Software and Architecture Design Specification

Architecture Overview

The software system follows an object-oriented approach with a Model-View-Controller design and a layered architecture. The layered architecture provides a clear separation of responsibilities and allows for flexibility in incorporating different user interfaces, fitness models, and selection models within the genetic algorithm. The system is designed to be modular and extensible, promoting code reuse and maintainability.

Architectural Decisions

1. Layered Architecture: The decision to adopt layered architecture allows for the separation of concerns and modular development. It facilitates the clear division of responsibilities between components and promotes flexibility in accommodating different UI implementations, fitness models, and selection models.

2. Model-View-Controller (MVC) Design: The MVC pattern is employed to separate the application logic into three interconnected components: the Model, View, and Controller. This design promotes loose coupling, enhances maintainability, and supports the implementation of different UI frameworks.

System Structure

The system consists of several key components that assist in executing the genetic algorithm:

1. Chromosome: The Chromosome object represents the low-level genetic material and handles its own drawings. It interacts with other system components for calculations and processing.

2. Population: The Population object manages the storage of Chromosomes, calculates metrics such as average fitness, best fitness, worst fitness, and hamming distance, and provides this information to high-level components.

3. Fitness and Selection Methods: The system incorporates various fitness methods, including Simple, Matching, Consecutive, and Alternating, as well as selection methods such as Diversity, Worst, Rank, Roulette, and Truncation. The

Strategy pattern is implemented to support flexible interchangeability of these methods.

4. Presentation Layer: The presentation layer includes components such as Chromosome Viewer, JavaFXLauncher, EvolutionFX, JSwingLauncher, and EvolutionViewer. These components interact with the domain objects, including ChromosomeComponent, EvolutionComponent, FittestComponent, and PopulationComponent, to draw objects on the graphical user interface (GUI) and perform actions by calling methods within the objects.

Design Risks

The following design risks should be considered and mitigated during the development process:

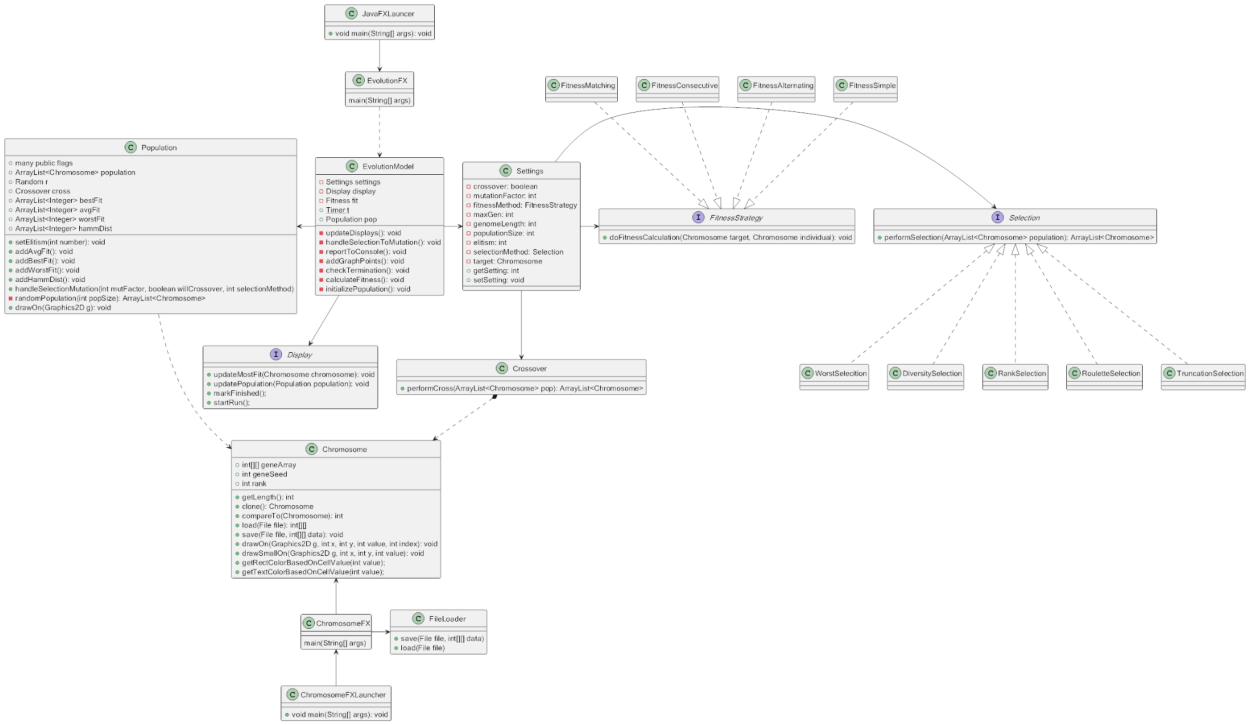
1. Complexity of Fitness and Selection Methods: The implementation and integration of various fitness and selection methods may introduce complexity and impact the system's performance and maintainability. Careful design and testing should be undertaken to ensure their proper functionality.

2. UI Flexibility: The system's flexibility to accommodate different UI frameworks and fitness models introduces the risk of increased complexity and potential inconsistencies. Thorough testing and validation are required to ensure seamless integration and user experience.

3. Extensibility and Modularity: The layered architecture and object-oriented design aim to promote extensibility and modularity. However, improper design decisions or lack of adherence to these principles could hinder future enhancements and maintainability. Regular code reviews and adherence to design guidelines are essential to mitigate this risk.

The tradeoff of using a layered architecture with MVC for the genetic algorithm project lies between the benefits of modularity, flexibility, and clear separation of concerns against the potential drawbacks of increased complexity and performance overhead. The decision to use this architecture should consider the project's size, complexity, anticipated future changes, and developer familiarity and expertise with this architectural style.

Ultimately, the use of a layered architecture with MVC can provide long-term benefits in terms of maintainability, extensibility, and scalability, especially when the project requires accommodating different UI options, fitness models, and selection models.


JavaFXLauncer
+ void main(String[] args): void
EvolutionFX
main(String[] args)
FitnessMatching
FitnessConsecutive
FitnessAlternating
FitnessSimple
Population
○ many public flags
○ ArrayList<Chromosome> population
○ Random r
○ Crossover cross
○ ArrayList<Integer> bestFit
○ ArrayList<Integer> avgFit
○ ArrayList<Integer> worstFit
○ ArrayList<Integer> hammDist
+ setElitism(int number): void
+ addAvgFit(): void
+ addBestFit(): void
+ addWorstFit(): void
+ addHammDist(): void
+ handleSelectionMutation(int mutFactor, boolean willCrossover, int selectionMethod)
- randomPopulation(int popSize): ArrayList<Chromosome>
+ drawOn(Graphics2D g): void
EvolutionModel
□ Settings settings
□ Display display
□ Fitness fit
□ Timer t
○ Population pop
- updateDisplays(): void
- handleSelectionToMutation(): void
- reportToConsole(): void
- addGraphPoints(): void
- checkTermination(): void
- calculateFitness(): void
- initializePopulation(): void
Settings
□ crossover: boolean
□ mutationFactor: int
□ fitnessMethod: FitnessStrategy
□ maxGen: int
□ genomeLength: int
□ populationSize: int
□ elitism: int
□ selectionMethod: Selection
□ target: Chromosome
○ getSetting: int
○ setSetting: void
FitnessStrategy
+ doFitnessCalculation(Chromosome target, Chromosome individual): void
Selection
+ performSelection(ArrayList<Chromosome> population): ArrayList<Chromosome>
Display
+ updateMostFit(Chromosome chromosome): void
+ updatePopulation(Population population): void
+ mainFinished();
+ startRun();
Crossover
+ performCross(ArrayList<Chromosome> pop): ArrayList<Chromosome>
WorstSelection
DiversitySelection
RankSelection
RouletteSelection
TruncationSelection
Chromosome
○ int[][] geneArray
○ int geneSeed
○ int rank
+ getLength(): int
+ clone(): Chromosome
+ compareTo(Chromosome): int
+ load(File file): int[][]
+ save(File file, int[][] data): void
+ drawOn(Graphics2D g, int x, int y, int value, int index): void
+ drawSmallOn(Graphics2D g, int x, int y, int value): void
+ getRectColorBasedOnCellValue(int value);
+ getTextColorBasedOnCellValue(int value);
ChromosomeFX
main(String[] args)
FileLoader
+ save(File file, int[][] data)
+ load(File file)
ChromosomeFXLauncher
+ void main(String[] args): void