа за композиране на мелодии в Глобалната мрежа, като предостави достъп до мащабна банка с оригинално създадени мелодии, неподлежащи на атаки за нарушени авторски права.

11 Информационни източници

[1] C. Fox, MusicGenie, http://sourceforge.net/projects/musicgenie

[2] Wikipedia, Java remote method invocation, http://en.wikipedia.org/wiki/Java_remote_method_invocation

[3] D. Reilly, Introduction to Java RMI, http://www.javacoffeebreak.com/articles/javarmi/javarmi.html

[4] MIDI File Format, http://www.piclist.com/techref/io/serial/midi/midifile.html

[5] Wikipedia, Differential evolution, http://en.wikipedia.org/wiki/Differential_evolution

[6] K. Price, R. Storn, Differential Evolution (DE), http://www.icsi.berkeley.edu/~storn/code.html

[7] Wikipedia, PostgreSQL, http://en.wikipedia.org/wiki/PostgreSQL

[8] PostgreSQL Global Development Group, About, http://www.postgresql.org/about/

[9] M. Glowiak, Mysql vs postgres, http://monstera.man.poznan.pl/wiki/index.php/Mysql_vs_postgres

[10] Sun Microsystems, Inc, Code Samples and Apps - Applets, http://java.sun.com/applets/

[11] Sun Microsystems, Inc, Java Security Architecture, http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc1.html

[12] I. Kerr, Open Source, CIPPIC Summer Fellow 2004,http://www.cippic.ca/open-source/

[13] J. Schoenberger, Musical Composition with Genetic Algorithms With Coherency Through Genotype, http://www.davidschoenberger.net/joy/career/research.html

[14] S. Draves, Electric Sheep, http://electricsheep.org/

Приложение А - Компакт диск

Компакт дискът, който е неразделна част от този документ, съдържа пълния програмен текст на разработката, презентация, както и всички необходими софтуерни модули за инсталирането на системата под операционна система WinXP32 SP2.

Приложение Б - Структура на базата данни

```
CREATE DATABASE mididermi WITH TEMPLATE = template0 ENCODING = 'UTF8';
ALTER DATABASE mididermi OWNER TO postgres;
COMMENT ON DATABASE mididermi IS 'MIDIDERMI database storage.';

SET client_encoding = 'UTF8';
SET standard_conforming_strings = off;
SET check_function_bodies = false;
SET client_min_messages = warning;
SET escape_string_warning = off;
SET search_path = public, pg_catalog;
SET default_tablespace = '';
SET default_with_oids = false;

CREATE PROCEDURAL LANGUAGE plpgsql;
ALTER PROCEDURAL LANGUAGE plpgsql OWNER TO postgres;

CREATE SEQUENCE melodies_id_seq
    INCREMENT BY 1
    NO MAXVALUE
    NO MINVALUE
    CACHE 1;
ALTER TABLE melodies_id_seq OWNER TO postgres;
COMMENT ON SEQUENCE melodies_id_seq IS 'Provide melodies table primary key unique value.';

CREATE TABLE melodies (
    id integer NOT NULL,
    system_id bigint,
    sequence character varying,
    timber integer,
    score integer,
    genre integer
);
ALTER TABLE melodies OWNER TO postgres;
ALTER TABLE ONLY melodies ADD CONSTRAINT standard PRIMARY KEY (id);

COMMENT ON TABLE melodies IS 'Melodies acting on the server side melody pool.';
COMMENT ON COLUMN melodies.id IS 'Unique identifier.';
COMMENT ON COLUMN melodies.system_id IS 'Unique identifier used into application system.';
COMMENT ON COLUMN melodies.sequence IS 'Sequence of notes, integer numbers separated by space
symbol, grouped by four for each note property (note index, offset from begging, duration of the
note, velocity of the note). ';
COMMENT ON COLUMN melodies.timber IS 'Timber index (instrument index into loaded bank of
instruments).';
COMMENT ON COLUMN melodies.score IS 'Melody score obtained by the end users.';
COMMENT ON COLUMN melodies.genre IS 'Genre index provided by predefined list of genres.';
```

```
CREATE FUNCTION add_melody(system_id bigint, sequence character varying, timber integer, score
integer, genre integer) RETURNS void
AS $_$DECLARE unique_id INTEGER;
BEGIN
        DELETE FROM melodies WHERE melodies.system_id = $1;
        unique_id = nextval('melodies_id_seq');
        INSERT INTO melodies VALUES (unique_id, $1, $2, $3, $4, $5);
END$_$ LANGUAGE plpgsql;
ALTER FUNCTION add_melody(system_id bigint, sequence character varying, timber integer, score
integer, genre integer) OWNER TO postgres;
COMMENT ON FUNCTION add_melody(bigint, character varying, integer, integer, integer) IS 'Add melody
or replace existing melody.';

CREATE FUNCTION delete_melodies() RETURNS void
AS $$BEGIN
        DELETE FROM melodies;
END$$ LANGUAGE plpgsql;
ALTER FUNCTION delete_melodies() OWNER TO postgres;
COMMENT ON FUNCTION delete_melodies() IS 'Delete all melodies.';

CREATE FUNCTION get_all_melodies() RETURNS SETOF melodies
AS $$BEGIN
        RETURN QUERY SELECT * FROM melodies;
END$$ LANGUAGE plpgsql;
ALTER FUNCTION get_all_melodies() OWNER TO postgres;
COMMENT ON FUNCTION get_all_melodies() IS 'Return all available melodies.';
```

Приложение В - Програмен код

В настоящото приложение е представен целия програмен текст на разработената система, това включва проектните файлове за изграждане на приложенията, конфигурационните файлове за настройки на системата и целият програмен код. Файловете за в азбучна последователност и са групирани в азбучната последователност на директориите в които се намират (главните директории предхождат поддиректориите).

build.bat

```
@echo off

cls

set classpath=

javac.exe -d ./bin ./src/ru/mail/teodorgig/mididermi/base/*.java -classpath ./bin
javac.exe -d ./bin ./src/ru/mail/teodorgig/mididermi/common/*.java -classpath ./bin
javac.exe -d ./bin ./src/ru/mail/teodorgig/mididermi/database/*.java -classpath ./bin
javac.exe -d ./bin ./src/ru/mail/teodorgig/mididermi/providers/*.java -classpath ./bin
javac.exe -d ./bin ./src/ru/mail/teodorgig/mididermi/server/*.java -classpath ./bin
javac.exe -d ./bin ./src/ru/mail/teodorgig/mididermi/client/*.java -classpath ./bin
```

```
copy .\src\client.html .\bin\client.html
copy .\src\database.properties .\bin\database.properties

cd ./bin

start rmiregistry.exe

echo RMI registry started ...

start java.exe -cp .;../lib/postgresql-8.3-603.jdbc4.jar -Djava.rmi.server.codebase=file:/C:/TEMP/
-Djava.rmi.server.hostname=127.0.0.1 -Djava.security.policy=../wideopen.policy
ru.mail.teodorgig.mididermi.server.MIDIDERMIServer -MINPOOL 2 -MAXPOOL 30 -MINEPOCHS 1 -MAXEPOCHS 5
-LR 1 -LT 1 -LD -LF -SD -SF

echo RMI server application registered ...

pause

start appletviewer.exe -J-Djava.rmi.server.codebase=http://127.0.0.1/ -J-
Djava.security.policy=../wideopen.policy client.html

echo RMI clinet application(s) started ...

pause

cd ./ru/mail/teodorgig/mididermi/client
del *.class
cd..
rd client

cd server
del *.class
cd..
rd server

cd common
del *.class
cd..
rd common

cd providers
del *.class
cd..
rd providers

cd database
del *.class
cd..
rd database

cd base
del *.class
cd..
rd base

cd ..
rd mididermi
cd ..
rd teodorgig
cd ..
rd mail
cd ..
rd ru
del client.html
del database.properties
cd ..

echo Binary directory clean ...

@echo on
```

wideopen.policy

```
grant {
       // Allow everything for now
       permission java.security.AllPermission;
};
```

client.html

```
<html>
  <head>
    <title>MIDIDERMI client ...</title>
  </head>

  <body>
    <applet code="ru.mail.teodorgig.mididermi.client.MIDIDERMIClient" width="640" height="480">
      <param name="rmi-server-address" value="127.0.0.1">
    </applet>
  </body>
</html>
```

database.properties

```
# Database connection host.
host = localhost

# Database connection port.
port = 5432

# Database username to be used for connection.
username = postgres

# Password of user to be connected.
password =

# Working database name.
database = mididermi
```

Melody.java

```
/*****************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                  *
 * New Bulgarian University                                                  *
 *                                                                           *
 * Copyright (c) 2007, 2008 Todor Balabanov                                  *
 *                                                                           *
 * http://tdb.hit.bg/                                                        *
 *                                                                           *
 * This program is free software; you can redistribute it and/or modify      *
 * it under the terms of the GNU General Public License as published by      *
```

```
 * the Free Software Foundation; either version 2 of the License, or           *
 * (at your option) any later version.                                         *
 *                                                                             *
 * This program is distributed in the hope that it will be useful,             *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of              *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the               *
 * GNU General Public License for more details.                                *
 *                                                                             *
 * You should have received a copy of the GNU General Public License along     *
 * with this program; if not, write to the Free Software Foundation, Inc.,     *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.                 *
 *******************************************************************************/

package ru.mail.teodorgig.mididermi.base;

import java.util.Vector;
import java.io.Serializable;

/**
 * Melody is a sequence of music notes.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public class Melody implements Cloneable, Serializable {
	/**
	 * Default serial version uid.
	 */
	private static final long serialVersionUID = 1L;

	/**
	 * Max time used during melody construction.
	 */
	private static final long MAX_TIME_MIDI_CONSTRUCTION = 100000L;

	/**
	 * Sequence of music notes.
	 */
	private Vector<Note> sequence;

	/**
	 * Melody timber or number of patch playing.
	 */
	private int timber;

	/**
	 * Score given by the users for this melody.
	 */
	private int score;

	/**
	 * Identifier for the unique melodies.
	 */
	private long id;

	/**
	 * Genre as number from predefined constants.
	 */
	private int genre;

	/**
	 * Convert long number to MIDI variable length array. Supporting method for
	 * data type conversion needed by MIDI specification of data representation.
	 *
	 * @param num
	 *            Regular long number.
	 *
	 * @return Array with MIDI var length values.
```

```
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
private char[] long2var(long num) {
        char res[] = new char[4];

        num = ((num & 0xffffff80) << 1) | num & 0x0000007f;
        num = ((num & 0xffff8000) << 1) | num & 0x00007fff;
        num = ((num & 0xff800000) << 1) | num & 0x007fffff;
        num = ((num & 0x80000000) << 1) | num & 0x7fffffff;

        num &= 0xffffff7f;
        num |= 0x00008000;
        num |= 0x00800000;
        num |= 0x80000000;

        /*
         * 4 bytes mask for MIDI variable length value.
         */
        res[0] = (char) ((num & 0x7f000000) >> 48);
        res[1] = (char) ((num & 0x007f0000) >> 32);
        res[2] = (char) ((num & 0x00007f00) >> 16);
        res[3] = (char) ((num & 0x0000007f) >> 0);
        res[0] |= 0x80;
        res[1] |= 0x80;
        res[2] |= 0x80;
        res[3] &= 0x7f;

        return (res);
}

/**
 * Provide unique identifier based on the time stamp and random number.
 *
 * @return Unique identifier.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 30 May 2008
 */
static public long getUniqueId() {
        return ((System.currentTimeMillis() << 3 | (long) (Math.random() * 8)));
}

/**
 * Sorting of the music notes into the melody according time offset from the
 * beginning of the melody. Sorting is needed because some melodies can be
 * disordered like notes starting in not proper moments.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public void sort() {
        Note a, b;
        boolean done = false;

        while (done == false) {
                done = true;

                for (int i = 0; i < sequence.size() - 1; i++) {
                        a = sequence.elementAt(i);
                        b = sequence.elementAt(i + 1);
```

```
                        if (a.getOffset() > b.getOffset()) {
                                sequence.removeElementAt(i);
                                sequence.insertElementAt(b, i);
                                sequence.removeElementAt(i + 1);
                                sequence.insertElementAt(a, i + 1);
                                done = false;
                        }
                }
        }
}

/**
 * Constructor without parameters. Constructor is needed to create internal
 * data structures of the object.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public Melody() {
        super();

        sequence = new Vector<Note>();
        timber = 1;
        score = 0;
}

/**
 * Melody length.
 *
 * @return Length of the melody in number of notes.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 27 May 2008
 */
int length() {
        return (sequence.size());
}

/**
 * Melody timber getter.
 *
 * @return Melody timber.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 27 May 2008
 */
public int getTimber() {
        return (timber);
}

/**
 * Melody timber setter.
 *
 * @param timber
 *            Melody timber.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 27 May 2008
```

```
 */
public void setTimber(int timber) {
	if (timber < 1) {
		timber = 1;
	}

	if (timber > 128) {
		timber = 128;
	}

	this.timber = timber;
}

/**
 * Melody score getter.
 *
 * @return Melody score.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public int getScore() {
	return (score);
}

/**
 * Melody score setter.
 *
 * @param score
 *            Melody score.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public void setScore(int score) {
	this.score = score;
}

/**
 * Melody score plus one.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 26 May 2008
 */
public void scoreUp() {
	score++;
}

/**
 * Melody score minus one.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 26 May 2008
 */
public void scoreDown() {
	score--;
}

/**
```

```
 * Melody identifier getter.
 *
 * @return Melody identifier.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public long getId() {
	return (id);
}

/**
 * Melody identifier setter.
 *
 * @param id
 *            Melody identifier.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public void setId(long id) {
	this.id = id;
}

/**
 * Melody genre getter.
 *
 * @return Numeric index of genre.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 03 Jun 2008
 */
public int getGenre() {
	return (genre);
}

/**
 * Melody genre setter.
 *
 * @param genre
 *            Numeric index of genre.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 03 Jun 2008
 */
public void setGenre(int genre) {
	this.genre = genre;
}

/**
 * Provide string representation of melody notes as group of numbers. This
 * representation is very useful for database melody storing. All numbers
 * are separated by space symbol. Amount of numbers in a group depends of
 * note properties available.
 *
 * @return String representation of the melody notes as group of numbers.
 *
 * @author Todor Balabanov
 *
```

```
 * @email teodorgig@mail.ru
 *
 * @date 27 Jun 2008
 */
public String getNotesInNumbers() {
	String result = "";
	Note note = null;

	for (int i = 0; i < sequence.size(); i++) {
		note = (Note) sequence.elementAt(i).clone();

		result += note.getNote();
		result += " ";
		result += note.getOffset();
		result += " ";
		result += note.getDuration();
		result += " ";
		result += note.getVelocity();
		result += " ";
	}

	result = result.trim();

	return (result);
}

/**
 * Add music note to the sequence. Sequence of notes can be extended by
 * simply adding new note. This process may need notes sort ordering.
 *
 * @param note
 *            Music note.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public void addNote(Note note) {
	sequence.add(note);
}

/**
 * Get note on specific moment in time.
 *
 * @param position
 *            Percent of moment in time between 0 and 1.
 *
 * @return Note in this special moment or null.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 15 Jun 2008
 */
public Note getNoteOn(double position) {
	Note note = null;
	Note result = null;

	double length = 0.0;
	for (int i = 0; i < sequence.size(); i++) {
		note = (Note) sequence.elementAt(i).clone();

		if (note.getOffset() + note.getDuration() > length) {
			length = note.getOffset() + note.getDuration();
		}
	}

	if (length > 0.0) {
```

```
                        double distance = 1.0;
                        for (int i = 0; i < sequence.size(); i++) {
                                note = (Note) sequence.elementAt(i).clone();

                                if (Math.abs(note.getOffset() / length - position) < distance) {
                                        distance = Math.abs(note.getOffset() / length - position);
                                        result = note;
                                }

                                if (Math.abs((note.getOffset() + note.getDuration()) / length -
position) < distance) {
                                        distance = Math.abs(note.getOffset() / length - position);
                                        result = note;
                                }
                        }
                }

                return (result);
        }

        /**
         * Calculates melody differential. Difference vector is needed during
         * evolution process and is done by simply calculate difference of all
         * parameters in each music note.
         *
         * @param val
         *            Subtractor.
         *
         * @return Differential.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 22 May 2008
         */
        public int[][] getDiffertial(Melody val) {
                int res[][] = new int[sequence.size()][4];
                Note a, b;

                for (int i = 0; i < sequence.size(); i++) {
                        a = sequence.elementAt(i);
                        b = val.sequence.elementAt(i % val.sequence.size());

                        res[i][0] = a.getNote() - b.getNote();
                        if (a.getOffset() == b.getOffset())
                                res[i][1] = 0;
                        else if (a.getOffset() < b.getOffset())
                                res[i][1] = -1;
                        else if (a.getOffset() > b.getOffset())
                                res[i][1] = +1;
                        res[i][2] = a.getDuration() - b.getDuration();
                        res[i][3] = a.getVelocity() - b.getVelocity();
                }

                return (res);
        }

        /**
         * Update melody according differential vector. It is needed during
         * evolution process and difference vector is added note by note in each
         * properties of the note. By this way new generation is created.
         *
         * @param differential
         *            Differential vector.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 22 May 2008
```

```
         */
        public void update(int differential[][]) {
                Note note = null;

                for (int i = 0; i < sequence.size(); i++) {
                        note = sequence.elementAt(i);
                        note.setNote(note.getNote() + differential[i % differential.length][0]);
                        note.setOffset(note.getOffset() + differential[i % differential.length][1]);
                        note.setDuration(note.getDuration() + differential[i % differential.length]
[2]);
                        note.setVelocity(note.getVelocity() + differential[i % differential.length]
[3]);
                        note = null;
                }
        }

        /**
         * Convert the music melody into MIDI byte file sequence. It is needed
         * because of the internal data representation and the possibilities Java
         * API to play MIDI stream.
         *
         * http://www.sonicspot.com/guide/midifiles.html
         *
         * http://jedi.ks.uiuc.edu/~johns/links/music/midifile.html
         *
         * @return Byte sequence.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 22 May 2008
         */
        public char[] toMidiBytes() {
                int size = 0;

                /*
                 * MIDI header size.
                 */
                size += 14;

                /*
                 * Track header size.
                 */
                size += 8;

                /*
                 * Copyrights size.
                 */
                size += 24;

                /*
                 * Select instrument.
                 */
                size += 11;

                /*
                 * Start and end note by number of notes by variable start time plus
                 * event bytes.
                 */
                size += 2 * sequence.size() * (4 + 3);

                /*
                 * Track end event.
                 */
                size += 4;

                char res[] = new char[size];

                for (int i = 0; i < res.length; i++)
                        res[i] = 0;
```

```
/*
 * MIDI header.
 */
res[0] = 'M';
res[1] = 'T';
res[2] = 'h';
res[3] = 'd';

/*
 * MIDI header size.
 */
res[4] = 0;
res[5] = 0;
res[6] = 0;
res[7] = 6;

/*
 * MIDI file type.
 */
res[8] = 0;
res[9] = 0;

/*
 * MIDI number of tracks.
 */
res[10] = 0;
res[11] = 1;

/*
 * MIDI ticks per beat.
 */
res[12] = 0;
res[13] = 120;

/*
 * Track header.
 */
res[14] = 'M';
res[15] = 'T';
res[16] = 'r';
res[17] = 'k';

/*
 * Track size.
 */
res[18] = 0;
res[19] = 0;
res[20] = 0;
res[21] = 0;

/*
 * Copyrights meta event.
 */
res[22] = 0x00;
res[23] = 0xff;
res[24] = 0x02;
res[25] = 0x80;
res[26] = 0x80;
res[27] = 0x80;
res[28] = 0x11;
res[29] = 'T';
res[30] = 'o';
res[31] = 'd';
res[32] = 'o';
res[33] = 'r';
res[34] = ' ';
res[35] = 'B';
res[36] = 'a';
res[37] = 'l';
res[38] = 'a';
```

```
res[39] = 'b';
res[40] = 'a';
res[41] = 'n';
res[42] = 'o';
res[43] = 'v';
res[44] = ' ';
res[45] = 0xa9;

/*
 * Bank select MSB=0, time 0, controller on channel 1.
 */
res[46] = 0x00;
res[47] = 0xb0;
res[48] = 0x00;
res[49] = 0x00;

/*
 * Bank select LSB=3, time 0, controller on channel 1.
 */
res[50] = 0x00;
res[51] = 0xb0;
res[52] = 0x20;
res[53] = 0x03;

/*
 * Program change on channel 1, time 0, patches from 1 to 128.
 */
res[54] = 0x00;
res[55] = 0xc0;
res[56] = (char) timber;

/*
 * Music events.
 */
int pos = 57;
for (long time = 0, last = 0; time < MAX_TIME_MIDI_CONSTRUCTION; time++) {
        for (int i = 0; i < sequence.size(); i++) {
                Note note = sequence.elementAt(i);
                /*
                 * Note start on channel.
                 */
                if (time == note.getOffset()) {
                        char[] offset = long2var(time - last);
                        res[pos++] = offset[0];
                        res[pos++] = offset[1];
                        res[pos++] = offset[2];
                        res[pos++] = offset[3];
                        res[pos++] = 0x90 | 0x00;
                        res[pos++] = (char) (note.getNote() & 0x7f);
                        res[pos++] = (char) (note.getVelocity() & 0x7f);
                        last = time;
                }
                /*
                 * Note end on channel.
                 */
                if (time == (note.getOffset() + note.getDuration())) {
                        char[] offset = long2var(time - last);
                        res[pos++] = offset[0];
                        res[pos++] = offset[1];
                        res[pos++] = offset[2];
                        res[pos++] = offset[3];
                        res[pos++] = 0x80 | 0x00;
                        res[pos++] = (char) (note.getNote() & 0x7f);
                        res[pos++] = (char) (note.getVelocity() & 0x7f);
                        last = time;
                }
        }
}

/*
 * The end of the track.
```

```
                 */
                res[pos++] = 0x00;
                res[pos++] = 0xff;
                res[pos++] = 0x2f;
                res[pos++] = 0x00;

                /*
                 * Track size. Subtract size of the header from the current position.
                 */
                long truckSize = pos - 22;
                res[18] = (char) ((truckSize & 0xff000000) >> 24);
                res[19] = (char) ((truckSize & 0x00ff0000) >> 16);
                res[20] = (char) ((truckSize & 0x0000ff00) >> 8);
                res[21] = (char) ((truckSize & 0x000000ff) >> 0);

                return (res);
        }

        /**
         * Compare two melodies. Because the environment is distributed it is very
         * common one melody to appear twice or more times. Because of that
         * comparison method is needed.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 22 May 2008
         */
        public boolean equals(Object obj) {
                Melody melody = null;

                try {
                        melody = (Melody) obj;
                } catch (Exception ex) {
                        return (false);
                }

                if (melody == null)
                        return (false);

                if (this.sequence.size() != melody.sequence.size())
                        return (false);

                for (int i = 0; i < this.sequence.size(); i++)
                        if ((this.sequence.elementAt(i)).equals(melody.sequence.elementAt(i)) ==
false)
                                return (false);

                return (true);
        }

        /**
         * Clone the melody object. The most clean way to make identical copy of
         * complex data structure.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 22 May 2008
         */
        public Object clone() {
                Melody melody = new Melody();

                melody.sequence = new Vector<Note>();

                for (int i = 0; i < sequence.size(); i++)
                        melody.sequence.add((Note) (sequence.elementAt(i)).clone());
                melody.timber = timber;
                melody.score = score;
```

```
                melody.id = id;

                return (melody);
        }

        /**
         * Transform melody properties into string.
         *
         * @return String representation of the melody properties.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 27 May 2008
         */
        public String toString() {
                return ("Melody id [" + getId() + "] Score [" + getScore() + "] Timber [" +
getTimber() + "] Length [" + length() + "]");
        }
}
```

Note.java

```
/*******************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                    *
 * New Bulgarian University                                                    *
 *                                                                             *
 * Copyright (c) 2007, 2008 Todor Balabanov                                    *
 *                                                                             *
 * http://tdb.hit.bg/                                                          *
 *                                                                             *
 * This program is free software; you can redistribute it and/or modify        *
 * it under the terms of the GNU General Public License as published by        *
 * the Free Software Foundation; either version 2 of the License, or           *
 * (at your option) any later version.                                         *
 *                                                                             *
 * This program is distributed in the hope that it will be useful,             *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of              *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the               *
 * GNU General Public License for more details.                                *
 *                                                                             *
 * You should have received a copy of the GNU General Public License along     *
 * with this program; if not, write to the Free Software Foundation, Inc.,     *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.                 *
 *******************************************************************************/

package ru.mail.teodorgig.mididermi.base;

import java.io.Serializable;

/**
 * Note represents single music note. Notes are used as objects in melodies.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public class Note implements Cloneable, Serializable {
        /**
         * Default serial version uid.
         */
        private static final long serialVersionUID = 1L;
```

```
/**
 * Min note number for random generation.
 */
private static final int MIN_RANDOM_NOTE = 0;

/**
 * Max note number for random generation.
 */
private static final int MAX_RANDOM_NOTE = 127;

/**
 * Min note offset for random generation.
 */
private static final int MIN_RANDOM_OFFSET = 1;

/**
 * Max note offset for random generation.
 */
private static final int MAX_RANDOM_OFFSET = 10000;

/**
 * Min note duration for random generation.
 */
private static final int MIN_RANDOM_DURATION = 1;

/**
 * Max note duration for random generation.
 */
private static final int MAX_RANDOM_DURATION = 100;

/**
 * Min note velocity for random generation.
 */
private static final int MIN_RANDOM_VELOCITY = 0;

/**
 * Max note velocity for random generation.
 */
private static final int MAX_RANDOM_VELOCITY = 127;

/**
 * Musical note number according MIDI standard.
 */
private int note;

/**
 * Musical note offset from the beginning of the melody.
 */
private int offset;

/**
 * Musical note duration.
 */
private int duration;

/**
 * Musical note velocity.
 */
private int velocity;

/**
 * Generate music note with random parameters. It is needed during random
 * melody construction.
 *
 * @return Music note.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
```

```
         * @date 22 May 2008
         */
        public static Note provideRandom() {
                int note = MIN_RANDOM_NOTE + (int) (Math.random() * (MAX_RANDOM_NOTE -
MIN_RANDOM_NOTE + 1));

                int offset = MIN_RANDOM_OFFSET + (int) (Math.random() * (MAX_RANDOM_OFFSET -
MIN_RANDOM_OFFSET + 1));

                int duration = MIN_RANDOM_DURATION + (int) (Math.random() * (MAX_RANDOM_DURATION -
MIN_RANDOM_DURATION + 1));

                int velocity = MIN_RANDOM_VELOCITY + (int) (Math.random() * (MAX_RANDOM_VELOCITY -
MIN_RANDOM_VELOCITY + 1));

                return (new Note(note, offset, duration, velocity));
        }

        /**
         * Constructor without parameters.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 22 May 2008
         */
        public Note() {
                this(0, 0, 0, 0);
        }

        /**
         * Constructor with parameters.
         *
         * @param note
         *            Note number.
         * @param offset
         *            Start time offset.
         * @param duration
         *            Note duration.
         * @param velocity
         *            Note velocity.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 22 May 2008
         */
        public Note(int note, int offset, int duration, int velocity) {
                super();

                this.setNote(note);
                this.setOffset(offset);
                this.setDuration(duration);
                this.setVelocity(velocity);
        }

        /**
         * Duration getter.
         *
         * @return Note duration.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 22 May 2008
         */
        public int getDuration() {
                return duration;
```

```
}

/**
 * Duration setter.
 *
 * @param duration
 *            Note duration.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public void setDuration(int duration) {
        if (duration < MIN_RANDOM_DURATION)
                duration = MIN_RANDOM_DURATION;

        if (duration > MAX_RANDOM_DURATION)
                duration = MAX_RANDOM_DURATION;

        this.duration = duration;
}

/**
 * Note number getter.
 *
 * @return Note number.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public int getNote() {
        return note;
}

/**
 * Note number setter.
 *
 * @param note
 *            Note number.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public void setNote(int note) {
        if (note < MIN_RANDOM_NOTE)
                note = MIN_RANDOM_NOTE;

        if (note > MAX_RANDOM_NOTE)
                note = MAX_RANDOM_NOTE;

        this.note = note;
}

/**
 * Note start time offset getter.
 *
 * @return Note start time offset.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
```

```
 */
public int getOffset() {
	return offset;
}

/**
 * Note start time offset setter.
 *
 * @param offset
 *            Note start time offset.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public void setOffset(int offset) {
	if (offset < MIN_RANDOM_OFFSET)
		offset = MIN_RANDOM_OFFSET;

	if (offset > MAX_RANDOM_OFFSET)
		offset = MAX_RANDOM_OFFSET;

	this.offset = offset;
}

/**
 * Note velocity getter.
 *
 * @return Note velocity.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public int getVelocity() {
	return velocity;
}

/**
 * Note velocity setter.
 *
 * @param velocity
 *            Note velocity.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public void setVelocity(int velocity) {
	if (velocity < MIN_RANDOM_VELOCITY)
		velocity = MIN_RANDOM_VELOCITY;

	if (velocity > MAX_RANDOM_VELOCITY)
		velocity = MAX_RANDOM_VELOCITY;

	this.velocity = velocity;
}

/**
 * Compare two music notes.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
```

```
         * @date 22 May 2008
         */
        public boolean equals(Object obj) {
                Note note = null;

                try {
                        note = (Note) obj;
                } catch (Exception ex) {
                        return (false);
                }

                if (note == null)
                        return (false);

                if (this.note != note.note || this.offset != note.offset || this.duration !=
note.duration || this.velocity != note.velocity)
                        return (false);

                return (true);
        }

        /**
         * Clone the note object. The most clean way to make identical copy of
         * complex data structure.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 22 May 2008
         */
        public Object clone() {
                return (new Note(note, offset, duration, velocity));
        }
}
```

Population.java

```
/*******************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                    *
 * New Bulgarian University                                                    *
 *                                                                             *
 * Copyright (c) 2007, 2008 Todor Balabanov                                    *
 *                                                                             *
 * http://tdb.hit.bg/                                                          *
 *                                                                             *
 * This program is free software; you can redistribute it and/or modify        *
 * it under the terms of the GNU General Public License as published by        *
 * the Free Software Foundation; either version 2 of the License, or           *
 * (at your option) any later version.                                         *
 *                                                                             *
 * This program is distributed in the hope that it will be useful,             *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of              *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the               *
 * GNU General Public License for more details.                                *
 *                                                                             *
 * You should have received a copy of the GNU General Public License along     *
 * with this program; if not, write to the Free Software Foundation, Inc.,     *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.                 *
 *******************************************************************************/

package ru.mail.teodorgig.mididermi.base;

import java.util.Vector;
import java.awt.Color;
```

```
import java.awt.Graphics;
import java.io.Serializable;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;

import javax.sound.midi.Sequence;
import javax.sound.midi.Sequencer;
import javax.sound.midi.MidiSystem;

/**
 * Differential evolution population class. Implementation of genetic algorithm
 * based optimization approach. All melodies are presented as chromosomes
 * (vectors). During evolution period recombination of chromosomes is applied
 * and human evaluation of the melody quality.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public class Population implements Cloneable, Serializable {
	/**
	 * Painter is checking for playing sequence and do visual effects according
	 * to melody.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 15 Jun 2008
	 */
	private class Painter extends Thread {
		/**
		 * Delay for redraw in milliseconds.
		 */
		private static final int REDRAW_DELAY = 60;

		/**
		 * Default run method of the tread.
		 *
		 * @author Todor Balabanov
		 *
		 * @email teodorgig@mail.ru
		 *
		 * @date 15 Jun 2008
		 */
		public void run() {
			while (true) {
				if (g != null && sequencer != null && sequencer.isRunning() == true) {
					double position = (double) sequencer.getMicrosecondPosition() /
(double) sequencer.getMicrosecondLength();

					Note note = melodyPaying.getNoteOn(position);

					if (note != null) {
						int red = g.getColor().getRed();
						int green = g.getColor().getGreen();
						int blue = g.getColor().getBlue();
						g.setColor(new Color((red + note.getNote()) % 256,
(green + note.getVelocity()) % 256, (blue + note.getDuration()) % 256));
					}

					int x = (int) (g.getClipBounds().x + ((double)
sequencer.getMicrosecondPosition() / (double) sequencer.getMicrosecondLength()) *
g.getClipBounds().width);
					int y = (int) (g.getClipBounds().y + ((double)
sequencer.getMicrosecondLength() / (double) sequencer.getMicrosecondPosition()) + Math.random() *
g.getClipBounds().height) % g.getClipBounds().height;
```

```
                                g.drawLine(x - 2, y, x - 2, y);
                                g.drawLine(x, y - 2, x, y - 2);
                                g.drawLine(x - 1, y, x - 1, y);
                                g.drawLine(x, y - 1, x, y - 1);
                                g.drawLine(x, y, x, y);
                                g.drawLine(x + 1, y, x + 1, y);
                                g.drawLine(x, y + 1, x, y + 1);
                                g.drawLine(x + 2, y, x + 2, y);
                                g.drawLine(x, y + 2, x, y + 2);
                        }

                        try {
                                Thread.sleep(REDRAW_DELAY);
                        } catch (InterruptedException ex) {
                                /*
                                 * It does not matter if thread is interrupted during
                                 * sleeping mode.
                                 */
                        }
                }
        }
}

/**
 * Default serial version uid.
 */
private static final long serialVersionUID = 1L;

/**
 * Time to sleep before next melody in milliseconds.
 */
private static long TIME_BEFORE_NEXT_MELODY = 500;

/**
 * Min population size for random generation.
 */
public static final int MIN_RANDOM_POPULATION = 3;

/**
 * Max population size for random generation.
 */
public static final int MAX_RANDOM_POPULATION = 5;

/**
 * Min population size for fractal generation.
 */
public static final int MIN_FRACTAL_POPULATION = 3;

/**
 * Max population size for fractal generation.
 */
public static final int MAX_FRACTAL_POPULATION = 5;

/**
 * Size of the DE population.
 */
private int size = 0;

/**
 * Chromosomes set of DE.
 */
private Vector<Melody> offspring;

/**
 * Handle to the melody which is playing.
 */
private Melody melodyPaying = null;

/**
 * Sequencer object for melody playing.
```

```
 */
private Sequencer sequencer = null;

/**
 * Graphics context to visualize music information.
 */
private Graphics g = null;

/**
 * Constructor without parameters. Internal structure is created during
 * constructor execution.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public Population() {
        super();

        offspring = new Vector<Melody>();
}

/**
 * Population size getter.
 *
 * @return Population size without new generation.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 30 May 2008
 */
public int getSize() {
        return (size);
}

/**
 * Population size setter.
 *
 * @param size
 *            Population size without new generation.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 30 May 2008
 */
public void setSize(int size) {
        this.size = size;
}

/**
 * All population melodies getter.
 *
 * @return All melodies.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public Vector<Melody> getMelodies() {
        Vector<Melody> melodies = new Vector<Melody>();

        for (int i = 0; i < offspring.size(); i++)
                melodies.add((Melody) (offspring.elementAt(i)).clone());
```

```
		return (melodies);
	}

	/**
	 * Add melody. Population can be extended during evolution process.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 22 May 2008
	 */
	public void add(Melody melody) {
		melody.setId(offspring.size());
		offspring.add(melody);

		size++;
	}

	/**
	 * Recombination of the chromosomes according to DE rules. Choosing
	 * chromosomes and do differential vector addition.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 22 May 2008
	 */
	public void recombine() {
		Melody melody = null;
		int size = offspring.size();

		for (int i = 0; i < size; i++) {
			melody = (Melody) (offspring.elementAt(i)).clone();
			melody.update((offspring.elementAt(i)).getDiffertial(offspring.elementAt((int)
(Math.random() * size))));
			melody.setId(offspring.size());

			offspring.add(melody);
		}
	}

	/**
	 * Determine the fitness function for each chromosome by creating a MIDI
	 * stream and evaluating from the user.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 22 May 2008
	 */
	public void evaluate() {
		for (int j = 0; j < offspring.size(); j++) {
			melodyPaying = offspring.elementAt(j);

			ByteArrayOutputStream bao = new ByteArrayOutputStream();
			DataOutputStream out = new DataOutputStream(bao);
			byte[] bytes = null;
			try {
				char[] chars = melodyPaying.toMidiBytes();
				for (int i = 0; i < chars.length; i++) {
					out.write(chars[i]);
				}
				out.flush();
				bytes = bao.toByteArray();
				bao.close();
			} catch (Exception ex) {
				ex.printStackTrace();
```

```
                }

                System.out.println(melodyPaying);

                ByteArrayInputStream bai = new ByteArrayInputStream(bytes);

                try {
                        sequencer = MidiSystem.getSequencer();
                        Sequence sequence = MidiSystem.getSequence(new DataInputStream(bai));

                        sequencer.open();
                        sequencer.setSequence(sequence);

                        Painter painter = new Painter();
                        painter.start();
                        sequencer.start();

                        Thread.sleep(sequencer.getMicrosecondLength() / 1000);
                        Thread.sleep(TIME_BEFORE_NEXT_MELODY);

                        sequencer.stop();
                        sequencer.close();
                        painter.interrupt();

                        sequencer = null;
                } catch (Exception ex) {
                        ex.printStackTrace();
                }
        }
}

/**
 * Sort the population according to the fitness value. One of the possible
 * ways to select parents for the next generation.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public void sort() {
        Melody a, b;
        boolean done = false;

        while (done == false) {
                done = true;

                for (int i = 0; i < offspring.size() - 1; i++) {
                        a = offspring.elementAt(i);
                        b = offspring.elementAt(i + 1);

                        if (a.getScore() < b.getScore()) {
                                offspring.removeElementAt(i);
                                offspring.insertElementAt(b, i);
                                offspring.removeElementAt(i + 1);
                                offspring.insertElementAt(a, i + 1);
                                done = false;
                        }
                }
        }
}

/**
 * Remove extra population chromosomes. New generation is added as part of
 * the old generation, after selection process only limited amount of
 * chormosomes should survive.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
```

```
 *
 * @date 22 May 2008
 */
public void shrink() {
	for (int i = offspring.size() - 1; i >= size; i--) {
		offspring.removeElementAt(i);
	}
}

/**
 * Population epochs. Calculates given number of epochs as part of
 * population evolution.
 *
 * @param number
 *            How many epochs.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public void epoches(int number) {
	for (int i = 0; i < number; i++) {
		recombine();
		evaluate();
		sort();
		shrink();
	}
}

/**
 * Score up of the currently playing melody. Method is provided as
 * connection between graphic user interface and real melody object.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 26 May 2008
 */
public void scoreUp() {
	melodyPaying.scoreUp();

	System.out.println(melodyPaying);
}

/**
 * Score down of the currently playing melody. Method is provided as
 * connection between graphic user interface and real melody object.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 26 May 2008
 */
public void scoreDown() {
	melodyPaying.scoreDown();

	System.out.println(melodyPaying);
}

/**
 * Clone the population object. The most clean way to make identical copy of
 * complex data structure.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
```

```
         * @date 22 May 2008
         */
        public Object clone() {
                Population population = new Population();

                population.offspring = new Vector<Melody>();

                for (int i = 0; i < offspring.size(); i++)
                        population.offspring.add((Melody) (offspring.elementAt(i)).clone());

                population.size = size;

                return (population);
        }

        /**
         * Set active graphics context.
         *
         * @param g
         *             Graphics context.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 15 Jun 2008
         */
        public void setGraphics(Graphics g) {
                this.g = g;
        }
}
```

MIDIDERMIClient.java

```
/*******************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                    *
 * New Bulgarian University                                                    *
 *                                                                             *
 * Copyright (c) 2007, 2008 Todor Balabanov                                    *
 *                                                                             *
 * http://tdb.hit.bg/                                                          *
 *                                                                             *
 * This program is free software; you can redistribute it and/or modify        *
 * it under the terms of the GNU General Public License as published by        *
 * the Free Software Foundation; either version 2 of the License, or           *
 * (at your option) any later version.                                         *
 *                                                                             *
 * This program is distributed in the hope that it will be useful,             *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of              *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the               *
 * GNU General Public License for more details.                                *
 *                                                                             *
 * You should have received a copy of the GNU General Public License along     *
 * with this program; if not, write to the Free Software Foundation, Inc.,     *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.                 *
 *******************************************************************************/

package ru.mail.teodorgig.mididermi.client;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.net.MalformedURLException;
```

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.NotBoundException;
import java.rmi.RMISecurityManager;

import javax.swing.JApplet;
import javax.swing.JFrame;

import ru.mail.teodorgig.mididermi.common.MIDIDERMITask;
import ru.mail.teodorgig.mididermi.common.MIDIDERMIInterface;

/**
 * RMI client luncher. All remote side calculations are done here. This class is
 * responsible to request calculating task from the server. In parallel of the
 * calculation process user evaluation is done.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public class MIDIDERMIClient extends JApplet implements KeyListener, Runnable {
	/**
	 * Default serial version uid.
	 */
	private static final long serialVersionUID = 1L;

	/**
	 * Instance of the remote server object.
	 */
	MIDIDERMIInterface simpleServerObject = null;

	/**
	 * Handle to task in progress to proceed keyboard events.
	 */
	private MIDIDERMITask task = null;

	/**
	 * Perform music playing and scoring.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 26 May 2008
	 */
	private void perform() {
		try {
			if (task == null) {
				task = simpleServerObject.request();

				Graphics g = this.getGraphics();
				g.setClip(this.getX(), this.getY(), this.getWidth(),
this.getHeight());
				task.setGraphics(g);

				(new Thread(this)).start();
			}
		} catch (Exception ex) {
			ex.printStackTrace();
		}
	}

	/**
	 * Standard key listener key pressed method.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
```

```
         * @date 26 May 2008
         */
        public void keyPressed(KeyEvent event) {
        }

        /**
         * Standard key listener key released method. By pressing up or down arrow
         * user is able to change the score of the melody.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 26 May 2008
         */
        public void keyReleased(KeyEvent event) {
                if (event.getKeyCode() == KeyEvent.VK_UP) {
                        task.scoreUp();
                }

                if (event.getKeyCode() == KeyEvent.VK_DOWN) {
                        task.scoreDown();
                }
        }

        /**
         * Standard key listener key typed method.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 26 May 2008
         */
        public void keyTyped(KeyEvent event) {
        }

        /**
         * Obtain RMI object.
         *
         * @param address
         *             Address of RMI remote server.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 26 May 2008
         */
        public void obtainRmiObject(String address) {
                System.setSecurityManager(new RMISecurityManager());

                try {
                        simpleServerObject = (MIDIDERMIInterface) Naming.lookup("rmi://" + address +
"/MIDIDERMIImplementInstance");
                } catch (MalformedURLException ex) {
                        ex.printStackTrace();
                } catch (RemoteException ex) {
                        ex.printStackTrace();
                } catch (NotBoundException ex) {
                        ex.printStackTrace();
                }
        }

        /**
         * Standard applet init method.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
```

```
 * @date 06 Jun 2008
 */
public void init() {
	this.addKeyListener(this);

	setSize(640, 480);
	setLocation(0, 0);

	try {
		String address = getParameter("rmi-server-address");
		if (address != null) {
			obtainRmiObject(address);
		}
	} catch (Exception ex) {
		/*
		 * Do not print anything because information is provided in main()
		 * when code is executed as stand-alone application.
		 */
	}

	setBackground(Color.BLACK);
	requestFocus();

	perform();
}

/**
 * Standard thread method run. It is needed because class is runnable.
 *
 * http://java.sys-con.com/read/46096.htm
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 07 Jun 2008
 */
public void run() {
	task.calculate();

	try {
		simpleServerObject.response(task);
	} catch (RemoteException ex) {
		ex.printStackTrace();
	}

	task = null;
}

/**
 * Main starting point of the program. In this method remote instance is
 * invoked and calculations are done in a loop.
 *
 * @param args
 *            Command line parameters.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public static void main(String[] args) {
	JFrame frame = new JFrame("MIDIDERMI client ...");
	frame.setSize(640, 480);

	MIDIDERMIClient applet = new MIDIDERMIClient();
	applet.obtainRmiObject(args[0]);

	frame.add(applet);
	frame.setVisible(true);
```

```
                applet.init();
                applet.start();
        }
}
```

MIDIDERMIInterface.java

```
/*****************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                  *
 * New Bulgarian University                                                  *
 *                                                                           *
 * Copyright (c) 2007, 2008 Todor Balabanov                                  *
 *                                                                           *
 * http://tdb.hit.bg/                                                        *
 *                                                                           *
 * This program is free software; you can redistribute it and/or modify      *
 * it under the terms of the GNU General Public License as published by      *
 * the Free Software Foundation; either version 2 of the License, or         *
 * (at your option) any later version.                                       *
 *                                                                           *
 * This program is distributed in the hope that it will be useful,           *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of            *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the             *
 * GNU General Public License for more details.                              *
 *                                                                           *
 * You should have received a copy of the GNU General Public License along   *
 * with this program; if not, write to the Free Software Foundation, Inc.,   *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.               *
 *****************************************************************************/

package ru.mail.teodorgig.mididermi.common;

import java.rmi.Remote;

/**
 * Remote interface functionality. Methods which server provides to the remote
 * clients.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public interface MIDIDERMIInterface extends Remote {
        /**
         * Request the task for calculation. By this method remote client request
         * task for calculation.
         *
         * @return Task.
         *
         * @throws java.rmi.RemoteException
         *             RMI should be safe.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 22 May 2008
         */
        public MIDIDERMITask request() throws java.rmi.RemoteException;

        /**
         * Return the calculated task. The result of the calculation on the remote
```

```
         * side is returned by the requested task structure.
         *
         * @param val
         *            Task.
         *
         * @throws java.rmi.RemoteException
         *             RMI should be safe.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 22 May 2008
         */
        public void response(MIDIDERMITask val) throws java.rmi.RemoteException;
}
```

MIDIDERMITask.java

```
/*******************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                    *
 * New Bulgarian University                                                    *
 *                                                                             *
 * Copyright (c) 2007, 2008 Todor Balabanov                                    *
 *                                                                             *
 * http://tdb.hit.bg/                                                          *
 *                                                                             *
 * This program is free software; you can redistribute it and/or modify        *
 * it under the terms of the GNU General Public License as published by        *
 * the Free Software Foundation; either version 2 of the License, or           *
 * (at your option) any later version.                                         *
 *                                                                             *
 * This program is distributed in the hope that it will be useful,             *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of              *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the               *
 * GNU General Public License for more details.                                *
 *                                                                             *
 * You should have received a copy of the GNU General Public License along     *
 * with this program; if not, write to the Free Software Foundation, Inc.,     *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.                 *
 *******************************************************************************/

package ru.mail.teodorgig.mididermi.common;

import java.awt.Graphics;
import java.io.Serializable;

import ru.mail.teodorgig.mididermi.base.Population;

/**
 * Task to be calculated on the remote side. Task class presents working logic
 * of the melody evolution and melody user based evaluation.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public class MIDIDERMITask implements Serializable {
        /**
         * Default serial version uid.
         */
        private static final long serialVersionUID = 1L;
```

```
/**
 * Number of epochs to be calculated in one invocation.
 */
private int numberOfEpochs = 1;

/**
 * Population to evolve.
 */
private Population population;

/**
 * Remote task constructor. It is used only to create copy of the input
 * parameter.
 *
 * @param population
 *            Population to evolve on the remote side.
 *
 * @param numberOfEpochs
 *            Number of epochs to be calculated on the client side.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public MIDIDERMITask(Population population, int numberOfEpochs) {
	super();

	this.population = (Population) population.clone();
	this.numberOfEpochs = numberOfEpochs;
}

/**
 * Remote side calculations. Executes DE evolution process for given number
 * of epochs.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public void calculate() {
	population.epoches(numberOfEpochs);
}

/**
 * Result of the remote calculation. Clone and return current stated of the
 * active population.
 *
 * @return Remote calculated population.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public Population getResult() {
	return ((Population) population.clone());
}

/**
 * Score up of the currently playing melody. Method is provided as
 * connection between graphic user interface and real melody object.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
```

```
	 * @date 26 May 2008
	 */
	public void scoreUp() {
		population.scoreUp();
	}

	/**
	 * Score down of the currently playing melody. Method is provided as
	 * connection between graphic user interface and real melody object.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 26 May 2008
	 */
	public void scoreDown() {
		population.scoreDown();
	}

	/**
	 * Set active graphics context.
	 *
	 * @param g
	 *            Graphics context.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 15 Jun 2008
	 */
	public void setGraphics(Graphics g) {
		population.setGraphics(g);
	}

	/**
	 * Number of evolution epochs getter.
	 *
	 * @return Number of epochs.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 27 Jun 2008
	 */
	public int getNumberOfEpochs() {
		return (numberOfEpochs);
	}

	/**
	 * Number of evolution epochs setter.
	 *
	 * @param numberOfEpochs
	 *            Number of epochs.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 27 Jun 2008
	 */
	public void setNumberOfEpochs(int numberOfEpochs) {
		this.numberOfEpochs = numberOfEpochs;
	}
}
```

DatabaseMediator.java

```
/*******************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                    *
 * New Bulgarian University                                                    *
 *                                                                             *
 * Copyright (c) 2007, 2008 Todor Balabanov                                    *
 *                                                                             *
 * http://tdb.hit.bg/                                                          *
 *                                                                             *
 * This program is free software; you can redistribute it and/or modify        *
 * it under the terms of the GNU General Public License as published by        *
 * the Free Software Foundation; either version 2 of the License, or           *
 * (at your option) any later version.                                         *
 *                                                                             *
 * This program is distributed in the hope that it will be useful,             *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of              *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the               *
 * GNU General Public License for more details.                                *
 *                                                                             *
 * You should have received a copy of the GNU General Public License along     *
 * with this program; if not, write to the Free Software Foundation, Inc.,     *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.                 *
 *******************************************************************************/

package ru.mail.teodorgig.mididermi.database;

import java.util.StringTokenizer;
import java.util.Vector;
import java.io.IOException;
import java.util.Properties;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.DriverManager;
import java.io.FileInputStream;
import java.io.FileNotFoundException;

import ru.mail.teodorgig.mididermi.base.Melody;
import ru.mail.teodorgig.mididermi.base.Note;

/**
 * Database mediator handles database functionality for loading and storing
 * melody data.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 23 Jun 2008
 */
public class DatabaseMediator {
	/**
	 * Database connection host.
	 */
	private static String host = "";

	/**
	 * Database connection port.
	 */
	private static String port = "";

	/**
	 * Database username to be used for connection.
	 */
	private static String username = "";
```

```
/**
 * Password of user to be connected.
 */
private static String password = "";

/**
 * Working database name.
 */
private static String database = "";

/**
 * Database connection URL string.
 */
private static String url;

/**
 * Connection object.
 */
private static Connection connection = null;

/**
 * Obtain database properties.
 */
static {
	Properties properties = new Properties();

	try {
		properties.load(new FileInputStream("database.properties"));
	} catch (FileNotFoundException ex) {
		ex.printStackTrace();
	} catch (IOException ex) {
		ex.printStackTrace();
	}

	host = properties.getProperty("host");
	port = properties.getProperty("port");
	username = properties.getProperty("username");
	password = properties.getProperty("password");
	database = properties.getProperty("database");

	url = "jdbc:postgresql://" + host + ":" + port + "/" + database;

	try {
		Class.forName("org.postgresql.Driver");
	} catch (ClassNotFoundException ex) {
		ex.printStackTrace();
	}

	try {
		connection = DriverManager.getConnection(url, username, password);
	} catch (SQLException ex) {
		ex.printStackTrace();
	}
}

/**
 * Extract all available melodies stored into database.
 *
 * @return All melodies stored into database.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 25 Jun 2008
 */
public static Vector<Melody> loadMelodies() {
	CallableStatement statement = null;
	ResultSet result = null;
	Vector<Melody> melodies = new Vector<Melody>();
```

```
		if (connection == null) {
			return (melodies);
		}

		try {
			statement = connection.prepareCall("{ call get_all_melodies() }",
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
			result = statement.executeQuery();

			while (result.next() != false) {
				Melody melody = new Melody();
				melody.setId(result.getLong("system_id"));
				String sequence = result.getString("sequence");
				StringTokenizer tokenizer = new StringTokenizer(sequence, " ", false);
				while (tokenizer.hasMoreElements()) {
					Note note = new Note();
					try {
						note.setNote((new Integer((String)
tokenizer.nextElement())).intValue());
						note.setOffset((new Integer((String)
tokenizer.nextElement())).intValue());
						note.setDuration((new Integer((String)
tokenizer.nextElement())).intValue());
						note.setVelocity((new Integer((String)
tokenizer.nextElement())).intValue());
					} catch (Exception ex) {
						ex.printStackTrace();
					}
					melody.addNote(note);
				}
				melody.setTimber(result.getInt("timber"));
				melody.setScore(result.getInt("score"));
				melody.setGenre(result.getInt("genre"));
			}
		} catch (SQLException ex) {
			ex.printStackTrace();
		}

		return (melodies);
	}

	/**
	 * Store list of melodies into database.
	 *
	 * @param melodies
	 *            Vector of melodies.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 25 Jun 2008
	 */
	public static void storeMelodies(Vector<Melody> melodies) {
		CallableStatement statement = null;
		Melody melody = null;

		if (connection == null) {
			return;
		}

		for (int i = 0; i < melodies.size(); i++) {
			melody = melodies.elementAt(i);

			try {
				statement = connection.prepareCall("{ call
add_melody(?, ?, ?, ?, ?) }");

				statement.setLong(1, melody.getId());
				statement.setString(2, melody.getNotesInNumbers());
				statement.setInt(3, melody.getTimber());
```

```
                                statement.setInt(4, melody.getScore());
                                statement.setInt(5, melody.getGenre());

                                statement.execute();
                        } catch (SQLException ex) {
                                ex.printStackTrace();
                        }
                }
        }
}
```

DatabaseSetMelodiesProvider.java

```
/*****************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                  *
 * New Bulgarian University                                                  *
 *                                                                           *
 * Copyright (c) 2007, 2008 Todor Balabanov                                  *
 *                                                                           *
 * http://tdb.hit.bg/                                                        *
 *                                                                           *
 * This program is free software; you can redistribute it and/or modify      *
 * it under the terms of the GNU General Public License as published by      *
 * the Free Software Foundation; either version 2 of the License, or         *
 * (at your option) any later version.                                       *
 *                                                                           *
 * This program is distributed in the hope that it will be useful,           *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of            *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the             *
 * GNU General Public License for more details.                              *
 *                                                                           *
 * You should have received a copy of the GNU General Public License along   *
 * with this program; if not, write to the Free Software Foundation, Inc.,   *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.               *
 *****************************************************************************/

package ru.mail.teodorgig.mididermi.providers;

import java.util.Vector;

import ru.mail.teodorgig.mididermi.base.Melody;
import ru.mail.teodorgig.mididermi.database.DatabaseMediator;

/**
 * Provide set of melodies from a given melody list.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 30 May 2008
 */
public class DatabaseSetMelodiesProvider {
        /**
         * Provide set of melodies from database.
         *
         * @return Set of melodies from database.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 24 Jun 2008
         */
        public static Vector<Melody> provide() {
```

```
                return (DatabaseMediator.loadMelodies());
        }
}
```

FileMelodyProvider.java

```
/*******************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                    *
 * New Bulgarian University                                                    *
 *                                                                             *
 * Copyright (c) 2007, 2008 Todor Balabanov                                    *
 *                                                                             *
 * http://tdb.hit.bg/                                                          *
 *                                                                             *
 * This program is free software; you can redistribute it and/or modify        *
 * it under the terms of the GNU General Public License as published by        *
 * the Free Software Foundation; either version 2 of the License, or           *
 * (at your option) any later version.                                         *
 *                                                                             *
 * This program is distributed in the hope that it will be useful,             *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of              *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the               *
 * GNU General Public License for more details.                                *
 *                                                                             *
 * You should have received a copy of the GNU General Public License along     *
 * with this program; if not, write to the Free Software Foundation, Inc.,     *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.                 *
 *******************************************************************************/

package ru.mail.teodorgig.mididermi.providers;

import java.io.FileReader;
import java.io.IOException;
import java.io.BufferedReader;
import java.util.StringTokenizer;

import ru.mail.teodorgig.mididermi.base.Note;
import ru.mail.teodorgig.mididermi.base.Melody;

/**
 * File melody provider class is responsible to generate melody with length,
 * timber and notes specified in a descriptor file.
 *
 * The format is [octave][note][duration].
 *
 * http://www.harmony-central.com/MIDI/Doc/table2.html
 *
 * [octave] is number between -1 and 9.
 *
 * [note] is symbol(s): A, A#, B, C, C#, D, D#, E, F, F#, G, G#.
 *
 * [duration] is in milliseconds.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 29 May 2008
 */
public class FileMelodyProvider {
        private static Note parseNote(String string) throws NotValidDescriptorFileException {
                String token = string;

                int octave = -2;
                if (string.charAt(0) == '-' && string.charAt(1) == '1') {
```

```
                octave = -1;
                string = string.substring(2);
            } else if (string.charAt(0) == '0') {
                octave = 0;
                string = string.substring(1);
            } else if (string.charAt(0) == '1') {
                octave = 1;
                string = string.substring(1);
            } else if (string.charAt(0) == '2') {
                octave = 2;
                string = string.substring(1);
            } else if (string.charAt(0) == '3') {
                octave = 3;
            } else if (string.charAt(0) == '4') {
                octave = 4;
                string = string.substring(1);
            } else if (string.charAt(0) == '5') {
                octave = 5;
                string = string.substring(1);
            } else if (string.charAt(0) == '6') {
                octave = 6;
                string = string.substring(1);
            } else if (string.charAt(0) == '7') {
                octave = 7;
                string = string.substring(1);
            } else if (string.charAt(0) == '8') {
                octave = 8;
                string = string.substring(1);
            } else if (string.charAt(0) == '9') {
                octave = 9;
                string = string.substring(1);
            } else {
                throw (new NotValidDescriptorFileException("Octave number is not correct [" +
token + "]!"));
            }

            int note = 0;
            if (string.charAt(0) == 'A' && string.charAt(1) != '#') {
                note += (octave + 1) * 12 + 9;
                string = string.substring(1);
            } else if (string.charAt(0) == 'A' && string.charAt(1) == '#') {
                note += (octave + 1) * 12 + 10;
                string = string.substring(2);
            } else if (string.charAt(0) == 'B' && string.charAt(1) != '#') {
                note += (octave + 1) * 12 + 11;
                string = string.substring(1);
            } else if (string.charAt(0) == 'C' && string.charAt(1) != '#') {
                note += (octave + 1) * 12 + 0;
                string = string.substring(1);
            } else if (string.charAt(0) == 'C' && string.charAt(1) == '#') {
                note += (octave + 1) * 12 + 0;
                string = string.substring(2);
            } else if (string.charAt(0) == 'D' && string.charAt(1) != '#') {
                note += (octave + 1) * 12 + 2;
                string = string.substring(1);
            } else if (string.charAt(0) == 'D' && string.charAt(1) == '#') {
                note += (octave + 1) * 12 + 3;
                string = string.substring(2);
            } else if (string.charAt(0) == 'E' && string.charAt(1) != '#') {
                note += (octave + 1) * 12 + 4;
                string = string.substring(1);
            } else if (string.charAt(0) == 'F' && string.charAt(1) != '#') {
                note += (octave + 1) * 12 + 5;
                string = string.substring(1);
            } else if (string.charAt(0) == 'F' && string.charAt(1) == '#') {
                note += (octave + 1) * 12 + 6;
                string = string.substring(2);
            } else if (string.charAt(0) == 'G' && string.charAt(1) != '#') {
                note += (octave + 1) * 12 + 7;
                string = string.substring(1);
            } else if (string.charAt(0) == 'G' && string.charAt(1) == '#') {
```

```
                        note += (octave + 1) * 12 + 8;
                        string = string.substring(2);
                } else {
                        throw (new NotValidDescriptorFileException("Note number is not correct [" +
token + "]!"));
                }

                int duration = 0;
                try {
                        duration = Integer.parseInt(string);
                } catch (NumberFormatException ex) {
                        throw (new NotValidDescriptorFileException("Note duration is not correct
number [" + token + "]!"));
                }

                // TODO Better way to select note velocity is needed.
                return (new Note(note, 0, duration, 64));
        }

        /**
         * Create melody as sequence of music notes by parsing descriptor file. By
         * this way system can be easy populated with a lot of melodies composed by
         * human. Descriptor file name is taken as parameter. If descriptor file is
         * not valid exception is thrown.
         *
         * http://www.harmony-central.com/MIDI/Doc/table2.html
         *
         * @param fileName
         *            File name of descriptor file.
         *
         * @return Melody constructed.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 29 May 2008
         */
        public static Melody provide(String fileName) throws NotValidDescriptorFileException,
IOException {
                // TODO Better file format description as documentation is needed.
                Melody melody = new Melody();

                String melodyText = "";

                BufferedReader in = new BufferedReader(new FileReader(fileName));
                String line = null;
                while ((line = in.readLine()) != null) {
                        melodyText += line;
                }
                in.close();

                StringTokenizer tokenizer = new StringTokenizer(melodyText, " ", false);

                /*
                 * First number into the file format is timber number.
                 */
                if (tokenizer.hasMoreTokens() == false) {
                        throw (new NotValidDescriptorFileException("Timber is not available!"));
                }

                /*
                 * Timber should be positive integer number.
                 */
                int timber = 0;
                try {
                        timber = Integer.parseInt(tokenizer.nextToken());
                } catch (NumberFormatException ex) {
                        throw (new NotValidDescriptorFileException("Timber is not correct number!"));
                }
```

```
                /*
                 * Timber should be between 1 and 128.
                 */
                if (timber < 1 || timber > 128) {
                        throw (new NotValidDescriptorFileException("Timber is not between 1 and
128!"));
                } else {
                        melody.setTimber(timber);
                }
                melody.setId(Melody.getUniqueId());
                melody.setScore(0);

                int timeCounter = 0;

                /*
                 * All notes are fowling after timber.
                 */
                while (tokenizer.hasMoreTokens() == true) {
                        Note note = parseNote(tokenizer.nextToken());

                        // TODO It is not clear how much time is between two notes.
                        note.setOffset(timeCounter);

                        melody.addNote(note);

                        timeCounter += note.getDuration();
                }

                return (melody);
        }
}
```

FileSetMelodiesProvider.java

```
/*****************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                  *
 * New Bulgarian University                                                  *
 *                                                                           *
 * Copyright (c) 2007, 2008 Todor Balabanov                                  *
 *                                                                           *
 * http://tdb.hit.bg/                                                        *
 *                                                                           *
 * This program is free software; you can redistribute it and/or modify      *
 * it under the terms of the GNU General Public License as published by      *
 * the Free Software Foundation; either version 2 of the License, or         *
 * (at your option) any later version.                                       *
 *                                                                           *
 * This program is distributed in the hope that it will be useful,           *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of            *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the             *
 * GNU General Public License for more details.                              *
 *                                                                           *
 * You should have received a copy of the GNU General Public License along   *
 * with this program; if not, write to the Free Software Foundation, Inc.,   *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.               *
 *****************************************************************************/

package ru.mail.teodorgig.mididermi.providers;

import java.io.File;
import java.io.IOException;
import java.util.Vector;

import ru.mail.teodorgig.mididermi.base.Melody;
```

```
/**
 * Provide set of melodies from a given melody list.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 30 May 2008
 */
public class FileSetMelodiesProvider {
	/**
	 * Project subfolder for molody files.
	 */
	private static String TEXT_MELODIES_FOLDER = "../melodies/";

	/**
	 * Provide set of melodies from list of files.
	 *
	 * @return Set of melodies from list of files.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 04 Jun 2008
	 */
	public static Vector<Melody> provide() {
		Vector<Melody> melodies = new Vector<Melody>();

		String filesList[] = (new File(TEXT_MELODIES_FOLDER)).list();

		Melody melody = null;
		for (int i = 0; filesList != null && i < filesList.length; i++) {
			if ((new File(TEXT_MELODIES_FOLDER + filesList[i])).isFile() == true) {
				try {
					melody = FileMelodyProvider.provide(TEXT_MELODIES_FOLDER +
filesList[i]);
					melody.setId(Melody.getUniqueId());
					melodies.add(melody);
				} catch (NotValidDescriptorFileException ex) {
					ex.printStackTrace();
				} catch (IOException ex) {
					ex.printStackTrace();
				}
			}
		}

		return (melodies);
	}
}
```

FractalMelodyProvider.java

```
/*******************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                    *
 * New Bulgarian University                                                    *
 *                                                                             *
 * Copyright (c) 2007, 2008 Todor Balabanov                                    *
 *                                                                             *
 * http://tdb.hit.bg/                                                          *
 *                                                                             *
 * This program is free software; you can redistribute it and/or modify        *
 * it under the terms of the GNU General Public License as published by        *
 * the Free Software Foundation; either version 2 of the License, or           *
 * (at your option) any later version.                                         *
```

```
 *                                                                             *
 * This program is distributed in the hope that it will be useful,             *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of              *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the               *
 * GNU General Public License for more details.                                *
 *                                                                             *
 * You should have received a copy of the GNU General Public License along     *
 * with this program; if not, write to the Free Software Foundation, Inc.,     *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.                 *
 *******************************************************************************/

package ru.mail.teodorgig.mididermi.providers;

import ru.mail.teodorgig.mididermi.base.Note;
import ru.mail.teodorgig.mididermi.base.Melody;

/**
 * Fractal melody provider class is responsible to generate melody with length,
 * timber and notes according proper fractal formulas.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 30 May 2008
 */
public class FractalMelodyProvider {
	/**
	 * Min melody sequence size for random generation.
	 */
	private static final int MIN_FRACTAL_SEQUENCE = 3;

	/**
	 * Max melody sequence size for random generation.
	 */
	private static final int MAX_FRACTAL_SEQUENCE = 15;

	/**
	 * Create fractal melody as sequence of music notes. By this way system can
	 * be easy populated with a lot of melodies.
	 *
	 * @return Melody constructed.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 29 May 2008
	 */
	public static Melody provide() {
		return (provide(MIN_FRACTAL_SEQUENCE + (int) (Math.random() * (MAX_FRACTAL_SEQUENCE -
MIN_FRACTAL_SEQUENCE + 1))));
	}

	/**
	 * Create fractal melody as sequence of music notes. By this way system can
	 * be easy populated with a lot of melodies. The difference with the
	 * previous method is only that length is specified by input parameter.
	 *
	 * @param length
	 *            The length of the melody.
	 *
	 * @return Melody constructed.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 29 May 2008
	 */
	public static Melody provide(int length) {
```

```
                Melody melody = new Melody();

                // TODO Better fractal formula can be implemented.
                Note next = null;
                Note previous = null;
                for (int i = 0; i < length; i++) {
                        next = Note.provideRandom();

                        if (i > 0 && previous.getNote() > next.getNote()) {
                                next.setNote(next.getNote() + 1);
                        } else if (i > 0 && previous.getNote() < next.getNote()) {
                                next.setNote(next.getNote() - 1);
                        }

                        melody.addNote(next);
                        previous = next;
                }

                melody.setTimber(1 + (int) (Math.random() * 128));
                melody.setId(Melody.getUniqueId());
                melody.setScore(0);

                melody.sort();

                return (melody);
        }
}
```

FractalSetMelodiesProvider.java

```
/*****************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                  *
 * New Bulgarian University                                                  *
 *                                                                           *
 * Copyright (c) 2007, 2008 Todor Balabanov                                  *
 *                                                                           *
 * http://tdb.hit.bg/                                                        *
 *                                                                           *
 * This program is free software; you can redistribute it and/or modify      *
 * it under the terms of the GNU General Public License as published by      *
 * the Free Software Foundation; either version 2 of the License, or         *
 * (at your option) any later version.                                       *
 *                                                                           *
 * This program is distributed in the hope that it will be useful,           *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of            *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the             *
 * GNU General Public License for more details.                              *
 *                                                                           *
 * You should have received a copy of the GNU General Public License along   *
 * with this program; if not, write to the Free Software Foundation, Inc.,   *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.               *
 *****************************************************************************/

package ru.mail.teodorgig.mididermi.providers;

import java.util.Vector;

import ru.mail.teodorgig.mididermi.base.Melody;
import ru.mail.teodorgig.mididermi.base.Population;

/**
 * Provide fractal set of melodies.
 *
 * @author Todor Balabanov
 *
```

```
 * @email teodorgig@mail.ru
 *
 * @date 30 May 2008
 */
public class FractalSetMelodiesProvider {
        /**
         * Provide fractal set of melodies.
         *
         * @return Fractal set of melodies.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 30 May 2008
         */
        public static Vector<Melody> provide() {
                return( provide(Population.MIN_FRACTAL_POPULATION + (int) (Math.random() *
(Population.MAX_FRACTAL_POPULATION - Population.MIN_FRACTAL_POPULATION + 1))) );
        }

        /**
         * Provide fractal set of melodies.
         *
         * @param size
         *             Number of chromosomes.
         *
         * @return Fractal set of melodies.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 30 May 2008
         */
        public static Vector<Melody> provide(int size) {
                Vector<Melody> melodies = new Vector<Melody>();

                Melody melody = null;
                for (int i = 0; i < size; i++) {
                        melody = FractalMelodyProvider.provide();
                        melody.setId(Melody.getUniqueId());

                        melodies.add(melody);
                }

                return (melodies);
        }
}
```

NotValidDescriptorFileException.java

```
/*******************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                    *
 * New Bulgarian University                                                    *
 *                                                                             *
 * Copyright (c) 2007, 2008 Todor Balabanov                                    *
 *                                                                             *
 * http://tdb.hit.bg/                                                          *
 *                                                                             *
 * This program is free software; you can redistribute it and/or modify        *
 * it under the terms of the GNU General Public License as published by        *
 * the Free Software Foundation; either version 2 of the License, or           *
 * (at your option) any later version.                                         *
 *                                                                             *
```

```
 * This program is distributed in the hope that it will be useful,            *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of             *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the              *
 * GNU General Public License for more details.                               *
 *                                                                            *
 * You should have received a copy of the GNU General Public License along    *
 * with this program; if not, write to the Free Software Foundation, Inc.,    *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.                *
 ******************************************************************************/

package ru.mail.teodorgig.mididermi.providers;

/**
 * Not valid descriptor file exception is used when input data in a plain text
 * melody description is not valid as part of the internal file format.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 29 May 2008
 */
public class NotValidDescriptorFileException extends Exception {
	/**
	 * Default serial version uid.
	 */
	private static final long serialVersionUID = 1L;

	/**
	 * Specific exception message.
	 */
	private String message = "";

	/**
	 * Exception constructor.
	 *
	 * @param message
	 *            Specific message.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 29 May 2008
	 */
	public NotValidDescriptorFileException(String message) {
		this.message = message;
	}

	/**
	 * Standard to string method.
	 *
	 * @return String representation of the exception
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 29 May 2008
	 */
	public String toStrrin() {
		return (super.toString() + " " + message);
	}
}
```

RandomMelodyProvider.java

```
/*****************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                  *
 * New Bulgarian University                                                  *
 *                                                                           *
 * Copyright (c) 2007, 2008 Todor Balabanov                                  *
 *                                                                           *
 * http://tdb.hit.bg/                                                        *
 *                                                                           *
 * This program is free software; you can redistribute it and/or modify      *
 * it under the terms of the GNU General Public License as published by      *
 * the Free Software Foundation; either version 2 of the License, or         *
 * (at your option) any later version.                                       *
 *                                                                           *
 * This program is distributed in the hope that it will be useful,           *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of            *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the             *
 * GNU General Public License for more details.                              *
 *                                                                           *
 * You should have received a copy of the GNU General Public License along   *
 * with this program; if not, write to the Free Software Foundation, Inc.,   *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.               *
 *****************************************************************************/

package ru.mail.teodorgig.mididermi.providers;

import ru.mail.teodorgig.mididermi.base.Note;
import ru.mail.teodorgig.mididermi.base.Melody;

/**
 * Random melody provider class is responsible to generate melody with random
 * length, random timber and random notes.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 29 May 2008
 */
public class RandomMelodyProvider {
	/**
	 * Min melody sequence size for random generation.
	 */
	private static final int MIN_RANDOM_SEQUENCE = 3;

	/**
	 * Max melody sequence size for random generation.
	 */
	private static final int MAX_RANDOM_SEQUENCE = 15;

	/**
	 * Create random melody as sequence of random music notes. By this way
	 * system can be easy populated with a lot of melodies, but melodies with no
	 * sense.
	 *
	 * @return Melody constructed.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 29 May 2008
	 */
	public static Melody provide() {
		return (provide(MIN_RANDOM_SEQUENCE + (int) (Math.random() * (MAX_RANDOM_SEQUENCE -
MIN_RANDOM_SEQUENCE + 1))));
	}
```

```
	/**
	 * Create random melody as sequence of random music notes. By this way
	 * system can be easy populated with a lot of melodies, but melodies with no
	 * sense. The difference with the previous method is only that length is
	 * specified by input parameter.
	 *
	 * @param length
	 *            The length of the melody.
	 *
	 * @return Melody constructed.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 29 May 2008
	 */
	public static Melody provide(int length) {
		Melody melody = new Melody();

		for (int i = 0; i < length; i++)
			melody.addNote(Note.provideRandom());

		melody.setTimber(1 + (int) (Math.random() * 128));
		melody.setId(Melody.getUniqueId());
		melody.setScore(0);

		melody.sort();

		return (melody);
	}
}
```

RandomSetMelodiesProvider.java

```
/*******************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                    *
 * New Bulgarian University                                                    *
 *                                                                             *
 * Copyright (c) 2007, 2008 Todor Balabanov                                    *
 *                                                                             *
 * http://tdb.hit.bg/                                                          *
 *                                                                             *
 * This program is free software; you can redistribute it and/or modify        *
 * it under the terms of the GNU General Public License as published by        *
 * the Free Software Foundation; either version 2 of the License, or           *
 * (at your option) any later version.                                         *
 *                                                                             *
 * This program is distributed in the hope that it will be useful,             *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of              *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the               *
 * GNU General Public License for more details.                                *
 *                                                                             *
 * You should have received a copy of the GNU General Public License along     *
 * with this program; if not, write to the Free Software Foundation, Inc.,     *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.                 *
 *******************************************************************************/

package ru.mail.teodorgig.mididermi.providers;

import java.util.Vector;

import ru.mail.teodorgig.mididermi.base.Melody;
import ru.mail.teodorgig.mididermi.base.Population;
```

```
/**
 * Provide random set of melodies.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 30 May 2008
 */
public class RandomSetMelodiesProvider {
	/**
	 * Provide random set of melodies.
	 *
	 * @return Random set of melodies.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 30 May 2008
	 */
	public static Vector<Melody> provide() {
		return( provide(Population.MIN_RANDOM_POPULATION + (int) (Math.random() *
(Population.MAX_RANDOM_POPULATION - Population.MIN_RANDOM_POPULATION + 1))) );
	}

	/**
	 * Provide random set of melodies.
	 *
	 * @param size
	 *            Number of chromosomes.
	 *
	 * @return Random set of melodies.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 30 May 2008
	 */
	public static Vector<Melody> provide(int size) {
		Vector<Melody> melodies = new Vector<Melody>();

		Melody melody = null;
		for (int i = 0; i < size; i++) {
			melody = RandomMelodyProvider.provide();
			melody.setId(Melody.getUniqueId());

			melodies.add(melody);
		}

		return (melodies);
	}
}
```

MelodyPool.java

```
/*******************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                    *
 * New Bulgarian University                                                    *
 *                                                                             *
 * Copyright (c) 2007, 2008 Todor Balabanov                                    *
 *                                                                             *
 * http://tdb.hit.bg/                                                          *
```

```
 *                                                                             *
 * This program is free software; you can redistribute it and/or modify        *
 * it under the terms of the GNU General Public License as published by        *
 * the Free Software Foundation; either version 2 of the License, or           *
 * (at your option) any later version.                                         *
 *                                                                             *
 * This program is distributed in the hope that it will be useful,             *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of              *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the               *
 * GNU General Public License for more details.                                *
 *                                                                             *
 * You should have received a copy of the GNU General Public License along     *
 * with this program; if not, write to the Free Software Foundation, Inc.,     *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.                 *
 *******************************************************************************/

package ru.mail.teodorgig.mididermi.server;

import java.util.Vector;

import ru.mail.teodorgig.mididermi.base.Melody;
import ru.mail.teodorgig.mididermi.base.Population;
import ru.mail.teodorgig.mididermi.database.DatabaseMediator;
import ru.mail.teodorgig.mididermi.providers.DatabaseSetMelodiesProvider;
import ru.mail.teodorgig.mididermi.providers.FileSetMelodiesProvider;
import ru.mail.teodorgig.mididermi.providers.FractalSetMelodiesProvider;
import ru.mail.teodorgig.mididermi.providers.RandomSetMelodiesProvider;

/**
 * Melody pool is the common server place where all melodies are presented in
 * the server's RAM.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 30 May 2008
 */
public class MelodyPool {
	/**
	 * Min pool subset size.
	 */
	private int minSubsetSize = 2;

	/**
	 * Max pool subset size.
	 */
	private int maxSubsetSize = 2;

	/**
	 * Common melody pool.
	 */
	private Vector<Melody> pool;

	/**
	 * How many random melodies to be created.
	 */
	private int randomMelodiesAmount = 0;

	/**
	 * How many random melodies to be created.
	 */
	private int fractalMelodiesAmount = 0;

	/**
	 * Flag for loading melodies from database.
	 */
	private boolean loadMelodiesFromDatabase = false;

	/**
	 * Flag for loading melodies from text files.
```

```
 */
private boolean loadMelodiesFromFiles = false;

/**
 * Flag for storing melodies into database.
 */
private boolean storeMelodiesIntoDatabase = false;

/**
 * Flag for storing melodies into MIDI files.
 */
private boolean storeMelodiesIntoFiles = false;

/**
 * Add set of melodies to existing pool.
 *
 * @param melodies
 *            Set of melodies to be added.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 24 Jun 2008
 */
private void addMelodies(Vector<Melody> melodies) {
        for (int i = 0; i < melodies.size(); i++) {
                pool.add(melodies.elementAt(i));
        }
}

/**
 * Default constructor.
 */
public MelodyPool() {
        pool = new Vector<Melody>();
}

/**
 * Initialize melody pool with melodies.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 30 May 2008
 */
public void init() {
        pool.clear();

        if (randomMelodiesAmount > 0) {
                addMelodies(RandomSetMelodiesProvider.provide(randomMelodiesAmount));
        }

        if (fractalMelodiesAmount > 0) {
                addMelodies(FractalSetMelodiesProvider.provide(fractalMelodiesAmount));
        }

        if (loadMelodiesFromDatabase = true) {
                addMelodies(DatabaseSetMelodiesProvider.provide());
        }

        if (loadMelodiesFromFiles = true) {
                addMelodies(FileSetMelodiesProvider.provide());
        }
}

/**
 * Standard finalization method.
 *
 * @author Todor Balabanov
```

```
         *
         * @email teodorgig@mail.ru
         *
         * @date 24 Jun 2008
         */
        public void persistentStore() {
                if (this.storeMelodiesIntoDatabase == true) {
                        DatabaseMediator.storeMelodies(pool);
                }

                if (this.storeMelodiesIntoFiles == true) {
                        MidiFileProducer.produce(pool);
                }
        }

        /**
         * Select random melodies from the common pool. From the server side not all
         * general population is sent to the client side. By random selection only
         * part of the general population is sent to the client side. By this way
         * different clients have different part of the melodies set and more
         * options to provide unique melodies.
         *
         * @return Population.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 22 May 2008
         */
        public Population provideRandomSubset() {
                int num = minSubsetSize + (int) (Math.random() * (maxSubsetSize - minSubsetSize +
1));

                if (num > pool.size())
                        num = pool.size();

                boolean take[] = new boolean[pool.size()];
                for (int i = 0; i < take.length; i++)
                        take[i] = false;

                while (num > 0) {
                        int index = (int) (Math.random() * pool.size());
                        if (take[index] == false) {
                                take[index] = true;
                                num--;
                        }
                }

                Population population = new Population();

                for (int i = 0; i < pool.size(); i++)
                        if (take[i] == true)
                                population.add((Melody) pool.elementAt(i).clone());

                return (population);
        }

        /**
         * Merge pool with the new set of melodies. Server side merge the general
         * population with population returned from the client side. By this way
         * general population is getting more varied.
         *
         * @param melodies
         *            New set of melodies.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 22 May 2008
```

```
         */
        public void merge(Vector<Melody> melodies) {
                for (int i = 0; i < melodies.size(); i++) {
                        boolean original = true;

                        for (int j = 0; j < pool.size(); j++)
                                if (pool.elementAt(j).equals(melodies.elementAt(i))) {
                                     pool.elementAt(j).setScore(pool.elementAt(j).getScore() +
melodies.elementAt(i).getScore());
                                        original = false;
                                        break;
                                }

                        if (original == true)
                                pool.add(melodies.elementAt(i));
                }
        }

        /**
         * Random melodies amount getter.
         *
         * @return Random melodies amount.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 24 Jun 2008
         */
        public int getRandomMelodiesAmount() {
                return (randomMelodiesAmount);
        }

        /**
         * Random melodies amount setter.
         *
         * @param randomMelodiesAmount
         *             Random melodies amount.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 24 Jun 2008
         */
        public void setRandomMelodiesAmount(int randomMelodiesAmount) {
                this.randomMelodiesAmount = randomMelodiesAmount;
        }

        /**
         * Fractal melodies amount setter.
         *
         * @param randomMelodiesAmount
         *             Random melodies amount.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 24 Jun 2008
         */
        public int getFractalMelodiesAmount() {
                return (fractalMelodiesAmount);
        }

        /**
         * Fractal melodies amount setter.
         *
         * @param fractalMelodiesAmount
         *             Random melodies amount.
         *
```

```
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 24 Jun 2008
 */
public void setFractalMelodiesAmount(int fractalMelodiesAmount) {
	this.fractalMelodiesAmount = fractalMelodiesAmount;
}

/**
 * Loading melodies form database flag getter.
 *
 * @return Flag value.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 24 Jun 2008
 */
public boolean isLoadMelodiesFromDatabase() {
	return (loadMelodiesFromDatabase);
}

/**
 * Loading melodies form database flag setter.
 *
 * @param loadMelodiesFromDatabase
 *            Flag value.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 24 Jun 2008
 */
public void setLoadMelodiesFromDatabase(boolean loadMelodiesFromDatabase) {
	this.loadMelodiesFromDatabase = loadMelodiesFromDatabase;
}

/**
 * Loading melodies form files flag getter.
 *
 * @return Flag value.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 24 Jun 2008
 */
public boolean isLoadMelodiesFromFiles() {
	return (loadMelodiesFromFiles);
}

/**
 * Loading melodies form files flag setter.
 *
 * @param loadMelodiesFromFiles
 *            Flag value.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 24 Jun 2008
 */
public void setLoadMelodiesFromFiles(boolean loadMelodiesFromFiles) {
	this.loadMelodiesFromFiles = loadMelodiesFromFiles;
}
```

```
/**
 * Storing melodies into database flag getter.
 *
 * @return Flag value.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 24 Jun 2008
 */
public boolean isStoreMelodiesIntoDatabase() {
	return (storeMelodiesIntoDatabase);
}

/**
 * Storing melodies into database flag setter.
 *
 * @param storeMelodiesIntoDatabase
 *            Flag value.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 24 Jun 2008
 */
public void setStoreMelodiesIntoDatabase(boolean storeMelodiesIntoDatabase) {
	this.storeMelodiesIntoDatabase = storeMelodiesIntoDatabase;
}

/**
 * Storing melodies into files flag getter.
 *
 * @return Flag vealue.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 24 Jun 2008
 */
public boolean isStoreMelodiesIntoFiles() {
	return (storeMelodiesIntoFiles);
}

/**
 * Storing melodies into files flag setter.
 *
 * @param storeMelodiesIntoFiles
 *            Flag value.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 24 Jun 2008
 */
public void setStoreMelodiesIntoFiles(boolean storeMelodiesIntoFiles) {
	this.storeMelodiesIntoFiles = storeMelodiesIntoFiles;
}

/**
 * Minimum pool subset size getter.
 *
 * @return Minimum size.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
```

```
         *
         * @date 27 Jun 2008
         */
        public int getMinSubsetSize() {
                return( minSubsetSize );
        }

        /**
         * Minimum pool subset size setter.
         *
         * @param minSubsetSize
         *            Minimum size.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 27 Jun 2008
         */
        public void setMinSubsetSize(int minSubsetSize) {
                this.minSubsetSize = minSubsetSize;

                /*
                 * Les than 2 chromosomes are not able to produce generation.
                 */
                if (this.minSubsetSize < 2) {
                        this.minSubsetSize = 2;
                }
        }

        /**
         * Maximum pool subset size getter.
         *
         * @return Maximum size.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 27 Jun 2008
         */
        public int getMaxSubsetSize() {
                return maxSubsetSize;
        }

        /**
         * Maximum pool subset size setter.
         *
         * @param maxSubsetSize Maximum size.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 27 Jun 2008
         */
        public void setMaxSubsetSize(int maxSubsetSize) {
                this.maxSubsetSize = maxSubsetSize;

                /*
                 * Les than 2 chromosomes are not able to produce generation.
                 */
                if (this.maxSubsetSize < 2) {
                        this.maxSubsetSize = 2;
                }
        }
}
```

MIDIDERMIImplement.java

```
/*****************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                  *
 * New Bulgarian University                                                  *
 *                                                                           *
 * Copyright (c) 2007, 2008 Todor Balabanov                                  *
 *                                                                           *
 * http://tdb.hit.bg/                                                        *
 *                                                                           *
 * This program is free software; you can redistribute it and/or modify      *
 * it under the terms of the GNU General Public License as published by      *
 * the Free Software Foundation; either version 2 of the License, or         *
 * (at your option) any later version.                                       *
 *                                                                           *
 * This program is distributed in the hope that it will be useful,           *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of            *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the             *
 * GNU General Public License for more details.                              *
 *                                                                           *
 * You should have received a copy of the GNU General Public License along   *
 * with this program; if not, write to the Free Software Foundation, Inc.,   *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.               *
 *****************************************************************************/

package ru.mail.teodorgig.mididermi.server;

import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

import ru.mail.teodorgig.mididermi.common.MIDIDERMITask;
import ru.mail.teodorgig.mididermi.common.MIDIDERMIInterface;

/**
 * RMI server implementation. All task are given from here and all results are
 * collected here.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public class MIDIDERMIImplement extends UnicastRemoteObject implements MIDIDERMIInterface {
	/**
	 * Default serial version uid.
	 */
	private static final long serialVersionUID = 1L;

	/**
	 * Minimum number of epochs to be calculated on the client side.
	 */
	private int minNumEpochs = 0;

	/**
	 * Maximum number of epochs to be calculated on the client side.
	 */
	private int maxNumEpochs = 1;

	/**
	 * Melody pool holding all server side available melodies.
	 */
	private MelodyPool melodyPool;

	/**
	 * RMI server constructor. Binding to the RMI register and initial
	 * population creation.
```

```
         *
         * @param name
         *            Name of the service.
         *
         * @throws RemoteException
         *             RMI should be safe.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 22 May 2008
         */
        public MIDIDERMIImplement(String name) throws RemoteException {
                super();

                try {
                        Naming.rebind(name, this);
                } catch (Exception ex) {
                        ex.printStackTrace();
                }

                melodyPool = new MelodyPool();

                melodyPool.init();
        }

        /**
         * Request the task for calculation. Client side call the method to obtain
         * task for calculation.
         *
         * @return Task.
         *
         * @throws java.rmi.RemoteException
         *             RMI should be safe.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 22 May 2008
         */
        public MIDIDERMITask request() throws RemoteException {
                int numberOfEpochs = minNumEpochs + (int) (Math.random() * (maxNumEpochs -
minNumEpochs + 1));

                return (new MIDIDERMITask(melodyPool.provideRandomSubset(), numberOfEpochs));
        }

        /**
         * Return the calculated task. Client side call the method to provide result
         * of calculations done.
         *
         * @param val
         *            Task.
         *
         * @throws java.rmi.RemoteException
         *             RMI should be safe.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 22 May 2008
         */
        public void response(MIDIDERMITask val) throws RemoteException {
                melodyPool.merge((val.getResult()).getMelodies());

                melodyPool.persistentStore();
        }
```

```
        /**
         * Initialize melody pool parameters.
         *
         * @param minPoolSubset
         *            Minimum pool subset size.
         *
         * @param maxPoolSubset
         *            Maximum pool subset size.
         *
         * @param randomMelodiesAmount
         *            Random melodies amount.
         *
         * @param fractalMelodiesAmount
         *            Fractal melodies amount.
         *
         * @param loadMelodiesFromDatabase
         *            Loading melodies form database flag.
         *
         * @param loadMelodiesFromFiles
         *            Loading melodies form files flag.
         *
         * @param storeMelodiesIntoDatabase
         *            Storing melodies into database flag.
         *
         * @param storeMelodiesIntoFiles
         *            Storing melodies into files flag.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 24 Jun 2008
         */
        public void init(int minPoolSubset, int maxPoolSubset, int randomMelodiesAmount, int
fractalMelodiesAmount, boolean loadMelodiesFromDatabase, boolean loadMelodiesFromFiles, boolean
storeMelodiesIntoDatabase, boolean storeMelodiesIntoFiles) {
                melodyPool.setMinSubsetSize(minPoolSubset);
                melodyPool.setMaxSubsetSize(maxPoolSubset);
                melodyPool.setRandomMelodiesAmount(randomMelodiesAmount);
                melodyPool.setFractalMelodiesAmount(fractalMelodiesAmount);
                melodyPool.setLoadMelodiesFromDatabase(loadMelodiesFromDatabase);
                melodyPool.setLoadMelodiesFromFiles(loadMelodiesFromFiles);
                melodyPool.setStoreMelodiesIntoDatabase(storeMelodiesIntoDatabase);
                melodyPool.setStoreMelodiesIntoFiles(storeMelodiesIntoFiles);
        }

        /**
         * Minimum number of epochs getter.
         *
         * @return Minimum number of epochs.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 27 Jun 2008
         */
        public int getMinNumEpochs() {
                return (minNumEpochs);
        }

        /**
         * Minimum number of epochs setter.
         *
         * @param minNumEpochs
         *            Minimum number of epochs.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
```

```
         * @date 27 Jun 2008
         */
        public void setMinNumEpochs(int minNumEpochs) {
                this.minNumEpochs = minNumEpochs;
        }

        /**
         * Maximum number of epochs setter.
         *
         * @return Maximum number of epochs.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 27 Jun 2008
         */
        public int getMaxNumEpochs() {
                return (maxNumEpochs);
        }

        /**
         * Maximum number of epochs setter.
         *
         * @param maxNumEpochs
         *            Maximum number of epochs.
         *
         * @author Todor Balabanov
         *
         * @email teodorgig@mail.ru
         *
         * @date 27 Jun 2008
         */
        public void setMaxNumEpochs(int maxNumEpochs) {
                this.maxNumEpochs = maxNumEpochs;
        }
}
```

MIDIDERMIServer.java

```
/*******************************************************************************
 * MIDI-DE-RMI, Version 0.2                                                    *
 * New Bulgarian University                                                    *
 *                                                                             *
 * Copyright (c) 2007, 2008 Todor Balabanov                                    *
 *                                                                             *
 * http://tdb.hit.bg/                                                          *
 *                                                                             *
 * This program is free software; you can redistribute it and/or modify        *
 * it under the terms of the GNU General Public License as published by        *
 * the Free Software Foundation; either version 2 of the License, or           *
 * (at your option) any later version.                                         *
 *                                                                             *
 * This program is distributed in the hope that it will be useful,             *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of              *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the               *
 * GNU General Public License for more details.                                *
 *                                                                             *
 * You should have received a copy of the GNU General Public License along     *
 * with this program; if not, write to the Free Software Foundation, Inc.,     *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.                 *
 *******************************************************************************/

package ru.mail.teodorgig.mididermi.server;
```

```
import java.rmi.RMISecurityManager;

/**
 * RMI server luncher. Create instance of the RMI server object.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 22 May 2008
 */
public class MIDIDERMIServer {
	/**
	 * Main starting point of the program. RMI server application instance is
	 * created.
	 *
	 * @param args
	 *            Command line parameters.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 22 May 2008
	 */
	public static void main(String[] args) {
		System.setSecurityManager(new RMISecurityManager());

		MIDIDERMIImplement implementation = null;

		try {
			implementation = new MIDIDERMIImplement("MIDIDERMIImplementInstance");
		} catch (Exception ex) {
			ex.printStackTrace();
		}

		int minPoolSubset = 2;
		int maxPoolSubset = 2;
		int randomMelodiesAmount = 0;
		int fractalMelodiesAmount = 0;
		boolean loadMelodiesFromDatabase = false;
		boolean loadMelodiesFromFiles = false;
		boolean storeMelodiesIntoDatabase = false;
		boolean storeMelodiesIntoFiles = false;
		for (int i = 0; i < args.length; i++) {
			if (args[i].equals("-MINPOOL")) {
				/*
				 * Minimum pool subset.
				 */
				try {
					minPoolSubset = (new Integer(args[i + 1])).intValue();
				} catch (Exception ex) {
					minPoolSubset = 2;
				}
			} else if (args[i].equals("-MAXPOOL")) {
				/*
				 * Maximum pool subset.
				 */
				try {
					maxPoolSubset = (new Integer(args[i + 1])).intValue();
				} catch (Exception ex) {
					maxPoolSubset = 2;
				}
			} else if (args[i].equals("-MINEPOCHS")) {
				/*
				 * Minimum epochs.
				 */
				try {
					implementation.setMinNumEpochs((new Integer(args[i +
1])).intValue());
				} catch (Exception ex) {
```

```
                                        implementation.setMinNumEpochs(0);
                                }
                        } else if (args[i].equals("-MAXEPOCHS")) {
                                /*
                                 * Maximum epochs.
                                 */
                                try {
                                        implementation.setMaxNumEpochs((new Integer(args[i +
1])).intValue());
                                } catch (Exception ex) {
                                        implementation.setMaxNumEpochs(0);
                                }
                        } else if (args[i].equals("-LR")) {
                                /*
                                 * Load random melodies.
                                 */
                                try {
                                        randomMelodiesAmount = (new Integer(args[i + 1])).intValue();
                                } catch (Exception ex) {
                                        randomMelodiesAmount = 0;
                                }
                        } else if (args[i].equals("-LT")) {
                                /*
                                 * Load fractal melodies.
                                 */
                                try {
                                        fractalMelodiesAmount = (new Integer(args[i + 1])).intValue();
                                } catch (Exception ex) {
                                        fractalMelodiesAmount = 0;
                                }
                        } else if (args[i].equals("-LD")) {
                                /*
                                 * Load all melodies from database.
                                 */
                                loadMelodiesFromDatabase = true;
                        } else if (args[i].equals("-LF")) {
                                /*
                                 * Load all melodies written as text files.
                                 */
                                loadMelodiesFromFiles = true;
                        } else if (args[i].equals("-SD")) {
                                /*
                                 * Store all melodies into database.
                                 */
                                storeMelodiesIntoDatabase = true;
                        } else if (args[i].equals("-SF")) {
                                /*
                                 * Store all melodies as MIDI binary files.
                                 */
                                storeMelodiesIntoFiles = true;
                        }
                }

                implementation.init(minPoolSubset, maxPoolSubset, randomMelodiesAmount,
fractalMelodiesAmount, loadMelodiesFromDatabase, loadMelodiesFromFiles, storeMelodiesIntoDatabase,
storeMelodiesIntoFiles);

                System.out.println("MIDI DE RMI Server bound ...");
        }
}
```

MidiFileProducer.java

```
/*************************************************************************
 * MIDI-DE-RMI, Version 0.2                                              *
```

```
 * New Bulgarian University                                                *
 *                                                                         *
 * Copyright (c) 2007, 2008 Todor Balabanov                                *
 *                                                                         *
 * http://tdb.hit.bg/                                                      *
 *                                                                         *
 * This program is free software; you can redistribute it and/or modify    *
 * it under the terms of the GNU General Public License as published by    *
 * the Free Software Foundation; either version 2 of the License, or       *
 * (at your option) any later version.                                     *
 *                                                                         *
 * This program is distributed in the hope that it will be useful,         *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of          *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the           *
 * GNU General Public License for more details.                            *
 *                                                                         *
 * You should have received a copy of the GNU General Public License along *
 * with this program; if not, write to the Free Software Foundation, Inc., *
 * 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.             *
 ***************************************************************************/

package ru.mail.teodorgig.mididermi.server;

import java.util.Vector;
import java.io.FileWriter;
import java.io.BufferedWriter;

import ru.mail.teodorgig.mididermi.base.Melody;

/**
 * Produce MIDI files in specified folder.
 *
 * @author Todor Balabanov
 *
 * @email teodorgig@mail.ru
 *
 * @date 04 Jun 2008
 */
public class MidiFileProducer {
	/**
	 * Project subfolder for saving files.
	 */
	private static String MIDI_SAVE_FOLDER = "../midis/";

	/**
	 * Produce MIDI files in the specified internal application folder.
	 *
	 * @param melodies
	 *            Vector of melodies.
	 *
	 * @author Todor Balabanov
	 *
	 * @email teodorgig@mail.ru
	 *
	 * @date 04 Jun 2008
	 */
	public static void produce(Vector<Melody> melodies) {
		for (int i = 0; i < melodies.size(); i++) {
			String fileName = MIDI_SAVE_FOLDER;
			Melody melody = melodies.elementAt(i);

			fileName += melody.getId();
			fileName += "_";
			fileName += melody.getGenre();
			fileName += "_";
			fileName += melody.getTimber();
			fileName += "_";
			fileName += System.currentTimeMillis();
			fileName += ".mid";

			try {
```

```
                        BufferedWriter out = new BufferedWriter(new FileWriter(fileName));
                        // TODO Something is wrong with saving binary information.
                        out.write(melody.toMidiBytes());
                        out.close();
                    } catch (Exception ex) {
                        System.err.println(ex);
                    }
                }
            }
        }
```