AI統合DAW - 技術仕様書・開発ガイド

対象読者: エンジニア、技術コンサルタント

バージョン: 1.0.0
最終更新日: 2025年6月19日

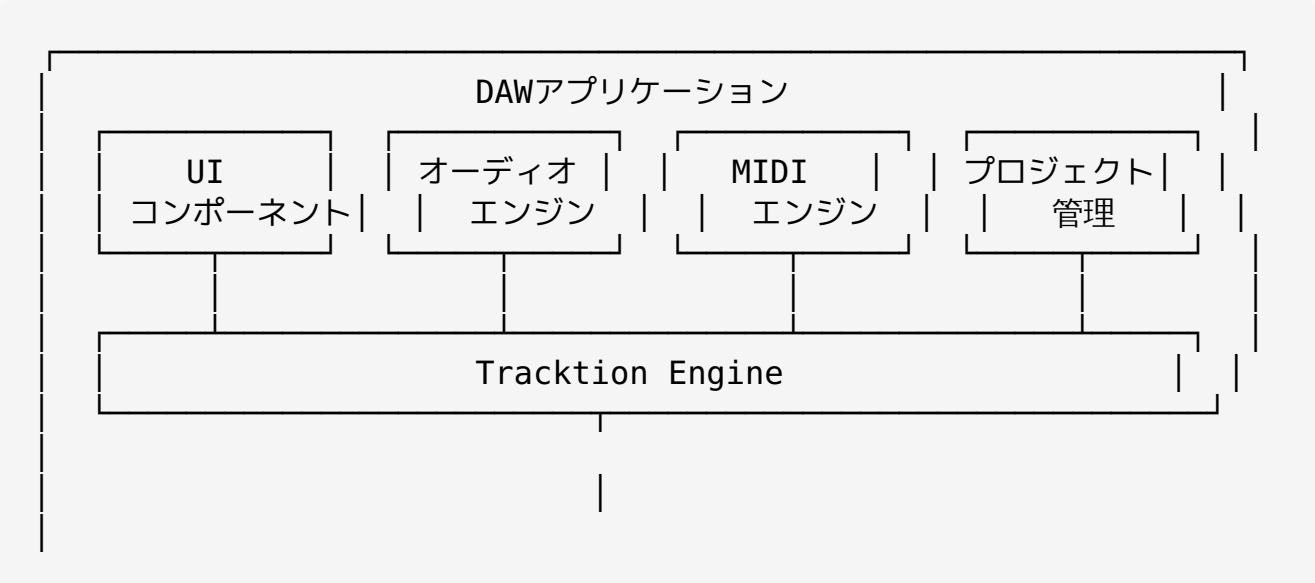
目次

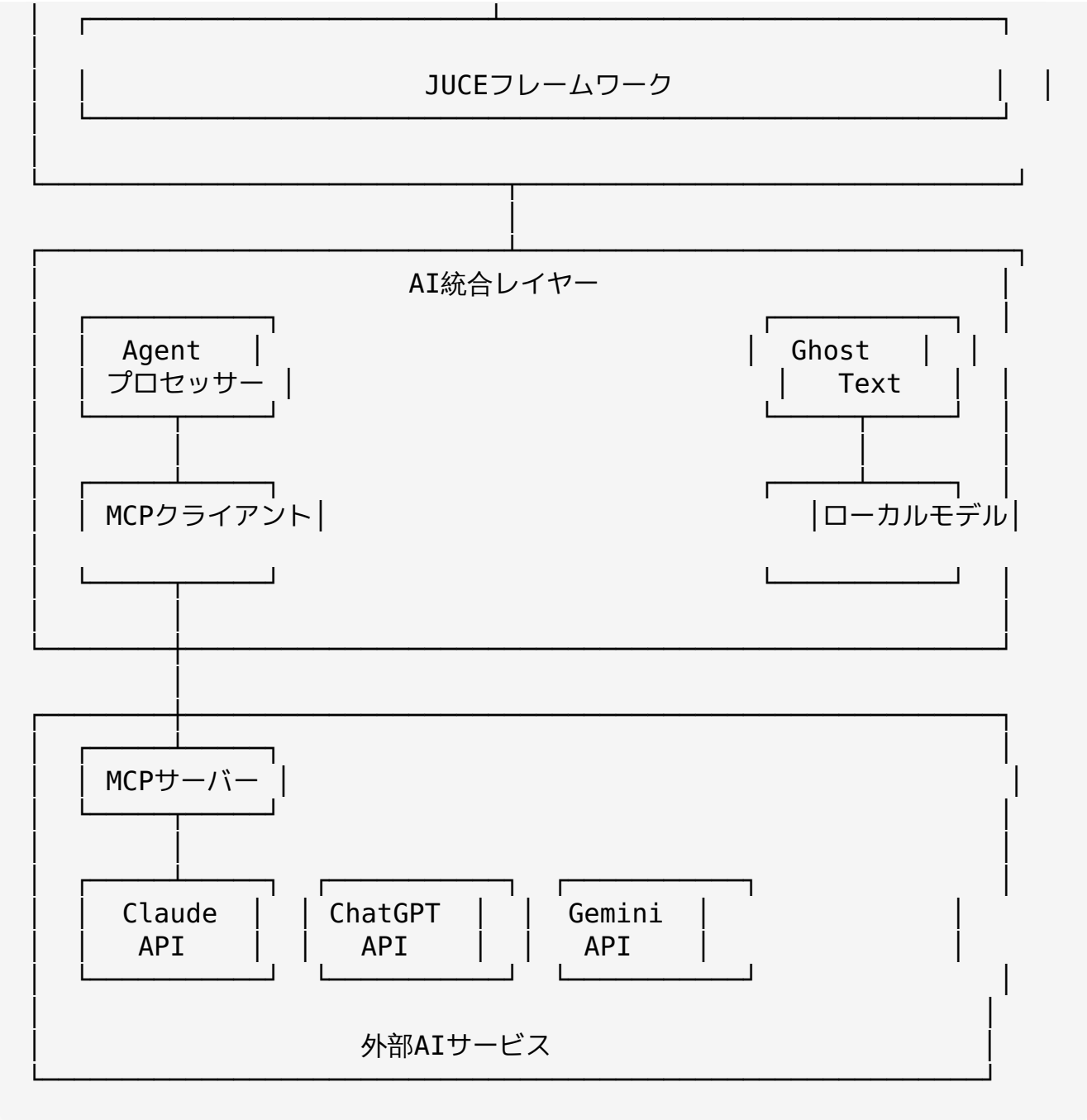
---

1. アーキテクチャ概要

1.1 システムアーキテクチャ

AI統合DAWは、以下の主要コンポーネントで構成されています:





1.2 技術スタック

1.2.1 フロントエンド

- **UI フレームワーク**: JUCE 7.x
- **グラフィックス**: JUCE Graphics API
- **オーディオビジュアライゼーション**: カスタムJUCEコンポーネント

1.2.2 バックエンド

- **オーディオエンジン**: Tracktion Engine
- **プラグインサポート**: VST3, AU, AAX
- **ファイル形式**: WAV, AIFF, MP3, FLAC, OGG
- **MIDI**: JUCE MIDI API

1.2.3 AI統合

- **Agent機能**: MCP (Model Context Protocol) サーバー
- **Ghost Text機能**: カスタムTransformerモデル (PyTorch)
- **AI API**: Claude API (Anthropic), ChatGPT API (OpenAI), Gemini API (Google)

1.2.4 インフラストラクチャ

- **認証**: Auth0
- **課金**: Stripe
- **クラウドストレージ**: AWS S3
- **分析**: Mixpanel/Amplitude

1.3 依存関係

依存関係	バージョン	用途
JUCE	7.x	クロスプラットフォームオーディオアプリケーションフレームワーク
Tracktion Engine	3.x	オーディオエンジンとDAW機能
PyTorch	2.x	Ghost Text用のTransformerモデル
MCP SDK	1.x	Model Context Protocol実装
RapidJSON	1.1.0	JSON解析
CURL	8.x	HTTP通信
SQLite	3.x	ローカルデータストレージ
OpenSSL	3.x	暗号化とセキュリティ

---

2. 開発環境セットアップ

2.1 必要なツール

- **OS**: Windows 11
- **IDE**: Visual Studio 2022
- **コンパイラ**: MSVC v143

- **ビルドシステム**: CMake 3.25+
- **バージョン管理**: Git 2.40+
- **パッケージマネージャー**: vcpkg

2.2 環境構築手順

2.2.1 前提条件のインストール

```
# Visual Studio 2022のインストール (Community Edition以上)
# 以下のワークロードを選択:
# - C++によるデスクトップ開発
# - ユニバーサルWindowsプラットフォーム開発

# Git のインストール
winget install --id Git.Git

# CMake のインストール
winget install --id Kitware.CMake

# Python 3.11 のインストール
winget install --id Python.Python.3.11

# vcpkg のセットアップ
git clone https://github.com/microsoft/vcpkg.git
cd vcpkg
.\bootstrap-vcpkg.bat
.\vcpkg integrate install
```

2.2.2 JUCEのセットアップ

```
# JUCEのクローン
git clone https://github.com/juce-framework/JUCE.git
cd JUCE
git checkout 7.0.5  # または最新の安定バージョン

# Projucerのビルド
cd extras/Projucer/Builds/VisualStudio2022
start Projucer.sln
# Visual Studioでビルド (Release構成)
```

2.2.3 Tracktion Engineのセットアップ

```
# Tracktion Engineのクローン
git clone https://github.com/Tracktion/tracktion_engine.git
```

2.2.4 プロジェクトのセットアップ

```
# プロジェクトリポジトリのクローン
git clone https://github.com/your-org/ai-daw.git
cd ai-daw

# 依存関係のインストール
.\scripts\install_dependencies.ps1

# CMakeプロジェクトの生成
mkdir build
cd build
cmake -G "Visual Studio 17 2022" -A x64 ..

# プロジェクトのビルド
cmake --build . --config Debug
```

2.3 開発環境の検証

```
# 単体テストの実行
cd build
ctest -C Debug

# サンプルプロジェクトの実行
.\Debug\AIDAWSample.exe
```

---

3. コードベース構成

3.1 ディレクトリ構造

```
ai-daw/
├── CMakeLists.txt          # メインCMakeファイル
├── assets/                 # アイコン、画像、フォントなど
├── docs/                   # ドキュメント
├── external/               # サードパーティライブラリ
│   ├── JUCE/               # JUCEサブモジュール
│   └── tracktion_engine/   # Tracktion Engineサブモジュール
├── scripts/                # ビルド・デプロイスクリプト
├── src/                    # ソースコード
│   ├── app/                # アプリケーションエントリポイント
│   ├── audio/              # オーディオエンジン関連
│   ├── midi/               # MIDI処理関連
│   ├── ai/                 # AI統合関連
│   │   ├── agent/          # Agent機能
```

```
│   │   └── ghost_text/       # Ghost Text機能
│   ├── ui/                   # UIコンポーネント
│   └── utils/                # ユーティリティクラス
├── tests/                    # テストコード
│   ├── unit/                 # 単体テスト
│   └── integration/          # 統合テスト
└── tools/                    # 開発ツール
```

3.2 コーディング規約

- **命名規則**:
- クラス: `PascalCase`
- メソッド: `camelCase`
- 変数: `camelCase`
- 定数: `UPPER_SNAKE_CASE`
- プライベートメンバ変数: `m_camelCase`
- **コードフォーマット**: Clang-Format (.clang-format ファイルに定義)
- **コメント**: Doxygen形式
- **エラー処理**: 例外とJUCE Result型の併用
- **メモリ管理**: スマートポインタ優先、RAII原則の遵守

3.3 バージョン管理戦略

- **ブランチ戦略**: GitFlow
- `main`: 安定リリース
- `develop`: 開発ブランチ
- `feature/*`: 機能開発
- `bugfix/*`: バグ修正
- `release/*`: リリース準備
- **コミットメッセージ**: Conventional Commits形式
- **バージョニング**: セマンティックバージョニング (MAJOR.MINOR.PATCH)

---

4. コアモジュール仕様

4.1 MainComponent

メインアプリケーションウィンドウを表すコンポーネント。

```
class MainComponent : public juce::Component,
                      public juce::ApplicationCommandTarget
{
```

```
public:
    MainComponent();
    ~MainComponent() override;

    // Component overrides
    void paint(juce::Graphics&) override;
    void resized() override;

    // ApplicationCommandTarget overrides
    ApplicationCommandTarget* getNextCommandTarget() override;
    void getAllCommands(juce::Array<juce::CommandID>&) override;
    void getCommandInfo(juce::CommandID,
juce::ApplicationCommandInfo&) override;
    bool perform(const InvocationInfo&) override;

private:
    // UI Components
    std::unique_ptr<TransportControls> m_transportControls;
    std::unique_ptr<TrackListComponent> m_trackList;
    std::unique_ptr<MixerComponent> m_mixer;
    std::unique_ptr<AgentProcessor> m_agentProcessor;
    std::unique_ptr<GhostTextEngine> m_ghostTextEngine;

    // Engine
    std::unique_ptr<AudioEngine> m_audioEngine;

    // License
    std::unique_ptr<LicenseManager> m_licenseManager;

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(MainComponent)
};
```

4.2 AudioEngine

Tracktion Engineを統合したオーディオ処理エンジン。

```
class AudioEngine
{
public:
    AudioEngine();
    ~AudioEngine();

    // Engine initialization
    bool initialize(const juce::String& applicationName);

    // Transport controls
    void play();
    void stop();
    void record();
```

```
    // Project management
    bool loadProject(const juce::File& file);
    bool saveProject(const juce::File& file);
    bool createNewProject();

    // Track management
    Track* addTrack(TrackType type);
    bool removeTrack(Track* track);

    // Audio device management
    juce::AudioDeviceManager& getAudioDeviceManager();

    // Engine access
    tracktion_engine::Engine& getEngine();
    tracktion_engine::Edit& getEdit();

private:
    std::unique_ptr<tracktion_engine::Engine> m_engine;
    std::unique_ptr<tracktion_engine::Edit> m_edit;

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(AudioEngine)
};
```

4.3 AgentProcessor

Agent機能を処理するクラス。MCPクライアントを使用してAIモデルと通信。

```
class AgentProcessor : public juce::Component,
                       public juce::TextEditor::Listener,
                       public juce::Button::Listener,
                       public juce::Timer
{
public:
    AgentProcessor(AudioEngine& audioEngine, LicenseManager&
licenseManager);
    ~AgentProcessor() override;

    // Component overrides
    void paint(juce::Graphics&) override;
    void resized() override;

    // TextEditor::Listener overrides
    void textEditorTextChanged(juce::TextEditor&) override;
    void textEditorReturnKeyPressed(juce::TextEditor&) override;

    // Button::Listener overrides
    void buttonClicked(juce::Button*) override;

    // Timer overrides
    void timerCallback() override;
```

```
    // Agent functionality
    void processPrompt(const juce::String& prompt);
    void applyGeneratedContent(const juce::String& content,
TrackType targetTrack);
    void cancelGeneration();

    // Settings
    void setModel(AIModel model);
    AIModel getModel() const;

private:
    AudioEngine& m_audioEngine;
    LicenseManager& m_licenseManager;
    std::unique_ptr<MCPClient> m_mcpClient;

    // UI Components
    juce::TextEditor m_promptEditor;
    juce::TextButton m_generateButton;
    juce::ComboBox m_modelSelector;
    juce::ProgressBar m_progressBar;

    // State
    bool m_isGenerating;
    juce::String m_currentPrompt;

    // Implementation details
    void setupUI();
    void parseGeneratedContent(const juce::String& content);
    bool checkLicenseForOperation();

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(AgentProcessor)
};
```

4.4 GhostTextEngine

Ghost Text機能を処理するクラス。ローカルのTransformerモデルを使用。

```
class GhostTextEngine : public juce::Timer
{
public:
    GhostTextEngine(AudioEngine& audioEngine);
    ~GhostTextEngine();

    // Initialization
    bool initialize();

    // Prediction
    void startPrediction();
    void stopPrediction();
```

```
    bool isPredicting() const;

    // Settings
    void setPredictionConfidence(float threshold);
    void setPredictionLookahead(int beats);
    void setModelPath(const juce::File& modelPath);

    // Callbacks
    void addListener(Listener* listener);
    void removeListener(Listener* listener);

    // Listener interface
    class Listener
    {
    public:
        virtual ~Listener() = default;
        virtual void predictionsUpdated(const
juce::Array<MidiPrediction>& predictions) = 0;
    };

    // Timer overrides
    void timerCallback() override;

private:
    AudioEngine& m_audioEngine;
    std::unique_ptr<PythonEnvironment> m_pythonEnv;

    // Model state
    bool m_isInitialized;
    bool m_isPredicting;
    float m_confidenceThreshold;
    int m_lookaheadBeats;
    juce::File m_modelPath;

    // Listeners
    juce::ListenerList<Listener> m_listeners;

    // Implementation details
    void processCurrentContext();
    juce::Array<MidiPrediction> runInference(const
juce::Array<MidiNote>& context);


JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(GhostTextEngine)
};
```

4.5 MCPClient

Model Context Protocol (MCP) クライアント実装。

```cpp
class MCPClient
{
public:
    MCPClient();
    ~MCPClient();

    // Connection
    bool connect(const juce::String& serverUrl);
    bool isConnected() const;
    void disconnect();

    // Model selection
    void setModel(AIModel model);
    AIModel getModel() const;
    juce::StringArray getAvailableModels();

    // Request handling
    juce::String sendPrompt(const juce::String& prompt);
    void sendPromptAsync(const juce::String& prompt,
std::function<void(const juce::String&)> callback);
    void cancelRequest();

    // Status
    bool isProcessing() const;
    float getProgress() const;

private:
    // Connection state
    juce::String m_serverUrl;
    bool m_isConnected;
    bool m_isProcessing;
    float m_progress;
    AIModel m_currentModel;

    // Network
    std::unique_ptr<juce::URL> m_endpoint;
    std::unique_ptr<juce::WebInputStream> m_stream;

    // Implementation details
    juce::String formatPrompt(const juce::String& rawPrompt);
    juce::String parseResponse(const juce::String& rawResponse);

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(MCPClient)
};
```

4.6 LicenseManager

ライセンス管理とサブスクリプション機能を処理するクラス。

```
class LicenseManager
{
public:
    LicenseManager();
    ~LicenseManager();

    // Initialization
    bool initialize();

    // License status
    bool isFeatureAvailable(Feature feature);
    bool isPremiumUser();
    juce::String getUserId();
    juce::Time getLicenseExpiryTime();

    // License operations
    bool activateLicense(const juce::String& licenseKey);
    bool deactivateLicense();
    bool refreshLicense();

    // Purchase flow
    void startPurchaseFlow(Feature feature);
    void restorePurchases();

    // Usage tracking
    void trackFeatureUsage(Feature feature);
    int getRemainingUsage(Feature feature);

    // Events
    void addListener(Listener* listener);
    void removeListener(Listener* listener);

    // Listener interface
    class Listener
    {
    public:
        virtual ~Listener() = default;
        virtual void licenseStatusChanged() = 0;
        virtual void purchaseCompleted(bool success, const
juce::String& message) = 0;
    };

private:
    // License state
    bool m_isInitialized;
    bool m_isPremium;
    juce::String m_userId;
    juce::Time m_expiryTime;
    std::map<Feature, int> m_usageCounters;

    // Network
```

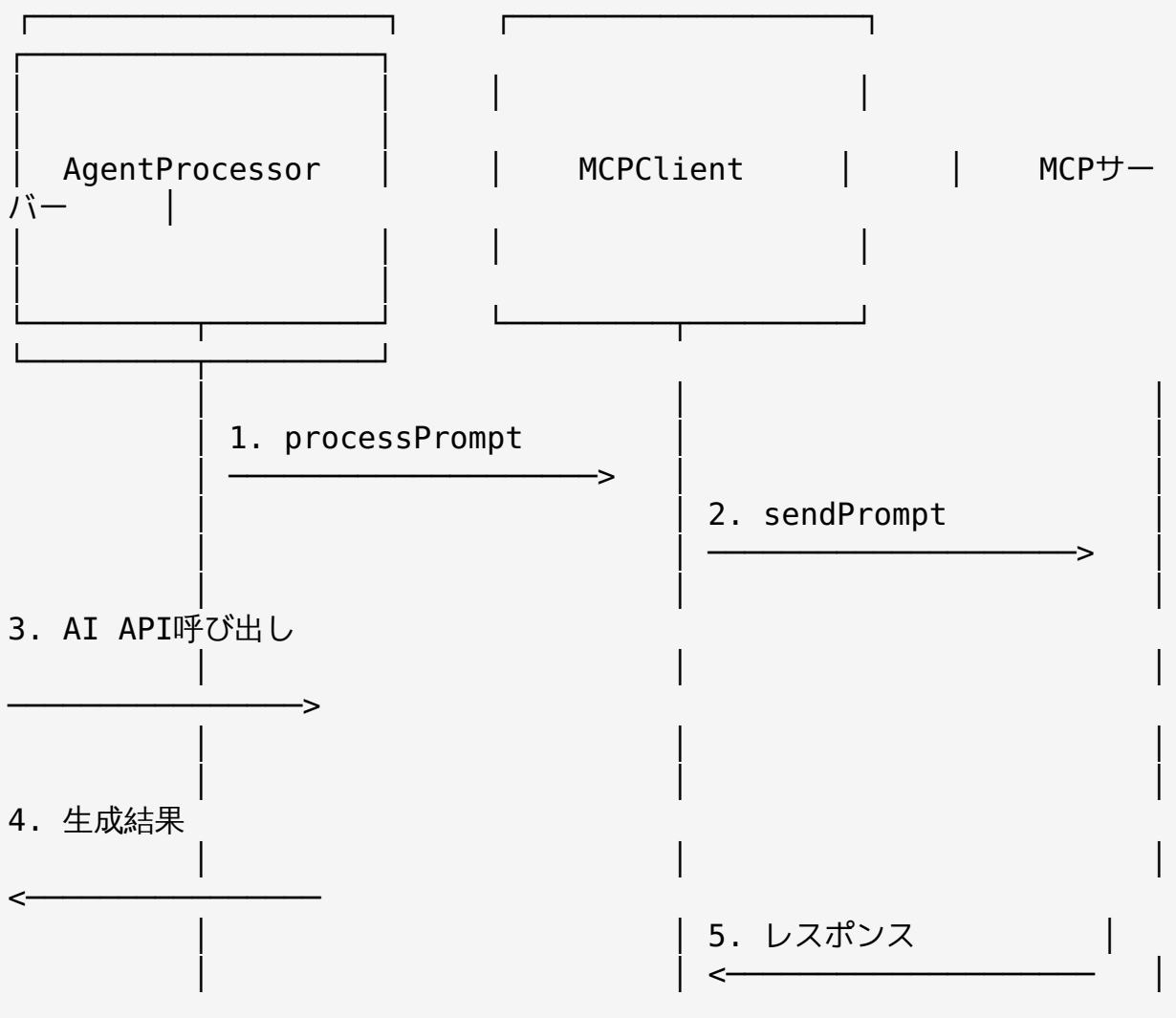
```
    std::unique_ptr<juce::URL> m_licenseServer;

    // Listeners
    juce::ListenerList<Listener> m_listeners;

    // Implementation details
    bool validateLicenseLocally();
    bool validateLicenseOnline();
    void loadLicenseFromDisk();
    void saveLicenseToDisk();

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(LicenseManager)
};
```

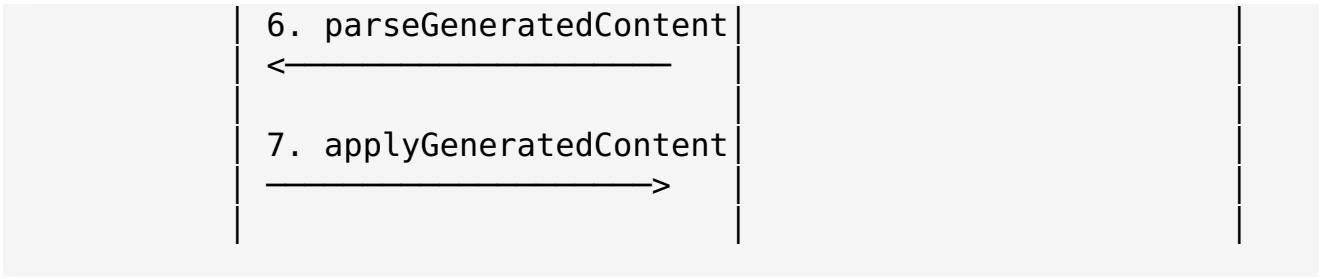
---

5. AI統合アーキテクチャ

5.1 Agent機能アーキテクチャ

Agent機能は、MCPサーバーを介して外部AIモデル（Claude、ChatGPT、Gemini）と通信し、自然言語指示から音楽要素を生成します。





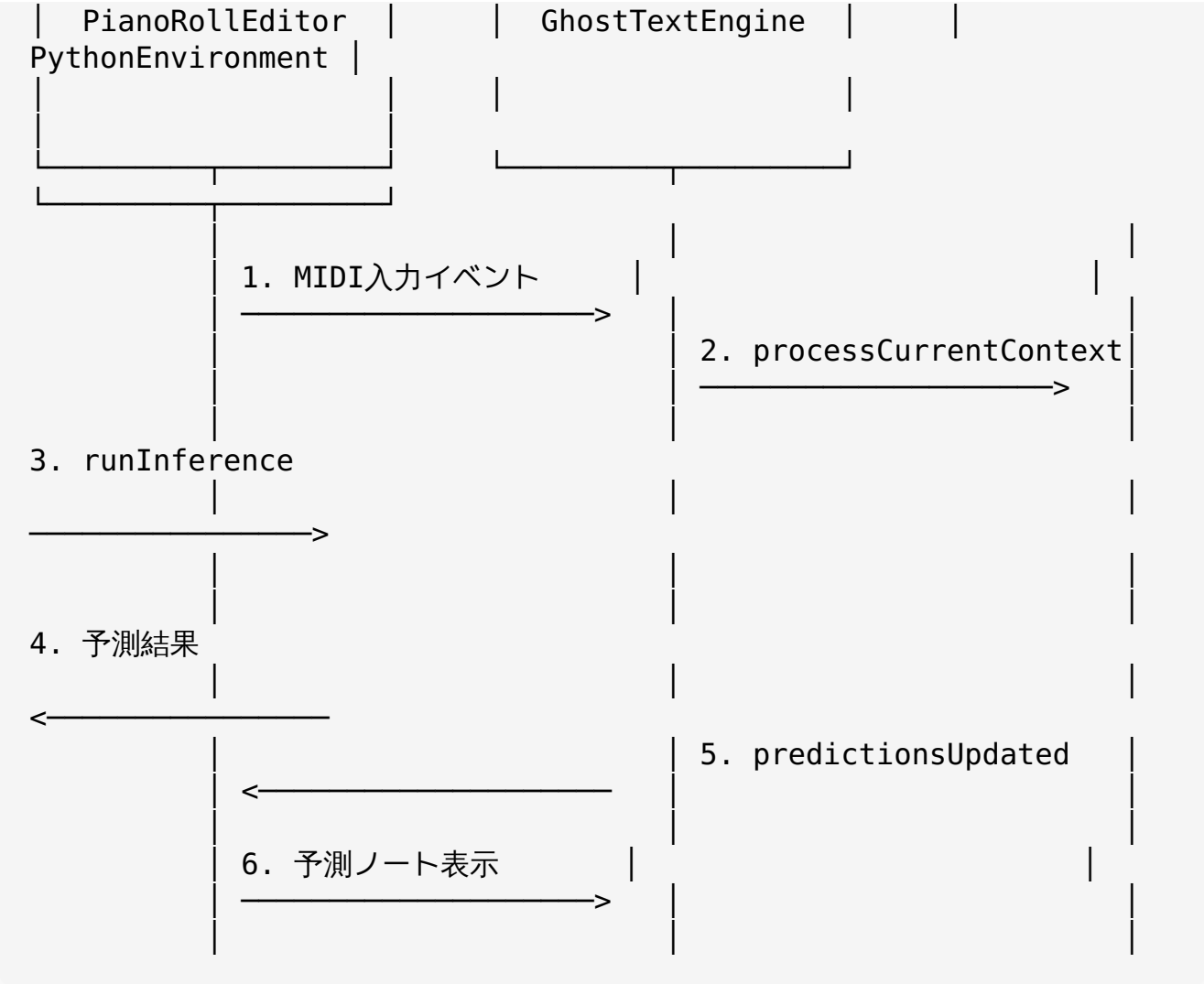
5.1.1 プロンプトエンジニアリング

Agent機能の効果的な動作のために、以下のプロンプトテンプレートを使用します:

```
あなたは音楽制作AIアシスタントです。以下の指示に基づいて、音楽要素を生成してください。

[指示]
{user_prompt}

[現在の曲情報]
テンポ: {tempo} BPM
キー: {key}
拍子: {time_signature}
ジャンル: {genre}

[出力形式]
以下の形式でJSONを出力してください：
{
  "type": "drum_pattern"|"bassline"|"chord_progression"|"melody",
  "notes": [
    {"pitch": 60, "start": 0.0, "duration": 0.25, "velocity": 100},
    ...
  ],
  "description": "生成した音楽要素の説明"
}

音楽理論的に正しく、指定されたテンポ、キー、拍子に合った音楽要素を生成してください。
```

5.2 Ghost Text機能アーキテクチャ

Ghost Text機能は、ローカルのTransformerモデルを使用して、ユーザーのMIDI入力をリアルタイムで予測・補完します。



5.2.1 モデルアーキテクチャ

Ghost Text機能で使用するTransformerモデルの概要:

- **モデルタイプ**: MIDI-Transformer
- **入力**: 過去のMIDIノートシーケンス（最大128ノート）
- **出力**: 次に演奏される可能性の高いノート（最大16ノート）
- **アーキテクチャ**:
- エンコーダー層: 6層
- デコーダー層: 6層
- 注意ヘッド: 8
- 埋め込みサイズ: 512
- **トレーニングデータ**: クラシック、ジャズ、ポップスの楽譜データセット
- **モデルサイズ**: 約200MB（量子化後）
- **推論速度**: 平均50ms（GPU使用時）

---

6. API仕様

6.1 MCPサーバーAPI

6.1.1 エンドポイント

- **ベースURL**: `http://localhost:8080/api/v1`

6.1.2 認証

- **タイプ**: Bearer Token
- **ヘッダー**: `Authorization: Bearer <token>`

6.1.3 エンドポイント一覧

モデル一覧取得

```
GET /models
```

レスポンス:

```
{
  "models": [
    {
      "id": "claude-3-opus",
      "name": "Claude 3 Opus",
      "provider": "anthropic",
      "capabilities": ["music_generation",
"pattern_recognition"]
    },
    {
      "id": "gpt-4",
      "name": "GPT-4",
      "provider": "openai",
      "capabilities": ["music_generation",
"pattern_recognition"]
    },
    {
      "id": "gemini-pro",
      "name": "Gemini Pro",
      "provider": "google",
      "capabilities": ["music_generation",
"pattern_recognition"]
    }
  ]
}
```

プロンプト送信

```
POST /generate
```

リクエスト:

```
{
  "model_id": "claude-3-opus",
  "prompt": "ファンキーなドラムパターンを4/4拍子で作成して",
  "context": {
    "tempo": 120,
    "key": "C Major",
    "time_signature": "4/4",
    "genre": "Funk"
  },
  "max_tokens": 1000
}
```

レスポンス:

```
{
  "id": "gen-123456",
  "result": {
    "type": "drum_pattern",
    "notes": [
      {"pitch": 36, "start": 0.0, "duration": 0.25, "velocity":
100},
      {"pitch": 42, "start": 0.5, "duration": 0.25, "velocity":
80},
      {"pitch": 38, "start": 1.0, "duration": 0.25, "velocity":
90},
      {"pitch": 42, "start": 1.5, "duration": 0.25, "velocity":
80},
      {"pitch": 36, "start": 2.0, "duration": 0.25, "velocity":
100},
      {"pitch": 42, "start": 2.5, "duration": 0.25, "velocity":
80},
      {"pitch": 38, "start": 3.0, "duration": 0.25, "velocity":
90},
      {"pitch": 42, "start": 3.5, "duration": 0.25, "velocity":
80}
    ],
    "description": "基本的なファンキードラムパターン。キックは1拍目と3拍
目、スネアは2拍目と4拍目、ハイハットは8分音符で刻みます。"
  }
}
```

生成状態確認

```
GET /generate/{id}/status
```

レスポンス:

```
{
  "id": "gen-123456",
  "status": "completed",
  "progress": 100
}
```

6.2 ライセンスサーバーAPI

6.2.1 エンドポイント

- **ベースURL**: `https://api.aidaw.com/license/v1`

6.2.2 認証

- **タイプ**: API Key
- **ヘッダー**: `X-API-Key: <api_key>`

6.2.3 エンドポイント一覧

ライセンス検証

```
POST /verify
```

リクエスト:

```
{
  "license_key": "XXXX-XXXX-XXXX-XXXX",
  "machine_id": "ABCDEF123456",
  "app_version": "1.0.0"
}
```

レスポンス:

```
{
  "valid": true,
  "user_id": "user-123456",
  "plan": "premium",
  "features": ["agent", "ghost_text", "advanced_effects"],
```

```
  "expiry_date": "2026-06-19T00:00:00Z",
  "usage_limits": {
    "agent_requests": 1000,
    "cloud_storage": 10240
  }
}
```

使用量報告

```
POST /usage
```

リクエスト:

```
{
  "license_key": "XXXX-XXXX-XXXX-XXXX",
  "machine_id": "ABCDEF123456",
  "feature": "agent",
  "count": 1
}
```

レスポンス:

```
{
  "success": true,
  "remaining": 999,
  "limit": 1000
}
```

---

7. ビルドプロセス

7.1 ビルド構成

構成	説明	最適化
Debug	デバッグシンボル付き、最適化なし	`/Od` (MSVC)
Release	最適化あり、デバッグシンボルなし	`/O2` (MSVC)
RelWithDebInfo	最適化あり、デバッグシンボルあり	`/O2 /Zi` (MSVC)

7.2 ビルドスクリプト

7.2.1 デバッグビルド

```
# デバッグビルド
cd build
cmake --build . --config Debug
```

7.2.2 リリースビルド

```
# リリースビルド
cd build
cmake --build . --config Release
```

7.2.3 インストーラービルド

```
# インストーラービルド
cd build
cmake --build . --config Release --target installer
```

7.3 依存関係の管理

依存関係は以下の方法で管理されます：

1. **CMake FetchContent**: JUCEとTracktion Engineの取得
2. **vcpkg**: その他のC++ライブラリ
3. **pip**: Python依存関係（Ghost Text機能用）

7.3.1 vcpkg依存関係

```
{
  "name": "ai-daw",
  "version": "1.0.0",
  "dependencies": [
    "curl",
    "rapidjson",
    "sqlite3",
    "openssl",
    "zlib"
  ]
}
```

7.3.2 Python依存関係

```
 torch==2.0.1
numpy==1.24.3
transformers==4.30.2
flask==2.3.2
```

---

8. テスト戦略

8.1 テストレベル

レベル	ツール	対象	自動化
単体テスト	Catch2	クラス、関数	CI/CD
統合テスト	カスタムフレームワーク	モジュール間連携	一部自動化
システムテスト	手動 + 自動UI	全体機能	一部自動化
パフォーマンステスト	プロファイラー	ボトルネック特定	一部自動化

8.2 テストカバレッジ目標

- **コアモジュール**: 90%以上
- **UI**: 70%以上
- **AI統合**: 80%以上
- **全体**: 75%以上

8.3 テスト自動化

CI/CDパイプラインでは以下のテストが自動実行されます:

1. 単体テスト
2. 静的コード解析
3. メモリリーク検出
4. ビルド検証
5. 基本的な統合テスト

8.4 テスト例

8.4.1 単体テスト例（Catch2）

```
TEST_CASE("MCPClient can parse response correctly", "[ai][mcp]")
{
    MCPClient client;

    SECTION("Valid JSON response")
    {
        juce::String rawResponse = R"({
            "type": "drum_pattern",
            "notes": [
                {"pitch": 36, "start": 0.0, "duration": 0.25,
"velocity": 100},
                {"pitch": 38, "start": 1.0, "duration": 0.25,
"velocity": 90}
            ],
            "description": "Basic pattern"
        })";

        juce::String result = client.parseResponse(rawResponse);

        REQUIRE(result.isNotEmpty());
        REQUIRE(result.contains("drum_pattern"));
    }

    SECTION("Invalid JSON response")
    {
        juce::String rawResponse = "Invalid JSON";

        juce::String result = client.parseResponse(rawResponse);

        REQUIRE(result.isEmpty());
    }
}
```

---

9. パフォーマンス最適化

9.1 オーディオパフォーマンス

9.1.1 オーディオスレッド

オーディオ処理は専用の高優先度スレッドで実行され、以下の最適化が適用されます：

- **ロックフリーデータ構造**: スレッド間通信に使用

- **メモリプール**: リアルタイムメモリ割り当てを回避
- **SIMD命令**: オーディオ処理の高速化（AVX2/SSE4.2）
- **バッファサイズ最適化**: レイテンシとCPU負荷のバランス

9.1.2 プラグイン処理

- **遅延補償**: プラグインの処理遅延を自動補正
- **プラグインサンドボックス**: クラッシュ時の分離
- **プラグインキャッシュ**: 起動時間短縮

9.2 AI機能のパフォーマンス

9.2.1 Agent機能

- **非同期処理**: UI応答性を維持
- **キャッシュ**: 類似プロンプトの結果を再利用
- **バックグラウンド生成**: 長時間の生成をバックグラウンドで実行

9.2.2 Ghost Text機能

- **モデル量子化**: 推論速度向上
- **バッチ処理**: 複数予測の効率的な処理
- **GPU加速**: 可能な場合はGPUを使用
- **コンテキスト最適化**: 予測精度と速度のバランス

9.3 UI最適化

- **レンダリングキャッシュ**: 複雑なUIコンポーネントの再描画最適化
- **遅延読み込み**: 必要に応じてリソースを読み込み
- **アニメーション最適化**: 60FPSを維持するための最適化
- **ダブルバッファリング**: 描画のちらつき防止

---

10. デプロイメント

10.1 パッケージング

10.1.1 Windows インストーラー

Windows用インストーラーはWiX Toolsetを使用して作成されます。

```
<!-- wix/Product.wxs -->
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
```

```
    <Product Id="*" Name="AI DAW" Language="1033"
Version="1.0.0"
             Manufacturer="Your Company" UpgradeCode="PUT-GUID-
HERE">
        <Package InstallerVersion="200" Compressed="yes"
InstallScope="perMachine" />
        <MajorUpgrade DowngradeErrorMessage="A newer version is
already installed." />
        <MediaTemplate EmbedCab="yes" />

        <Feature Id="ProductFeature" Title="AI DAW" Level="1">
            <ComponentGroupRef Id="ProductComponents" />
            <ComponentGroupRef Id="ProductShortcuts" />
        </Feature>

        <UIRef Id="WixUI_InstallDir" />
        <Property Id="WIXUI_INSTALLDIR" Value="INSTALLFOLDER" />

        <WixVariable Id="WixUILicenseRtf" Value="License.rtf" />
        <WixVariable Id="WixUIBannerBmp" Value="banner.bmp" />
        <WixVariable Id="WixUIDialogBmp" Value="dialog.bmp" />
    </Product>

    <!-- Directory structure, components, etc. -->
</Wix>
```

10.1.2 インストーラー作成スクリプト

```
# インストーラー作成
cd build
cmake --build . --config Release --target installer

# デジタル署名
signtool sign /tr http://timestamp.digicert.com /td sha256 /fd
sha256 /a ".\installer\AIDAW-1.0.0-win64.msi"
```

10.2 自動更新システム

アプリケーションは以下の自動更新システムを実装します:

1. **更新チェック**: 起動時と定期的に更新を確認
2. **差分更新**: 可能な場合は差分のみをダウンロード
3. **バックグラウンドダウンロード**: 更新をバックグラウンドでダウンロード
4. **再起動時適用**: アプリケーション再起動時に更新を適用

10.2.1 更新マニフェスト

```
{
  "version": "1.0.1",
  "release_date": "2025-07-01T00:00:00Z",
  "download_url": "https://download.aidaw.com/updates/
AIDAW-1.0.1-win64.msi",
  "patch_url": "https://download.aidaw.com/updates/AIDAW-1.0.0-
to-1.0.1.patch",
  "patch_size": 15000000,
  "full_size": 250000000,
  "sha256":
"abcdef1234567890abcdef1234567890abcdef1234567890abcdef1234567890",
  "release_notes": "バグ修正と機能改善",
  "min_version": "1.0.0",
  "is_mandatory": false
}
```

10.3 テレメトリとクラッシュレポート

アプリケーションは以下のテレメトリデータを収集します（ユーザーの同意を得た場合）：

1. **使用統計**: 機能の使用頻度
2. **パフォーマンスメトリクス**: CPU/メモリ使用率、レイテンシ
3. **クラッシュレポート**: スタックトレース、システム情報
4. **機能使用パターン**: ワークフロー分析

10.3.1 テレメトリ実装

```
class TelemetryManager
{
public:
    static TelemetryManager& getInstance();

    void initialize();
    void setEnabled(bool enabled);
    bool isEnabled() const;

    void trackEvent(const juce::String& category, const
juce::String& action, const juce::String& label = "", int value
= 0);
    void trackError(const juce::String& errorMessage, const
juce::String& errorDetails);
    void trackCrash(const juce::String& reason, const
juce::String& stackTrace);
    void trackPerformance(const juce::String& metric, double
value);
```

```
private:
    TelemetryManager();
    ~TelemetryManager();

    bool m_isEnabled;
    bool m_isInitialized;

    void sendData(const juce::var& data);


JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(TelemetryManager)
};
```

注: この技術仕様書は開発の進行に伴い更新されます。最新バージョンは常にプロジェクト管理システムで確認してください。