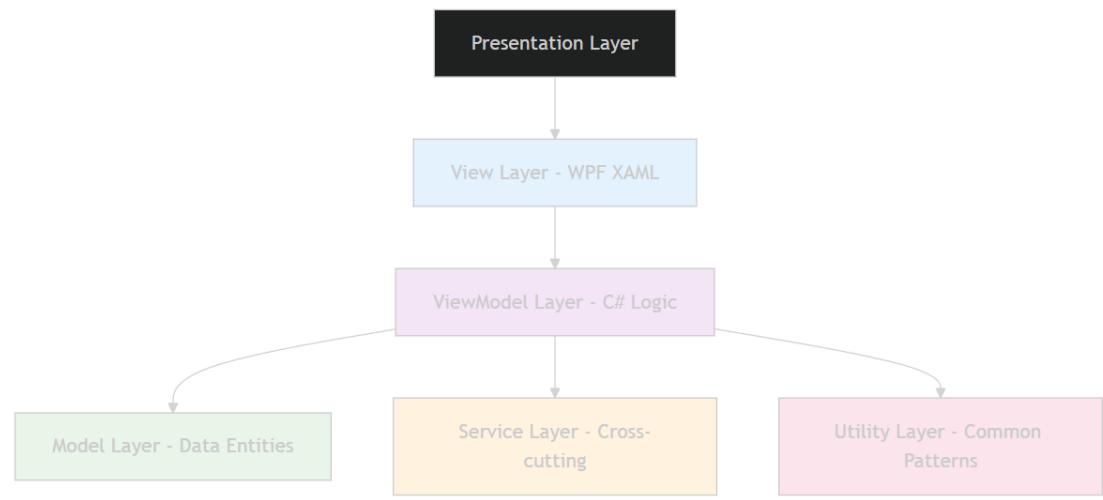


TaikoLite Rhythm Game System Design Report

System Architecture Overview

High-Level Design

The TaikoLite rhythm game follows a strict Model-View-ViewModel (MVVM) architectural pattern with clear separation of concerns. The system is organized into five distinct layers that communicate through well-defined interfaces and data binding.



Layered Architecture

View Layer : The View layer consists of WPF UserControls that define the user interface:

- TitleView: Entry screen with game title and start button
- SongSelectView: Song selection interface with list and details panel
- GameView: Core gameplay screen with note scrolling and real-time feedback
- ResultView: Post-game statistics display

Each View follows the Data Binding Pattern, connecting to corresponding ViewModels through bindings and commands.

Minimal code-behind is used, primarily for focus management and input event forwarding.

ViewModel Layer : ViewModels implement application logic and state management:

- TitleViewModel: Simple navigation to song selection
- SongSelectViewModel: Manages song collection and selection validation
- GameViewModel: Complex gameplay logic including timing, scoring, and note management
- ResultViewModel: Displays game results and provides navigation
- MainViewModel: Application root managing current view state

All ViewModels implement the Observer Pattern via `INotifyPropertyChanged` for reactive UI updates.

Model Layer : Business entities representing core game data:

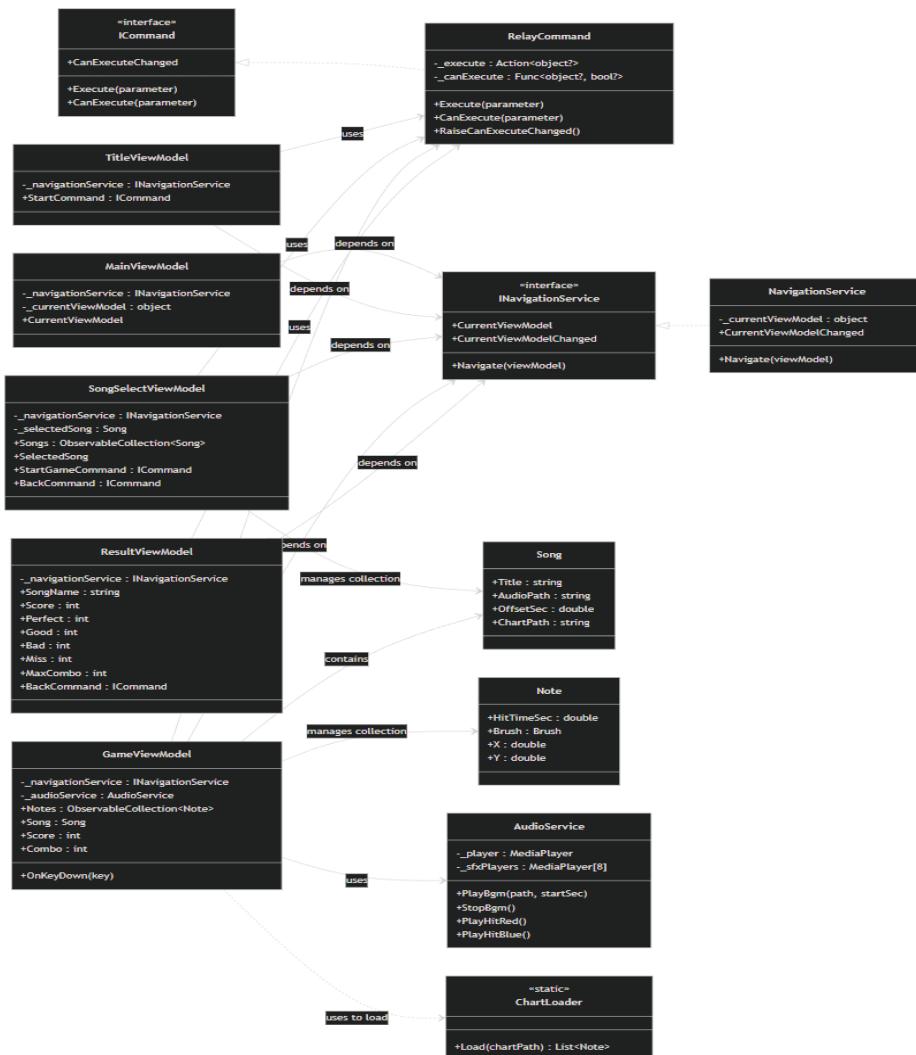
- Song: Immutable data class containing song metadata (title, audio path, chart path)
- Note: Observable object representing game notes with real-time position updates
- **Service Layer :** Services handle cross-cutting concerns and external resource management:

- NavigationService: Manages screen transitions using a simple state machine pattern
- AudioService: Handles audio playback with object pooling for sound effects
- ChartLoader: Static utility for parsing note chart files

Utility Layer

- RelayCommand: Generic implementation of the Command Pattern for MVVM command binding

Class Diagrams



Class Reuse and Extensibility

Reusable Components

- RelayCommand: Generic command implementation used across all ViewModels
- NavigationService Pattern: Abstracted navigation logic reusable in any WPF MVVM application
- AudioService with Object Pooling: Optimized audio playback system
- ChartLoader: Configurable file parser for different chart formats

Extensibility Points

- Song System: Easy addition of new songs through configuration
- Note Types: Extensible through inheritance for different note behaviors

- Game Modes: Base architecture supports multiple gameplay variations
- Input Systems: Abstracted input handling for different control schemes

Inheritance and Interface Implementation

- Note implements INotifyPropertyChanged for observable position updates
- NavigationService implements INavigationService interface
- All ViewModels use common patterns for property notification

Design Patterns Implemented

MVVM Pattern

- Separation of Concerns: Views handle UI, ViewModels handle logic, Models handle data
- Data Binding: Two-way binding connects UI elements to ViewModel properties
- Command Pattern: RelayCommand binds UI actions to ViewModel methods

Factory Pattern

- ChartLoader creates Note instances from chart file data
- Centralized object creation with validation

Service Pattern

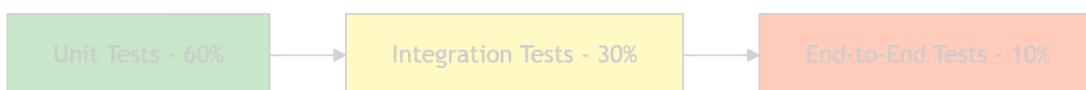
- NavigationService manages screen transitions
- AudioService handles all audio playback with object pooling
- Loose coupling through interface-based design

Object Pool Pattern

- AudioService maintains pool of 8 MediaPlayer instances
- Efficient sound effect playback during rapid gameplay
- Reduces garbage collection overhead

Testing Strategy

A three-tier testing approach ensures application reliability.



Unit Testing

- GameViewModel: Score calculation, timing logic, state transitions
- AudioService: Playback control, error handling, resource management
- ChartLoader: File parsing, data validation, note creation
- NavigationService: State management, event notification

Integration Testing

- View-ViewModel Integration: Data binding verification
- Service Integration: Audio timing with game loop synchronization

- Navigation Flow: Complete user journey testing
- Input Handling: Keyboard events to game actions

End-to-End Testing

- Complete game playthrough with various accuracy levels
- Audio synchronization across different system configurations
- Error recovery scenarios (missing files, invalid data)
- Performance testing under maximum note density

Technical Challenges and Solutions

1. Challenge: Audio Synchronization

Problem: Millisecond-accurate timing required between visual notes and audio playback.

Solution:

- Implemented configurable audio offset per song
- Used high-resolution Stopwatch for timing
- Leveraged MediaPlayer events for precise audio positioning
- Empirical calibration through gameplay testing

2. Challenge: Real-time Note Rendering

Problem: Smooth animation of multiple moving notes while maintaining 60 FPS.

Solution:

- Canvas-based rendering with manual coordinate management
- ObservableCollection with property change notifications
- Optimized game loop using DispatcherTimer
- Minimal UI updates through selective property binding

Development Process Challenges

- Agile Adaptation: Modified sprints and daily stand-ups for small team
- Self-Learning: Structured approach to WPF, MVVM, and game programming
- Integration Testing: Mock services for isolated component testing

Project Status Assessment

Baseline Requirements Fulfillment

- ✓ Complete game flow (Title → Song Select → Gameplay → Results)
- ✓ Real-time note scrolling with timing judgement
- ✓ Score calculation and combo tracking
- ✓ Audio playback with hit sound effects
- ✓ Polished UI with visual feedback

User-Friendliness

- Strengths: Intuitive navigation, immediate feedback, accessible controls
- Improvements: Tutorial system, difficulty grading, accessibility options

Robustness

- Error Handling: Graceful recovery from missing files, audio errors
- Performance: Consistent 60 FPS, efficient memory usage
- Reliability: Stable gameplay experience across different systems

Deployment Status

- Single executable with embedded resources
- No external dependencies beyond .NET Runtime
- Simple installation (extract and run)
- Compatible with Windows 10+ systems

Project Achievements

- Clean Architecture: Well-structured MVVM implementation
- Pattern Application: Multiple design patterns appropriately applied
- Robust Implementation: Error-resistant with good performance
- Extensible Design: Foundation for future enhancements
- Learning Demonstration: Practical software engineering application

Supplementary Resources

- [1] Microsoft Corporation, "WPF Overview," Microsoft Docs, 2023. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/desktop/wpf/>.
- [2] J. Smith, "Patterns: WPF Apps With the Model-View-ViewModel Design Pattern," *MSDN Magazine*, Feb. 2009. [Online]. Available: <https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern>.
- [3] R. Nystrom, "Game Programming Patterns: Game Loop," 2014. [Online]. Available: <https://gameprogrammingpatterns.com/game-loop.html>.
- [4] J. Goddard, "Keeping the Beat: A Guide to Rhythm Game Programming," *Game Developer*, 2021. [Online]. Available: <https://www.gamedeveloper.com/audio/keeping-the-beat-a-guide-to-rhythm-game-programming>.
- [5] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1994.
- [6] Microsoft Corporation, "C# Programming Guide," Microsoft Docs, 2023. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/>.
- [7] Microsoft Corporation, "MediaPlayer Class," .NET API Browser, 2023. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/api/system.windows.media.mediaplayer>. [Accessed: Nov. 2024].