Linux Multimedia Studio 1

Alexandre Hudon 9580433

Simon Symeonidis 5887887

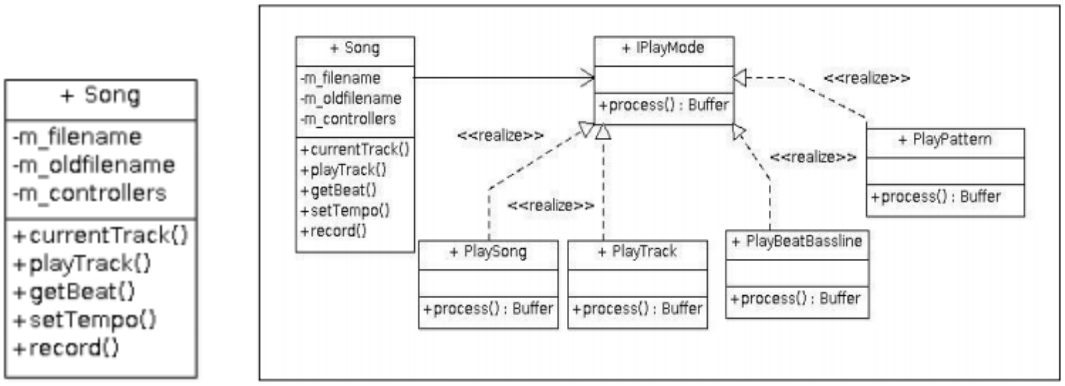
Due: April 7th, 11 marks

Refactorings

Refactoring 1: Strategy pattern for processing of play modes.
Patch pull request: https://github.com/LMMS/lmms/pull/575
Fork : https://github.com/AHudon/lmms.git

Recall from Milestone 3:

Before **After**



Patchset

Change 0/15: Adopt the strategy pattern for song processNextBuffer().

The following patchset provides an increase in cohesion by removing the responsibility of performing processing based on play mode from method processNextBuffer(…). Instead, since this responsibility differs based on the play mode but can be triggered from a common context, the adoption of the strategy pattern provides a mean to obtain the desired behavior based on the run-time type of the associated play mode.

Change 1/15: Extracting playMode switch statement and move to new method.
Link: https://github.com/AHudon/lmms/commit/fef355556ea0213b05470cc2593d6da5433517c8

The logic of play mode does not belong with the one of buffer processing. By extracting the code that deals with play modes to a new method, setPlayMode, the cohesion of method processNextBuffer increases and it is easier to understand what is meant by the switch statement responsible for performing the appropriate play mode.

```
80 src/core/song.cpp                                                    View
@@ -198,45 +198,7 @@ void song::processNextBuffer()
198 198         TrackList track_list;
199 199         int tco_num = -1;
200 200
201     -       switch( m_playMode )
202     -       {
203     -               case Mode_PlaySong:
204     -                       track_list = tracks();
205     -                       // at song-start we have to reset the LFOs
206     -                       if( m_playPos[Mode_PlaySong] == 0 )
207     -                       {
208     -                               EnvelopeAndLfoParameters::instances()->reset();
209     -                       }
210     -                       break;
211     -
212     -               case Mode_PlayTrack:
213     -                       track_list.push_back( m_trackToPlay );
214     -                       break;
215     -
216     -               case Mode_PlayBB:
217     -                       if( engine::getBBTrackContainer()->numOfBBs() > 0 )
218     -                       {
219     -                               tco_num = engine::getBBTrackContainer()->
220     -                                                                       currentBB();
221     -                               track_list.push_back( bbTrack::findBBTrack(
222     -                                                                       tco_num ) );
223     -                       }
224     -                       break;
225     -
226     -               case Mode_PlayPattern:
227     -                       if( m_patternToPlay != NULL )
228     -                       {
229     -                               tco_num = m_patternToPlay->getTrack()->
230     -                                               getTCONum( m_patternToPlay );
231     -                               track_list.push_back(
232     -                                               m_patternToPlay->getTrack() );
233     -                       }
234     -                       break;
235     -
236     -               default:
237     -                       return;
238     -
239     -       }
```

```diff
+       setPlayMode(m_playMode, track_list, tco_num);

        if( track_list.empty() == true )
        {
@@ -384,7 +346,47 @@ void song::processNextBuffer()
        }
 }

+void song::setPlayMode(PlayModes m_playMode, TrackList track_list, int tco_num){
+  switch( m_playMode )
+       {
+               case Mode_PlaySong:
+                       track_list = tracks();
+                       // at song-start we have to reset the LFOs
+                       if( m_playPos[Mode_PlaySong] == 0 )
+                       {
+                               EnvelopeAndLfoParameters::instances()->reset();
+                       }
+                       break;
+
+               case Mode_PlayTrack:
+                       track_list.push_back( m_trackToPlay );
+                       break;
+
+               case Mode_PlayBB:
+                       if( engine::getBBTrackContainer()->numOfBBs() > 0 )
+                       {
+                               tco_num = engine::getBBTrackContainer()->
+                                                               currentBB();
+                               track_list.push_back( bbTrack::findBBTrack(
+                                                               tco_num ) );
+                       }
+                       break;

+               case Mode_PlayPattern:
+                       if( m_patternToPlay != NULL )
+                       {
+                               tco_num = m_patternToPlay->getTrack()->
+                                               getTCONum( m_patternToPlay );
+                               track_list.push_back(
+                                               m_patternToPlay->getTrack() );
+                       }
+                       break;
+
+               default:
+                       return;
+
+       }
+}
```

Change 2/15: Creating interface IPlayMode
Link: https://github.com/AHudon/lmms/commit/1db3224d86bf4a2d660ac6c5d028ed6198d16ee4

The interface IPlayMode is necessary as it represents the declared types of its children that are implementing the method process().

33 include/iplaymode.h View

```
@@ -0,0 +1,33 @@
+/*
+ * iplaymode.h - an interface for the different play modes.
+ *
+ * Copyright (c) 2008-2009 Tobias Doerffel <tobydox/at/users.sourceforge.net>
+ *
+ * This file is part of Linux MultiMedia Studio - http://lmms.sourceforge.net
+ *
+ * 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 (see COPYING); if not, write to the
+ * Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
+ * Boston, MA 02110-1301 USA.
+ *
+ */
+
+
+#ifndef IPLAYMODE_H
+#define IPLAYMODE_H
+
+class IPlayMode
+{
+};
+
+#endif // IPLAYMODE_H
```

Change 3/15: Adding behavior definition to IPlayMode
Link: https://github.com/AHudon/lmms/commit/fe14b52ee354952510c3eaad063e651351c23ba9

Method process(…), declared as virtual, needs to be implemented by all the subclasses to ensure that they all have their own strategy/version of the process method. The process invoked will be based on the dynamic type of the selected play mode.

5 include/iplaymode.h View

```diff
@@ -28,6 +28,11 @@

 class IPlayMode
 {
+public:
+    virtual ~IPlayMode(){}
+    virtual void process(PlayModes, TrackList,int) = 0;
 };

+
+
 #endif // IPLAYMODE_H
```

Change 4/15: Creating subclass PlaySong
Link: https://github.com/AHudon/lmms/commit/3c4bc304c8794a91989e6416a3b484ab6b7fe8dc

The first play mode of the switch statement is the one responsible for playing the song. The class PlaySong must inherit the method process from its parent playmode.

27 include/PlaySong.h View

```
@@ -0,0 +1,27 @@
+/*
+    Copyright 2014 <copyright holder> <email>
+
+    Licensed under the Apache License, Version 2.0 (the "License");
+    you may not use this file except in compliance with the License.
+    You may obtain a copy of the License at
+
+        http://www.apache.org/licenses/LICENSE-2.0
+
+    Unless required by applicable law or agreed to in writing, software
+    distributed under the License is distributed on an "AS IS" BASIS,
+    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
+    See the License for the specific language governing permissions and
+    limitations under the License.
+*/
+
+
+#ifndef PLAYSONG_H
+#define PLAYSONG_H
+#include <iplaymode.h>
+
+class PlaySong : public IPlayMode
+{
+  void process();
+};
+
+#endif // PLAYSONG_H
```

22 src/core/PlaySong.cpp View

```
@@ -0,0 +1,22 @@
+/*
+    Copyright 2014 <copyright holder> <email>
+
+    Licensed under the Apache License, Version 2.0 (the "License");
+    you may not use this file except in compliance with the License.
+    You may obtain a copy of the License at
+
+        http://www.apache.org/licenses/LICENSE-2.0
+
+    Unless required by applicable law or agreed to in writing, software
+    distributed under the License is distributed on an "AS IS" BASIS,
+    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
+    See the License for the specific language governing permissions and
+    limitations under the License.
+*/
+
+
+#include "PlaySong.h"
+
+void PlaySong::process(){
+
+}
```

Change 5/15: Make track list and member variable of song
Link: https://github.com/AHudon/lmms/commit/5be4955701ec7751be0a3e86ec4ea8b2f72b62c5

One of the identified problems with the previous implementation of song.cpp is that variable track_list needs to be accessed and changed by classes coupled to song.cpp. This can be done if track list a

member variable of class song.cpp rather than created locally in method of class song.cpp and then passed off to remote invoking classes.

```
8 include/song.h                                                    View
@@ -94,8 +94,8 @@ class EXPORT song : public TrackContainer
 94  94
 95  95
 96  96         void processNextBuffer();
 97     -       void setPlayMode(PlayModes, TrackList, int);
 98     -
        97 +       void setPlayMode(PlayModes, int);
        98 +       TrackList getTrackList();
 99  99         inline int getMilliseconds() const
100 100         {
101 101                 return m_elapsedMilliSeconds;
@@ -299,7 +299,7 @@ class EXPORT song : public TrackContainer
299 299         song();
300 300         song( const song & );
301 301         virtual ~song();
302     -
       302 +       TrackList m_tracklist;
303 303
304 304         inline tact_t currentTact() const
305 305         {
@@ -358,7 +358,7 @@ class EXPORT song : public TrackContainer
358 358         friend class SongEditor;
359 359         friend class mainWindow;
360 360         friend class ControllerRackView;
361     -
       361 +       friend class PlaySong;
362 362  signals:
363 363         void projectLoaded();
364 364         void playbackStateChanged();
```

Song.cpp (showing only the new accessor):

```
207 +TrackContainer::TrackList song::getTrackList(){
208 +
209 +  return m_tracklist;
210 +}
211 +void song::setPlayMode(PlayModes m_playMode, int tco_num){
```

Change 6/15: Moving PlaySong logic from setPlayMode to PlaySong.cpp
Link: https://github.com/AHudon/lmms/commit/5be4955701ec7751be0a3e86ec4ea8b2f72b62c5

The logic of playsong is now moved to class playsong. Instead of directly executing the behavior from playsong, it will now be done via an instance of PlaySong by calling process(…). This way, the logic of playsong is in its own unit rather than spread inside method setPlayMode(…).

45 src/core/PlaySong.cpp View

```diff
@@ -1,22 +1,37 @@
 /*
-    Copyright 2014 <copyright holder> <email>
-
-    Licensed under the Apache License, Version 2.0 (the "License");
-    you may not use this file except in compliance with the License.
-    You may obtain a copy of the License at
-
-        http://www.apache.org/licenses/LICENSE-2.0
-
-    Unless required by applicable law or agreed to in writing, software
-    distributed under the License is distributed on an "AS IS" BASIS,
-    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
-    See the License for the specific language governing permissions and
-    limitations under the License.
-*/
+ * PlaySong.cpp - PlaySong strategy.
+ *
+ * Copyright (c) 2008-2009 Tobias Doerffel <tobydox/at/users.sourceforge.net>
+ *
+ * This file is part of Linux MultiMedia Studio - http://lmms.sourceforge.net
+ *
+ * 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 (see COPYING); if not, write to the
+ * Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
+ * Boston, MA 02110-1301 USA.
+ *
+ */


 #include "PlaySong.h"
+#include "TrackContainer.h"
+#include "EnvelopeAndLfoParameters.h"

-void PlaySong::process(){
+   IPlayMode::TrackList PlaySong::process(song *const aSong){
+
+  const TrackList track_list = aSong->tracks();
+
+  EnvelopeAndLfoParameters::instances()->reset();

+  return track_list;
 }
```

Change 7/15: Creating class PlayTrack
Link: https://github.com/AHudon/lmms/commit/a395c308d23aa0e5520e679e03d5dc053d52766a

The second play mode of the switch statement is the one responsible for playing a track. The class PlayTack must inherit the method process from its parent playmode (see next change).

35 include/PlayTrack.h View

```
@@ -0,0 +1,35 @@
+/*
+ * PlayTrack.h - PlayTrack strategy.
+ *
+ * Copyright (c) 2008-2009 Tobias Doerffel <tobydox/at/users.sourceforge.net>
+ *
+ * This file is part of Linux MultiMedia Studio - http://lmms.sourceforge.net
+ *
+ * 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 (see COPYING); if not, write to the
+ * Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
+ * Boston, MA 02110-1301 USA.
+ *
+ */
+
+
+
+#ifndef PLAYTRACK_H
+#define PLAYTRACK_H
+#include <iplaymode.h>
+
+class PlayTrack
+{
+};
+
+#endif // PLAYTRACK_H
```

Change 8/15: Accessor for track to play
Link: https://github.com/AHudon/lmms/commit/4b5d76ab9467bc10fd6d7713cba53994dc1747dd

PlayTrack requires access to member variable track to play defined in class song. Considering PlayTrack is given access to the contextual song, an accessor to retrieve member variable tracktoplay is added to class song.

2 include/song.h View

```diff
@@ -96,6 +96,8 @@ class EXPORT song : public TrackContainer
 96  96         void processNextBuffer();
 97  97         void setPlayMode(PlayModes, int);
 98  98         TrackList getTrackList();
     99 +       track* getTrackToPlay();
    100 +
 99 101         inline int getMilliseconds() const
100 102         {
101 103                 return m_elapsedMilliSeconds;
```

10 src/core/song.cpp View

```diff
@@ -196,18 +196,22 @@ void song::processNextBuffer()
196 196         {
197 197                 return;
198 198         }
199     -
200 199
201 200         int tco_num = -1;
202 201         setPlayMode(m_playMode, tco_num);
203     -
204     -
205 202  }
206 203
    204 +
    205 +
207 206  TrackContainer::TrackList song::getTrackList(){
208 207
209 208    return m_tracklist;
210 209  }
    210 +
    211 +track * song::getTrackToPlay(){
    212 +  return m_trackToPlay;
    213 +}
    214 +
211 215  void song::setPlayMode(PlayModes m_playMode, int tco_num){
212 216    switch( m_playMode )
213 217        {
```

Change 9/15: Moving PlayTrack logic from setPlayMode to PlayTrack.cpp
Link: https://github.com/AHudon/lmms/commit/d1b6c6a36e57674ebfbf1f5e1bd07126d3ca5e0a

The logic of playtrack is now moved to class playtrack. Instead of directly executing the behavior from playtrack, it will now be done via an instance of PlayTrack by calling process(…). This way, the logic of playtrack is in its own unit rather than spread inside method setPlayMode(…).

1 src/core/PlayTrack.cpp View

```diff
@@ -30,6 +30,7 @@
 30  30
 31  31    const TrackList track_list = aSong->tracks();
 32  32
     33 +  aSong->m_tracklist.push_back(aSong->m_trackToPlay);
 33  34
 34  35    return track_list;
 35  36  }
```

9 src/core/song.cpp View

```diff
@@ -53,6 +53,7 @@
 #include "NotePlayHandle.h"
 #include "pattern.h"
 #include "PlaySong.h"
+#include "PlayTrack.h"
 #include "PianoRoll.h"
 #include "ProjectJournal.h"
 #include "project_notes.h"
@@ -213,20 +214,24 @@ track * song::getTrackToPlay(){
 }

 void song::setPlayMode(PlayModes m_playMode, int tco_num){
+  PlaySong *ps = new PlaySong;
+  PlayTrack *aPlayTrack = new PlayTrack;
   switch( m_playMode )
        {
+
                case Mode_PlaySong:

                        // at song-start we have to reset the LFOs
                        if( m_playPos[Mode_PlaySong] == 0 )
                        {
-                               PlaySong *ps = new PlaySong;
+
                                m_tracklist = ps->process(this);
                        }
                        break;

                case Mode_PlayTrack:
-                       m_tracklist.push_back( m_trackToPlay );
+
+                       aPlayTrack->process(this);
                        break;

                case Mode_PlayBB:
```

Change 10/15: Creating class PlayBB
Link: https://github.com/AHudon/lmms/commit/a8125f60ddbe06fc9263f081e70acdce035f2a94

The third play mode of the switch statement is the one responsible for Mode_PlayBB. The class PlayBB must inherit the method process from its parent playmode (see next change).

33 include/PlayBB.h View

```
@@ -0,0 +1,33 @@
+/*
+ * PlayBB.h - PlayBB strategy.
+ *
+ * Copyright (c) 2008-2009 Tobias Doerffel <tobydox/at/users.sourceforge.net>
+ *
+ * This file is part of Linux MultiMedia Studio - http://lmms.sourceforge.net
+ *
+ * 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 (see COPYING); if not, write to the
+ * Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
+ * Boston, MA 02110-1301 USA.
+ *
+ */
+
+
+#ifndef PLAYBB_H
+#define PLAYBB_H
+
+class PlayBB
+{
+};
+
+#endif // PLAYBB_H
```

Change 11/15: Moving PlayBB logic from setPlayMode to PlayBB.cpp
Link: https://github.com/AHudon/lmms/commit/a4121111c0e728ad43642f421900ce4431ce3ada

The logic of playbb is now moved to class playbb. Instead of directly executing the behavior from playbb, it will now be done via an instance of PlayBB by calling process(…). This way, the logic of playbb is in its own unit rather than spread inside method setPlayMode(…).

15 src/core/PlayBB.cpp View

```
@@ -24,4 +24,19 @@


 #include "PlayBB.h"
+#include "TrackContainer.h"
+#include "bb_track.h"
+#include "bb_track_container.h"

+   IPlayMode::TrackList PlayBB::process(song *const aSong){
+
+  if( engine::getBBTrackContainer()->numOfBBs() > 0 )
+                        {
+                                int tco_num = engine::getBBTrackContainer()->
+                                                                        currentBB();
+                                aSong->m_tracklist.push_back( bbTrack::findBBTrack(
+                                                                        tco_num ) );
+                        }
+
+  return aSong->m_tracklist;
+}
```

```
236 233
237 234                 case Mode_PlayBB:
238     -                       if( engine::getBBTrackContainer()->numOfBBs() > 0 )
239     -                       {
240     -                               tco_num = engine::getBBTrackContainer()->
241     -                                                               currentBB();
242     -                               m_tracklist.push_back( bbTrack::findBBTrack(
243     -                                                               tco_num ) );
244     -                       }
    235 +                       aPlayBB->process(this);
245 236                         break;
246 237
247 238                 case Mode_PlayPattern:
```

Change 12/15: Creating class PlayPattern
Link: https://github.com/AHudon/lmms/commit/26ea9493eb6272a5375d87ccab021e593e6dc392

The last play mode of the switch statement is the one responsible for playing a particular pattern. The class PlayPlattern must inherit the method process from its parent playmode (see next change).

```
33 include/PlayPattern.h                                                View
... ... @@ -0,0 +1,33 @@
  1 +/*
  2 + * PlayPattern.h - PlayPattern strategy.
  3 + *
  4 + * Copyright (c) 2008-2009 Tobias Doerffel <tobydox/at/users.sourceforge.net>
  5 + *
  6 + * This file is part of Linux MultiMedia Studio - http://lmms.sourceforge.net
  7 + *
  8 + * This program is free software; you can redistribute it and/or
  9 + * modify it under the terms of the GNU General Public
 10 + * License as published by the Free Software Foundation; either
 11 + * version 2 of the License, or (at your option) any later version.
 12 + *
 13 + * This program is distributed in the hope that it will be useful,
 14 + * but WITHOUT ANY WARRANTY; without even the implied warranty of
 15 + * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 16 + * General Public License for more details.
 17 + *
 18 + * You should have received a copy of the GNU General Public
 19 + * License along with this program (see COPYING); if not, write to the
 20 + * Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
 21 + * Boston, MA 02110-1301 USA.
 22 + *
 23 + */
 24 +
 25 +
 26 +#ifndef PLAYPATTERN_H
 27 +#define PLAYPATTERN_H
 28 +
 29 +class PlayPattern
 30 +{
 31 +};
 32 +
 33 +#endif // PLAYPATTERN_H
```

Change 13/15: Moving PlayPattern logic from setPlayMode to PlayPattern.cpp
Link:

https://github.com/AHudon/lmms/commit/f90c820707ff2a815b3e8bbee5cbbb54b12502cc

The logic of playpattern is now moved to class playpattern. Instead of directly executing the behavior from playpattern, it will now be done via an instance of PlayPattern by calling process(…). This way, the logic of playpattern is in its own unit rather than spread inside method setPlayMode(…).

16 src/core/PlayPattern.cpp View

```diff
@@ -24,4 +24,20 @@


 #include "PlayPattern.h"
+#include "TrackContainer.h"
+#include "track.h"
+#include "pattern.h"

+
+  IPlayMode::TrackList PlayPattern::process(song *const aSong){
+
+                        if( aSong->m_patternToPlay != NULL )
+                        {
+                                aSong->m_patternToPlay->getTrack()->
+                                                getTCONum( aSong->m_patternToPlay );
+                                aSong->m_tracklist.push_back(
+                                                aSong->m_patternToPlay->getTrack() );
+                        }
+
+  return aSong->m_tracklist;
+}
```

12 src/core/song.cpp View

```diff
@@ -52,9 +52,10 @@
 #include "DataFile.h"
 #include "NotePlayHandle.h"
 #include "pattern.h"
+#include "PlayBB.h"
+#include "PlayPattern.h"
 #include "PlaySong.h"
 #include "PlayTrack.h"
-#include "PlayBB.h"
 #include "PianoRoll.h"
 #include "ProjectJournal.h"
 #include "project_notes.h"
@@ -218,6 +219,7 @@ void song::setPlayMode(PlayModes m_playMode, int tco_num){
   PlaySong *ps = new PlaySong;
   PlayTrack *aPlayTrack = new PlayTrack;
   PlayBB *aPlayBB = new PlayBB;
+  PlayPattern *aPlayPattern = new PlayPattern;
   switch( m_playMode )
        {

@@ -236,13 +238,7 @@ void song::setPlayMode(PlayModes m_playMode, int tco_num){
                        break;

                case Mode_PlayPattern:
-                       if( m_patternToPlay != NULL )
-                       {
-                               tco_num = m_patternToPlay->getTrack()->
-                                               getTCONum( m_patternToPlay );
-                               m_tracklist.push_back(
-                                               m_patternToPlay->getTrack() );
-                       }
+                       aPlayPattern->process(this);
                        break;

                default:
```

Change 14/15: Modify return type of process to void.
Link: https://github.com/AHudon/lmms/commit/06a112b3c427b7df2133b7b898c29e62b86ccbec

Since track list is updated directly from the context (song), there is no need for a return type as modifications will be directly applied on the song.

2 include/PlayBB.h View

```
@@ -30,7 +30,7 @@
 class PlayBB : public IPlayMode
 {
 public:
-  TrackList process(song *const);
+  void process(song *const);
 };

 #endif // PLAYBB_H
```

2 include/PlayPattern.h View

```
@@ -30,7 +30,7 @@
 class PlayPattern : public IPlayMode
 {
 public:
-  TrackList process(song *const);
+  void process(song *const);
 };

 #endif // PLAYPATTERN_H
```

2 include/PlayTrack.h View

```
@@ -31,7 +31,7 @@
 class PlayTrack : public IPlayMode
 {
 public:
-  TrackList process(song *const);
+  void process(song *const);
 };

 #endif // PLAYTRACK_H
```

2 include/iplaymode.h View

```diff
@@ -33,7 +33,7 @@ class IPlayMode
 public:
     typedef QVector<track *> TrackList;
     virtual ~IPlayMode(){}
-    virtual TrackList process(song *const) = 0;
+    virtual void process(song *const) = 0;
 };
```

4 src/core/PlayBB.cpp View

```diff
@@ -28,7 +28,7 @@
 #include "bb_track.h"
 #include "bb_track_container.h"

-   IPlayMode::TrackList PlayBB::process(song *const aSong){
+   void PlayBB::process(song *const aSong){

   if( engine::getBBTrackContainer()->numOfBBs() > 0 )
                        {
@@ -37,6 +37,4 @@
                                aSong->m_tracklist.push_back( bbTrack::findBBTrack(
                                                                tco_num ) );
                        }
-
-  return aSong->m_tracklist;
 }
```

3 src/core/PlayPattern.cpp View

```diff
@@ -29,7 +29,7 @@
 #include "pattern.h"


-IPlayMode::TrackList PlayPattern::process(song *const aSong){
+void PlayPattern::process(song *const aSong){

                        if( aSong->m_patternToPlay != NULL )
                        {
@@ -39,5 +39,4 @@
                                                aSong->m_patternToPlay->getTrack() );
                        }

-return aSong->m_tracklist;
 }
```

3 src/core/PlaySong.cpp View

```diff
@@ -28,12 +28,11 @@
 #include "EnvelopeAndLfoParameters.h"


-IPlayMode::TrackList PlaySong::process(song *const aSong){
+void PlaySong::process(song *const aSong){

  aSong->m_tracklist = aSong->tracks();
  if( aSong->m_playPos[Mode_PlaySong] == 0 )
                        {
  EnvelopeAndLfoParameters::instances()->reset();
                        }
-return aSong->m_tracklist;
 }
```

2 src/core/PlaySong.h View

```diff
@@ -31,7 +31,7 @@
 class PlaySong : public IPlayMode
 {
 public:
-TrackList process(song *const);
+void process(song *const);
```

Change 15/15: Update the playmode switch statements to assign dynamically strategies
Link: https://github.com/AHudon/lmms/commit/4ffd15202258bd7067d09e3691d28b92adc41aff

This change dynamically assigns a type to a variable of static type IPlaymode at run-time based on the desired play mode. This reduces the number of objects that needs to be created inside the switch statement.

28 src/core/song.cpp View

```diff
@@ -216,41 +216,41 @@ track * song::getTrackToPlay(){
 }

 void song::setPlayMode(PlayModes m_playMode, int tco_num){
-  PlaySong *ps = new PlaySong;
-  PlayTrack *aPlayTrack = new PlayTrack;
-  PlayBB *aPlayBB = new PlayBB;
-  PlayPattern *aPlayPattern = new PlayPattern;
+
+  IPlayMode * pm;
+
   switch( m_playMode )
        {

                case Mode_PlaySong:
-
-                       ps->process(this);
-                       break;
+                 pm = new PlaySong;
+                 break;

                case Mode_PlayTrack:
-
-                       aPlayTrack->process(this);
-                       break;
+                 pm = new PlayTrack;
+                 break;

                case Mode_PlayBB:
-                       aPlayBB->process(this);
-                       break;
+                 pm = new PlayBB;
+                 break;

                case Mode_PlayPattern:
-                       aPlayPattern->process(this);
-                       break;
+                 pm = new PlayPattern;
+                 break;

                default:
                        return;

        }

+       pm->process(this);
+
        if( m_tracklist.empty() == true )
        {
                return;
        }

+
```

Before

```
198         TrackList track_list;
199         int tco_num = -1;
200
201         switch( m_playMode )
202         {
203                 case Mode_PlaySong:
204                         track_list = tracks();
205                         // at song-start we have to reset the LFOs
206                         if( m_playPos[Mode_PlaySong] == 0 )
207                         {
208                                 EnvelopeAndLfoParameters::instances()->reset();
209                         }
210                         break;
211
212                 case Mode_PlayTrack:
213                         track_list.push_back( m_trackToPlay );
214                         break;
215
216                 case Mode_PlayBB:
217                         if( engine::getBBTrackContainer()->numOfBBs() > 0 )
218                         {
219                                 tco_num = engine::getBBTrackContainer()->
220                                                                 currentBB();
221                                 track_list.push_back( bbTrack::findBBTrack(
222                                                                 tco_num ) );
223                         }
224                         break;
225
226                 case Mode_PlayPattern:
227                         if( m_patternToPlay != NULL )
228                         {
229                                 tco_num = m_patternToPlay->getTrack()->
230                                                 getTCONum( m_patternToPlay );
231                                 track_list.push_back(
232                                                 m_patternToPlay->getTrack() );
233                         }
234                         break;
235
236                 default:
237                         return;
238
239         }
```

After

```
void song::setPlayMode(PlayModes m_playMode, int tco_num){

  IPlayMode * pm;

  switch( m_playMode )
        {

                case Mode_PlaySong:
                  pm = new PlaySong;
                  break;

                case Mode_PlayTrack:
                  pm = new PlayTrack;
                  break;

                case Mode_PlayBB:
                  pm = new PlayBB;
                  break;

                case Mode_PlayPattern:
                  pm = new PlayPattern;
                  break;

                default:
                        return;

        }

        pm->process(this);
```

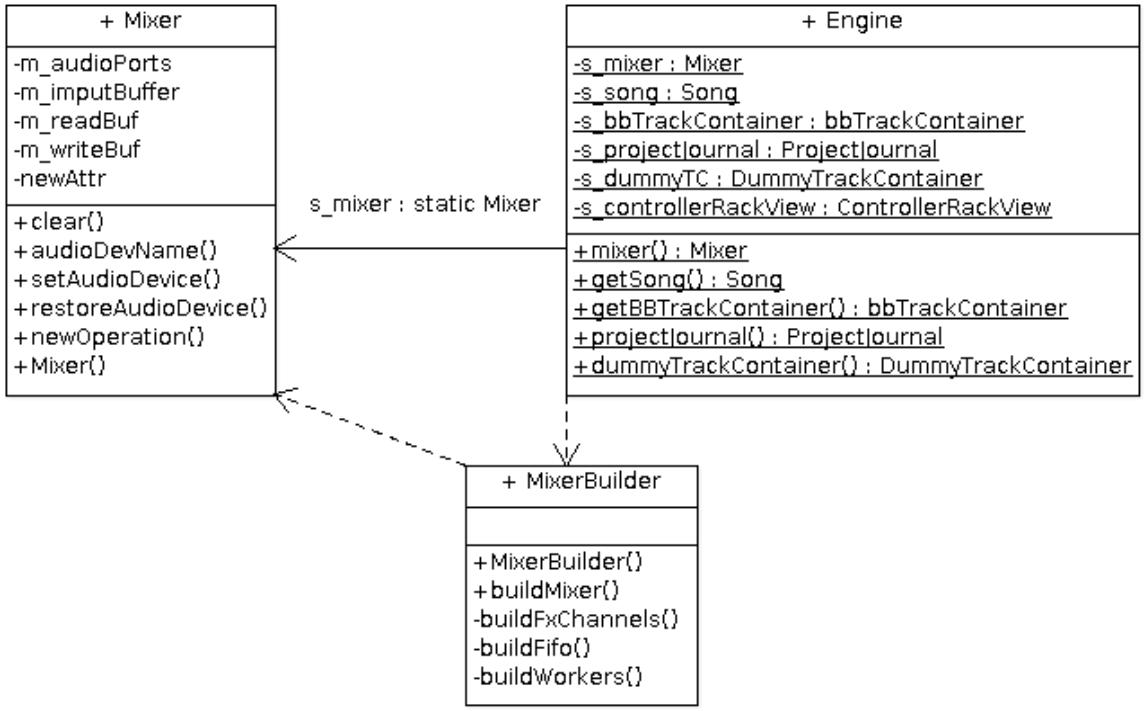
Note:

processNextBuffer(..) decreased in size. However, it could be refactored further more as it encompasses several other responsibilities such as frame processing.

Refactoring 2: move fabrication logic to a builder.

Patch Pull Request: (see patch # on Fork)
Fork: https://github.com/psyomn/lmms.git

Recall from the previous milestone the following diagram, that hints the required refactoring.



Originally, the body of the constructor of mixer had all the fabrication logic, and was rather complex - it ranged much more than just initializing different values. It had to check also if the current session was being run with a GUI or not.

The builder aims to remove this logic, and create the object with the proper configurations, while hiding everything from the programmer.

Change 1/8

Link : https://github.com/psyomn/lmms/commit/0925fc2186a566db9286e5652e46ad6a491993e2

First, we notice that some instance variables are not being initialized properly like the rest. These are pushed to the top in oder to be initialized normaly.

```
6 src/core/Mixer.cpp
@@ -303,6 +303,9 @@ void MixerWorkerThread::processJobQueue()
303 303         m_inputBufferWrite( 1 ),
304 304         m_readBuf( NULL ),
305 305         m_writeBuf( NULL ),
    306 +       m_readBuffer( 0 ),
    307 +       m_writeBuffer( 1 ),
    308 +       m_poolDepth( 2 ),
306 309         m_cpuLoad( 0 ),
307 310         m_workers(),
308 311         m_numWorkers( QThread::idealThreadCount()-1 ),
@@ -380,9 +383,6 @@ void MixerWorkerThread::processJobQueue()
380 383                 m_workers.push_back( wt );
381 384         }
382 385
383     -       m_poolDepth = 2;
384     -       m_readBuffer = 0;
385     -       m_writeBuffer = 1;
386 386  }
387 387
388 388
```

Change 2/8

Link : https://github.com/psyomn/lmms/commit/08b31f0f268492aecee167a60811b8e28d1820c2

Rename a global variable to proper capitalized convention. It was actually confusing to understand where the variable was coming from in a non-global sort of falsified sense, due to its name being camel cased, rather than the proper way (eg: GLOBAL_VARIABLE).

```diff
6 include/FxMixer.h
@@ -31,7 +31,7 @@
 #include "JournallingObject.h"


-const int NumFxChannels = 64;
+const int NUM_FX_CHANNELS = 64;


 struct FxChannel
@@ -79,7 +79,7 @@ class FxMixer : public JournallingObject, public

        FxChannel * effectChannel( int _ch )
        {
-               if( _ch >= 0 && _ch <= NumFxChannels )
+               if( _ch >= 0 && _ch <= NUM_FX_CHANNELS)
                {
                        return m_fxChannels[_ch];
                }
@@ -88,7 +88,7 @@ class FxMixer : public JournallingObject, public


 private:
-       FxChannel * m_fxChannels[NumFxChannels+1];      // +1 = ma
+       FxChannel * m_fxChannels[ NUM_FX_CHANNELS + 1 ];        //

```

Change 3/ 8

Link : https://github.com/psyomn/lmms/commit/2a9c3116663980d99b8e187f7e074611efc2e834

Add stubs of the MixerBuilder, and make sure that the configuration of the project sees these files and compiles. Since CMake was used, and globbing for source files, we had to update the CMakeCache.txt by reconfiguring CMake whenever a new source was added.

```
+class MixerBuilder
+{
+public:
+       MixerBuilder(){}
+       ~MixerBuilder(){}
+
+       Mixer* build();
+
+private:
+
+  void buildFrames();
+
+       void buildFxChannels();
+
+       void buildFifo();
+
+  /* Pointer to the mixer instance we're building */
+       Mixer* p_mixer;
+};
+
+#endif /* __MIXER_BUILDER_H__ */
```

```
+Mixer* MixerBuilder::build()
+{
+       Mixer* mixer = new Mixer();
+
+  buildFrames();
+       buildFxChannels();
+       buildFifo();
+
+       return mixer;
+}
+
+/** Build the frames */
+void MixerBuilder::buildFrames()
+{
+}
+
+/** Build the fx channels */
+void MixerBuilder::buildFxChannels()
+{
+}
+
+/** Build the fifo stuff */
+void MixerBuilder::buildFifo()
+{
+}
```

Change 4/8

Link : https://github.com/psyomn/lmms/commit/f53c18ff08cc90f0c0b165ac862557b0aec7d6cf

Move fabrication logic to the builder, since the builder should be the expert in creating the mixer, and not the mixer itself.

```
-       else if( configManager::inst()->value( "mixer", "framesperaudiobuffer"
-                                               ).toInt() >= 32 )
-       {
-               m_framesPerPeriod =
-                       (fpp_t) configManager::inst()->value( "mixer",
-                                       "framesperaudiobuffer" ).toInt();
-
-               if( m_framesPerPeriod > DEFAULT_BUFFER_SIZE )
-               {
-                       m_fifo = new fifo( m_framesPerPeriod
-                                                       / DEFAULT_BUFFER_SIZE );
-                       m_framesPerPeriod = DEFAULT_BUFFER_SIZE;
-               }
-               else
-               {
-                       m_fifo = new fifo( 1 );
-               }
-       }
-       else
-       {
-               configManager::inst()->setValue( "mixer",
-                                                       "framesperaudiobuffer",
-                               QString::number( m_framesPerPeriod ) );
-               m_fifo = new fifo( 1 );
-       }
-
        m_workingBuf = (sampleFrame*) aligned_malloc( m_framesPerPeriod *
```

```
+       for( int i = 0; i < 2; ++i )
+       {
+               m_mixer->m_inputBufferFrames[i] = 0;
+               m_mixer->m_inputBufferSize[i] = DEFAULT_BUFFER_SIZE * 100;
+               m_mixer->m_inputBuffer[i] = new sampleFrame[ DEFAULT_BUFFER_SIZE * 100 ];
+
+               m_mixer->clearAudioBuffer(
+                       m_mixer->m_inputBuffer[i],
+                       m_mixer->m_inputBufferSize[i] );
+       }
 }

 /** Build the fx channels */
 void MixerBuilder::buildFxChannels()
 {
+       for( int i = 1; i < NUM_FX_CHANNELS + 1; ++i )
+       {
+               __fx_channel_jobs[i-1] = (fx_ch_t) i;
+       }
 }
```

```
 /** Build the fifo stuff */
 void MixerBuilder::buildFifo()
 {
+       if ( !engine::hasGUI() )
+       {
+               m_mixer->m_framesPerPeriod = DEFAULT_BUFFER_SIZE;
+               m_mixer->m_fifo = new fifo( 1 );
+       }
+       else if( configManager::inst()->value( "mixer", "framesperaudiobuffer"
+                                               ).toInt() >= 32 )
+       {
+               m_mixer->m_framesPerPeriod =
+                       (fpp_t) configManager::inst()->value( "mixer",
+                                       "framesperaudiobuffer" ).toInt();
+
+               if( m_mixer->m_framesPerPeriod > DEFAULT_BUFFER_SIZE )
+               {
+                       m_mixer->m_fifo = new fifo( m_mixer->m_framesPerPeriod
+                                                       / DEFAULT_BUFFER_SIZE );
+                       m_mixer->m_framesPerPeriod = DEFAULT_BUFFER_SIZE;
+               }
+               else
+               {
+                       m_mixer->m_fifo = new fifo( 1 );
+               }
+       }
+       else
+       {
+               configManager::inst()->setValue( "mixer",
+                                                       "framesperaudiobuffer",
+                               QString::number( m_mixer->m_framesPerPeriod ) );
+               m_mixer->m_fifo = new fifo( 1 );
+       }
 }
+
```

Change 5/8

Link : https://github.com/psyomn/lmms/commit/3a6b280b82298cf90ae20818a6c8888ad977437b

Some of the functions in MixerBuilder require the use of a free function in Mixer.cpp. This function is something that allocates alligned memory for the Mixer to use for its buffers. In order to get this working for the Builder, we need to separate it into another class. This also means that it can be reused in the future by other components as well. We create a new class called MemoryHelper, and move the free function in there.

```
-static void aligned_free( void * _buf )
-{
-       if( _buf != NULL )
-       {
-               int *ptr2=(int *)_buf - 1;
-               _buf = (char *)_buf- *ptr2;
-               free(_buf);
-       }
-}
-
-static void * aligned_malloc( int _bytes )
-{
-       char *ptr,*ptr2,*aligned_ptr;
-       int align_mask = ALIGN_SIZE- 1;
-       ptr=(char *)malloc(_bytes +ALIGN_SIZE+ sizeof(int));
-       if(ptr==NULL) return(NULL);
-
-       ptr2 = ptr + sizeof(int);
-       aligned_ptr = ptr2 + (ALIGN_SIZE- ((size_t)ptr2 & align_mask));
-
-
-       ptr2 = aligned_ptr - sizeof(int);
-       *((int *)ptr2)=(int)(aligned_ptr - ptr);
-
-       return(aligned_ptr);
-}
-
-
-
```

```
+#ifndef __MEMORY_HELPER_H__
+#define __MEMORY_HELPER_H__
+
+/** aligned malloc functionality for any other class to use */
+class MemoryHelper
+{
+public:
+  static void* alignedMalloc(int);
+       static void alignedFree(void*);
+private:
+  /* Don't instantiate me */
+  MemoryHelper(){};
+}
+
+#endif /* __MEMORY_HELPER_H__ */
```

```
+void* MemoryHelper::alignedMalloc(int _bytes)
+{
+       char* ptr;
+       char* ptr2;
+       char* aligned_ptr;
+       int align_mask = ALIGN_SIZE - 1;
+
+       ptr = (char *) malloc( _bytes + ALIGN_SIZE + sizeof(int) );
+       if ( ptr==NULL ) return(NULL);
+
+       ptr2 = ptr + sizeof(int);
+       aligned_ptr = ptr2 + (ALIGN_SIZE- ((size_t)ptr2 & align_mask));
+
+       ptr2 = aligned_ptr - sizeof(int);
+       *((int *)ptr2) = (int) (aligned_ptr - ptr);
+
+       return(aligned_ptr);
+}
+
+void MemoryHelper::alignedFree(void* _buf)
+{
+       if( _buf != NULL )
+       {
+               int *ptr2=(int *)_buf - 1;
+               _buf = (char *)_buf - *ptr2;
+               free(_buf);
+       }
+}
```

Change 6/8

Link : https://github.com/psyomn/lmms/commit/5249623f795e902fb34d0f54d90c51d80a0ba495

Finally we move the last part of the fabrication into the Builder, from the Mixer. This goes to the frame

```
 {
-       m_workingBuf = (sampleFrame*)
-               MemoryHelper::alignedMalloc( m_framesPerPeriod * sizeof( sampleFrame ) );
-
-       for( int i = 0; i < 3; i++ )
-       {
-               m_readBuf = (surroundSampleFrame*)
-                       MemoryHelper::alignedMalloc( m_framesPerPeriod *
-                               sizeof( surroundSampleFrame ) );
-
-               clearAudioBuffer( m_readBuf, m_framesPerPeriod );
-               m_bufferPool.push_back( m_readBuf );
-       }
-
        for( int i = 0; i < m_numWorkers+1; ++i )
        {
                MixerWorkerThread * wt = new MixerWorkerThread( i, this );
@@ -308,7 +295,6 @@ void MixerWorkerThread::processJobQueue()
                }
                m_workers.push_back( wt );
        }
```

building part of the MixerBuilder.

```
+#include "MemoryHelper.h"
 #include "config_mgr.h"
 #include "engine.h"

@@ -58,6 +59,22 @@ void MixerBuilder::buildFrames()
                        m_mixer->m_inputBuffer[i],
                        m_mixer->m_inputBufferSize[i] );
        }
+
+       m_mixer->m_workingBuf = (sampleFrame*)
+               MemoryHelper::alignedMalloc(
+                 m_mixer->m_framesPerPeriod * sizeof( sampleFrame ) );
+
+       for( int i = 0; i < 3; i++ )
+       {
+               m_mixer->m_readBuf = (surroundSampleFrame*)
+                       MemoryHelper::alignedMalloc( m_mixer->m_framesPerPeriod *
+                               sizeof( surroundSampleFrame ) );
+
+               m_mixer->clearAudioBuffer( m_mixer->m_readBuf, m_mixer->m_framesPerPeriod );
+               m_mixer->m_bufferPool.push_back( m_mixer->m_readBuf );
+       }
+
+
 }
```

Change 7/8

Link : https://github.com/psyomn/lmms/commit/49cdf475bfb68a9cfbcd1412e8038b11aa97c693

We are also able to break down the building of the FiFo channels using an interfacing method 'buildFifo', which checks the gui status, and redirects the call to 'buildHeadlessFifo' if there is none. This is a valid way of hiding the construction of the Mixer from potential users of the class.

```
2 include/MixerBuilder.h
@@ -48,6 +48,8 @@ class MixerBuilder

        void buildFifo();

+       void buildHeadlessFifo();
+
   /* Pointer to the mixer instance we're building */
        Mixer* m_mixer;
 };
```

20 src/core/MixerBuilder.cpp View

```diff
@@ -94,8 +94,20 @@ void MixerBuilder::buildFifo()
 94 94                 m_mixer->m_framesPerPeriod = DEFAULT_BUFFER_SIZE;
 95 95                 m_mixer->m_fifo = new fifo( 1 );
 96 96         }
 97    -       else if( configManager::inst()->value( "mixer", "framesperaudiobuffer"
 98    -                                               ).toInt() >= 32 )
    97 +       else
    98 +       {
    99 +               buildHeadlessFifo();
   100 +       }
   101 +}
   102 +
   103 +
   104 +/**
   105 + * Fabrication of fifo of a mixer that is headless (no gui)
   106 + */
   107 +void MixerBuilder::buildHeadlessFifo() {
   108 +       int frames = configManager::inst()->value( "mixer", "framesperaudiobuffer").toInt()
   109 +
   110 +       if( frames >= 32 )
 99 111         {
100 112                 m_mixer->m_framesPerPeriod =
101 113                         (fpp_t) configManager::inst()->value( "mixer",
@@ -114,11 +126,9 @@ void MixerBuilder::buildFifo()
114 126         }
115 127         else
116 128         {
117    -               configManager::inst()->setValue( "mixer",
118    -                                                "framesperaudiobuffer",
   129 +               configManager::inst()->setValue( "mixer", "framesperaudiobuffer",
119 130                                 QString::number( m_mixer->m_framesPerPeriod ) );
120 131                 m_mixer->m_fifo = new fifo( 1 );
```

Change 8/8

Link : https://github.com/psyomn/lmms/commit/4faf51d2b90207feb4c5df0aa678a69aeee6cb0b

Finally we make the Engine use the builder instead of instantiating the Mixer directly.

```
+#include "MixerBuilder.h"
 #include "pattern.h"
 #include "PianoRoll.h"
 #include "PresetPreviewPlayHandle.h"
@@ -70,12 +71,13 @@

 void engine::init( const bool _has_gui )
 {
+        MixerBuilder mixerBuilder;
         s_hasGUI = _has_gui;

         initPluginFileHandling();

         s_projectJournal = new ProjectJournal;
-        s_mixer = new Mixer;
+        s_mixer = mixerBuilder.build();
         s_song = new song;
         s_fxMixer = new FxMixer;
         s_bbTrackContainer = new bbTrackContainer;
```